Design and Evaluation of Multi-Objective Online Scheduling Strategies for Parallel Machines using Computational Intelligence

Doctorial Thesis
Carsten Franke

Genehmigte Dissertation zur
Erlangung des akademischen Grades eines Doktors an der
Fakultät für Elektrotechnik und Informationstechnik
Universität Dortmund

Abteilung Informationstechnik
Institut für Roboterforschung

15.11.2006

Acknowledgements

It is my pleasure to thank all the people who supported me to make this thesis possible. In particular, I would like to express my thanks to my doctorial advisor, Prof. Dr.-Ing. Uwe Schwiegelshohn, who supervised my research work and encouraged me every time. I also want to thank Prof. Dr.-Ing. Hans-Paul Schwefel for his helpful support.
Further, I wish to thank all members of the Computer Engineering Institute at the University of Dortmund for providing a pleasant and always helpful working atmosphere. I am especially grateful to my research colleagues, Joachim Lepping, Volker Hamscher, Alexander Papaspyrou, Jörg Platte, Peter Resch, Lars Schley, Baiyi Song, Achim Streit, and Ramin Yahyapour with whom I had the pleasure to work with. I am also deeply indebted to all undergraduate students who contributed to my research work.
Finally, I would further like to thank Claudia for all her help, patience, and love.

1. Advisor:	Prof. Dr.-Ing. Uwe Schwiegelshohn
2. Advisor:	Prof. Dr.-Ing. Hans-Paul Schwefel
Tag der Einreichung:	31. May 2006
Tag der Prüfung:	31. October 2006
Stand vom:	15. November 2006

Abstract

This thesis presents a methodology for automatically generating online scheduling algorithms for a complex objective defined by a machine provider. Such complex objective functions are required if the providers have several simple objectives. For example, the different relationships to the various users must be incorporated during the development of appropriate scheduling algorithms. Our research is focused on online scheduling with independent parallel jobs, multiple identical machines and a small user community. First, Evolutionary Algorithms are used to exemplarily create a 7-dimensional solution space of feasible schedules of a given workload trace. Within this step no preferences between different basic objectives need to be defined. This solution space enables the resource providers to define a complex evaluation objective based on their specific preferences. Second, optimized scheduling strategies are generated by using two different approaches. On the one hand, an adaptation of a Greedy scheduling algorithm is applied which uses weights to create an order of jobs. These job weights are extracted again from workload traces with the help of Evolutionary Algorithms. On the other hand, a Fuzzy rule based scheduling system will be applied. Here, we classify a scheduling situation which consists of many parameters like the day time, the week day, the waiting queue length etc. Depending on this classification, a Fuzzy rule based system chooses an appropriate sorting criterion for the waiting job queue and a suitable scheduling algorithm. Finally, both approaches, the Greedy scheduling strategy and the Fuzzy rule based scheduling system, are compared by using again workload traces. The achieved results demonstrate the applicability of our approach to generate such multi-objective scheduling strategies.

Contents

Chapter 1

Introduction

Scheduling is the task of assigning rare machines over time to given problems or tasks that are often called jobs. The generation of a flight schedule by allocating airplanes and airplane crews to flight connections with a specific flight time is one example of such scheduling problems, see Vance et al. [158, 159]. However, the number of application fields where scheduling is important to increase the system performance is very wide. This ranges from parallel computers, Grid Computing, logistics, telecommunications, and production processes to mobil devices.

Both the jobs and the machines may have special properties. For example, jobs can require specific processing times and machines may have a limited capacity. A schedule is called valid if during the assignment of the available machines to the existing jobs all job and machine properties as well as additional constraints are satisfied. Such constraints may be of various types and may apply to jobs and to machines. Examples for the above mentioned flight scheduling problem are the rest times for the airplane crews or the technical service intervals for the airplanes. Properties are always related to single jobs or machines and can be verified individually, while constraints always describe dependencies between several jobs or machines within the resulting schedule. Therefore, the verification of constraints is more complex than the examination of properties. However, the constraints in most real life scenarios can be verified easily as the dependencies between jobs and resources are usually not too complex.

The scheduling problem is the selection of the best schedule of all valid schedules. To this end, an objective function is needed to compare different possible schedules. Many problems that are discussed in the literature only use simple scalar evaluation objectives to determine the quality of schedules. Such simple objectives are, for example, the makespan [66], which is defined as the latest completion time of a fixed job set, or the sum of the weighted completion [89] or response times [163] of all jobs.

However, in many real life scenarios, the machine providers would like to optimize more than one objective. For example, they do not want to process all jobs in the same way as those jobs compete for the available machines. Some of the jobs might be more important than others as the corresponding users or user groups have special relationships to the machine providers. Furthermore, the machine providers often want to distinguish between different kinds of jobs depending on their resource demand [116].

The general problem that we address in this work is depicted in Figure 1.1. Here, different users or customers with varying goals are submitting jobs to a system over time. The sequence of those jobs forms the overall user demand. The scheduling system is responsible to assign this demand to the available machines. During this process, the scheduling strategy needs to resolve conflicts between competing jobs. The overall assignment generates the resulting schedule.

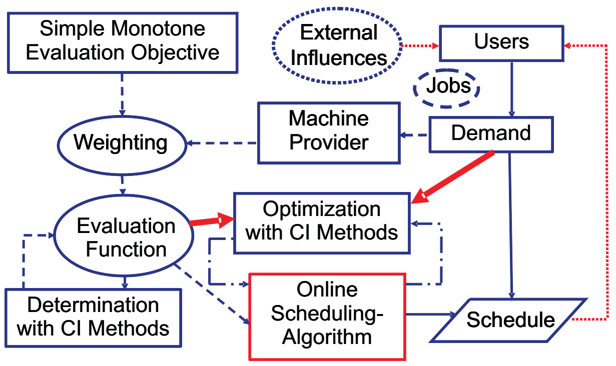

Simple Monotone Evaluation Objective
External Influences
Users
Jobs
Weighting
Machine Provider
Demand
Evaluation Function
Optimization with CI Methods
Online Scheduling-Algorithm
Schedule
Determination with CI Methods

Figure 1.1: Basic problem description. The two thick arrows specify the input for the optimization of the online scheduling algorithm. The solid lines specify the online scheduling case. The dashed lines explain the dependencies in the real environment and during the algorithm development. The two dotted lines represent the long term feedback on the users.

The scheduling system for the assignment of machines to jobs can be implemented by the machine providers. However, the providers in most cases are not able to develop new scheduling systems themselves. Hence, they only choose between several existing scheduling algorithms and customize them. The resulting strategies often do not fit to the local machines. Therefore, this thesis presents a methodology to automatically generate appropriate scheduling strategies that fit to the local demand. The machine providers are aware of the user demand and the resulting schedule. In the single optimization case, the resulting schedule is only evaluated using a simple monotone evaluation objective, like the sum of the weighted response times. However, as described above, the machine providers would like to use more complex objective functions. These functions should be generated by weighing the underlying simple evaluation objectives. However, the weighing is not trivial as the providers are normally not aware of the set of possible schedules and the relation to the schedules that can be achieved by using standard scheduling strategies. Therefore, the machine administrators cannot provide such a combined objective function. Thus, our methodology first generates the Pareto front of possible schedules with multi-objective Evolutionary Algorithms, see Section 3.2. This allows the definition of more complex objective functions by specifying preferred regions within the resulting Pareto front and generating corresponding objective functions. Finally, the machine providers need a method that generates scheduling strategies based on the individual objectives. To this end, single-objective Evolutionary Algorithms are used to parameterize existing scheduling strategies. The user demand and the specified complex objective function are used during this optimization procedure.

Figure 1.1 further includes external influences and feedback of the various users resulting from the last scheduling decisions. Both items highly influence the long term user satisfaction. Especially the feedback is of high importance and might result in a changed user behavior for following job submissions. However, the feedback as well as external influences, e.g. power failures of the parallel computers or certain holidays, are neglected in our work. These influences highly depend on the local environment. Therefore, it is difficult to deduce and model such influences.

The presented research exemplarily focuses on scheduling in two application areas. First, the design and development of scheduling strategies for high performance parallel computers will be presented. This is the main part of this thesis where all concepts will be demonstrated. Second, the scheduling within Computational Grids will be discussed and evaluated.
High performance parallel computers are still becoming significantly faster. The speed of the processors and the interconnection network is increasing. Nevertheless, the computing power is still a limited resource. This is caused by many new applications that can be computed on nowadays computers and on the growing execution time of existing programs as the input data size and complexity increase. Therefore, the management and scheduling of such machines is important for the users and providers as such machines are rare and expensive. The growing network infrastructure world-wide enables the users to freely choose between existing high performance computers. This results in a changed user behavior on local machines. Therefore, the local scheduling must be adapted in order to avoid the migration of local users, as this would normally reduce the profit of the local machine provider. To this end, new scheduling strategies have to be developed that are able to include the user demand and the machine provider objectives.
The example of high performance parallel computers fits to the general problem structure. In this case, single processor nodes represent the machines that can be used to calculate computational tasks (jobs) of certain users individually. Each job is characterized by the degree of parallelism, the execution time and further criteria, see Feitelson and Nitzberg [56]. Several high performance computer centers provide job traces from the past that include detailed information about the type and structure of the executed computational tasks. Those so called workloads are available in the Standard Workload Format [54], described by Chapin et al. [20]. Hence, such data are used during the development of appropriate scheduling strategies.
The scheduling of high performance parallel computers is an online problem as the precise execution times of all jobs are not known in advance. Furthermore, information about future jobs is not available [73]. Schedules must be rearranged frequently. The different users or their computational agents use the resulting scheduling information given by the system to decide where to execute the individual jobs [17]. Additionally, the number of available nodes within such high performance computers is usually high. As the generation of new schedules is initiated frequently, the corresponding processing time is short. Accordingly, iterative or quasi-offline scheduling methods cannot be used in this scenario. Thus, online scheduling strategies with a short processing time must be developed that work well in most of the cases.
Because of the limited processing time, existing online scheduling systems at real installations mainly use a Greedy scheduling [70] in combination with First Come First Serve [157] methods and Backfilling [99, 68]. Those methods produce a very low utilization in the worst case [138], but work well and fast in practice [59]. Other scheduling strategies with an often higher execution time are not well established at real installations. For example, a rule based scheduling system could work well for high performance parallel computers, but is not used in practice. In the last years, market oriented scheduling approaches have been published [18]. Those methods use very flexible evaluation objectives and are able to adapt to variations of the input stream. However, existing research projects are limited to two simple linear objective functions [17].
The scheduling strategies used at real installations, as the First Come First Serve mentioned above, do not provide prioritizations of single jobs or job groups. Therefore, such systems are not able to optimize schedules according to any evaluation function. The prioritization of certain user groups at real installations is implemented by quotas [154] or by partitioning [55]

the existing parallel computers. A partition can then exclusively be assigned to such a user group [82]. This normally reduces the average utilization of the corresponding machine, see Feitelson [55].

The second application presented in this document is the scheduling of Computational Grids. Within such a Computational Grid many high performance computers are connected via a high performance interconnection network [62]. The participating computing sites are often geographically distributed. In comparison to a single parallel computer the Grid enables the users to execute jobs that cannot be processed locally. At the same time the location of the executing machines is in some sense hidden from the users. Furthermore, more complex tasks can be calculated by combining several machines from different parallel computers at the same time. The various high performance computers within such a Grid are still owned and maintained by different providers. Thus, each of these providers might have an individual scheduling policy and, correspondingly, a different scheduling objective. Therefore, the Grid scheduling tasks must cooperate with the different local scheduling strategies. For this reason, the Grid scheduling task is focused on the discovery of possible machines and on the distribution of the computational jobs to those machines. Both problems are complex as the machines in a Grid are heterogeneous and jobs may prefer certain types of machines. Furthermore, the commercial relationships between the participating users and machine providers are complex and not known to the Grid scheduling strategy. Additionally, the usage of certain machines might be limited to a smaller user group. The overall scheduling strategy is therefore based on heuristics and can be divided into three main steps. First, appropriate machines are located. Second, the job preferences are combined with the machine information in order to extract the best combination. Finally, the job is started on the selected machines. The presented Grid scheduling strategies differ in the way they explore the available machines. Furthermore, the different strategies use different procedures to select the machines that have to execute the individual jobs. During our Grid scheduling development, we assume that the local scheduling strategies do not vary. The scheduling strategies that are developed during the first part of this document represent possible local scheduling strategies within the Grid scenario.

Document Structure

This document is divided into three parts with 12 chapters. The first part presents the state of the art in the areas of scheduling, Evolutionary optimization, and parallel job scheduling. The second part presents the main part of this thesis and details the development of multi-objective online scheduling strategies for parallel machines. The last part presents our scheduling approaches within the Grid context.

Chapter 2 introduces the single and multi-objective scheduling in detail. Moreover, this chapter provides an overview of all existing multi-objective approaches known to the author that use more than 2 objectives. Chapter 3 introduces the concept of Evolutionary optimization that is applied in the later chapters of this thesis. Especially Evolution Strategies are described in more detail. Additionally, the used multi-objective Evolutionary Algorithms are introduced. The first part of this work ends with the state of the art overview of scheduling parallel computers in Chapter 4.

The second part of this thesis first introduces a method to scale workload traces from real existing machines in Chapter 5. Those scaled traces are then used to develop our multi-objective scheduling strategies. Chapter 6 describes our methodology to automatically generate schedul-

ing strategies for parallel machines in detail. This is followed by Chapter 7 that includes our approach to generate the set of possible schedules based on several simple scheduling objectives. In Chapter 8, we describe our approach to individually extract the machine provider's complex scheduling objective. Furthermore, we exemplarily choose such a complex objective without the loss of generality in order to demonstrate the strength of our approach within the following scheduling strategy development. Chapter 9 presents six different approaches to develop scheduling strategies that optimize the complex scheduling objective of the machine providers. The achieved quality of all six strategies is evaluated in Chapter 10.
The third part consists of Chapter 11 where we detail all of our Grid scheduling approaches. This document ends in Chapter 12 with a brief conclusion and an outlook of possible extensions in future works.

Part I

State of the Art

Chapter 2

Scheduling Systems

In this chapter, we focus on scheduling problems in general and possible solution methods. This serves as the necessary background for the development of the advanced scheduling strategies for parallel computers later in this work. Scheduling is the decision process of assigning one or several machines to jobs over time. A scheduling problem only occurs if several jobs are competing for the available machines at the same time. Here, the scheduling system needs to select the best schedule of all valid schedules. The selection of the best schedule uses an objective function that ranks all generated schedules. This objective function is normally defined by the machine provider or administrator. Since often heuristics are applied, the process of generating the final schedule does not necessarily result in the optimal schedule.
The scheduling system does not affect the results of the individual jobs but significantly influences the efficiency of the whole system. Note that the efficiency is reflected by the provider defined objective function and does not necessarily correspond with a high system utilization. From the provider's perspective, the increase in efficiency can be compared with having more machines to process the available jobs.
Scheduling problems occur in a wide area of applications, as mentioned in Chapter 1, and have been studied for a long time. As existing scheduling systems from different areas often have similarities in terms of the problem structure and constraints, a common framework and notation can typically be created to describe those scheduling problems. Based on this common problem description the complexity and corresponding algorithms can be described easily.
In the following section, the framework and notation for scheduling problems will be introduced. Afterwards, the schedules are classified followed by a brief overview of the complexity of scheduling problems. Next, single and multi-objective scheduling problems and related solution strategies are introduced. This chapter ends with a brief description of the design of a scheduling system.

2.1 General Framework and Notation

In many application fields different scheduling systems have been established. All those systems have common characteristics, dependencies, constraints, and objectives. Therefore, those different application systems could be described with an equal notation. Based on such a notation the complexity of scheduling systems can be analyzed and if necessary, e.g. for NP-complete problems, heuristics can be conceived.

Graham et al. proposed in 1979 a general notation for scheduling problems [70]. This notation was later refined and extended by Blazewicz et al. [13]. In the following, this notation is briefly introduced. However, here we only present often used elements. For a more comprehensive introduction see Pinedo [123] or T'kind et al. [156].
A scheduling system consists of n jobs and m machines. A job represents a real task from a real world problem. Any possible form of resource where the task or problem is processed is called machine. Within this document the terms *resource* and *machine* will be used interchangeably. A job is normally denoted by j and a certain machine by i. The set of all jobs is τ. The processing time of job j on machine i is described by p_{ij}. If the processing time of a job is independent of the machine, the notation can be reduced to p_j. If the precise processing time of job j is not known before the job has been completed, the users often need to provide an estimation $\bar{p}_j$. The release date of job j is given by r_j. If a system uses weights to specify priorities of jobs, the weight of job j is noted as w_j. Within this document, we always assume that a single job only consists of a single operation where one or several machines are used at the same time. Such jobs are called mono-operational, see T'kind et. al. [156]. A job that would consist of several operations can be modeled by a sequence of jobs with corresponding precedence constraints. The original notation by Graham et. al. used m_j to denote the number of operations in a job shop model (they did not refer to parallel jobs). The notation introduced by Drozdowski instead used $size_j$ to denote the number of machines needed for execution of a parallel job [36]. However, here m_j will be used to specify that job j needs m_j machines in parallel, as introduced by Feitelson et. al. [112]. The completion time of job j is denoted by C_j.
The general notation is decomposed into three fields: $\alpha|\beta|\gamma$. The α-field describes the machine environment of the scheduling system. β includes a set of constraints between jobs and machines within the system. The last element γ describes the objective function. Note that in many applications the term criterion is used instead of objective. However, in the context of scheduling systems the term objective is more common. Within this thesis both terms will be used interchangeably.
Possible values within the α field are, for example, 1 for a single machine, Pm for m parallel identical machines, Qm for m parallel machines with different speeds and Rm for m unrelated parallel machines. The notation can also describe flow shops (Fm, FFc), job shops (Jm, FJc), open jobs (Om), and other scenarios. Furthermore, in this work we assume renewable machines as defined by T'kindt et. al. [156]. In contrast to consumable machines, renewable machines become available again after use.
The β field includes a set of possible constraints. In the original publication by Graham et. al. [70] only 6 possible entries were defined. This was extended in later works, for example by Blazewicz et. al. [13]. Lists of possible constraints that are frequently found in the literature have been collected by Pinedo [123] and T'kind et. al. [156]. Here, just some basic constraints will be presented. The term p_j describes that the processing time of job j on all machines is identical. $\bar{p}_j$ specifies that only an estimate for the processing time of job j is given. With m_j we describe that job j needs m machines in parallel. If in the β-field parallel identical machines are specified (P_m), the term M_j describes that job j can only be computed on a subset M_j of all machines m. Distinct release dates of jobs are specified with r_j. Within this work, by $\bar{r}_j$ we denote that jobs have individual release dates that are not known in advance. In many applications, jobs have a due date d_j to finish. If this is a strict deadline, it is denoted by $\bar{d}_j$. Especially in theoretical publications, scheduling problems are often distinguished into problems with or without preemption $(prmp)$. If preemption is allowed,

jobs can be interrupted and the execution is resumed later on. In the context of parallel computing this is further divided, as a local preemption specifies that a job can locally be preempted but must be started on the same machine later. So, the job is not allowed to be migrated to other machines in opposite to a migratable preemption. Another important aspect of many scheduling problems are precedence constraints, denoted by *prec*. If such constraints exist, certain jobs are only allowed to be started if all predecessors finished their processing. The last aspects that will be mentioned here, in order to specify scheduling constraints, are machine breakdowns (*brkdwn*). Machine breakdowns imply that certain machines are not continuously available. However, often it is assumed that those times are known in advance.

The last part of the general notation scheme γ describes the objective of the scheduling problem. This objective is often defined by the machine provider or administrator. Most real scheduling systems and also the majority of the publications are using only a simple singular objective. Examples are the well-known makespan C_{max}, which is the completion time of the last job to finish from a fixed set of jobs or the total weighted completion time ($\sum w_j \cdot C_j$). A more comprehensive list of such objective functions can be found in the book by Pinedo [123]. For most of the scheduling problems it is important to have a regular objective function. Regular objective functions are nondecreasing in the completion times of the executed jobs.

Scheduling problems are often divided into *online* and *offline* problems. The generation of offline schedules uses all information about all job and machine characteristics as well as all constraints that are all known from the beginning. In opposite, online problems deal with uncertainty. Here, the release dates of jobs are not known in advance ($\bar{r}_j$) and the precise processing time is unknown until the job has been completed ($\bar{p}_j$). Online systems are further divided into online *clairvoyant* and online *non-clairvoyant* scheduling problems, see, for instance, Motwani et. al. [110]. Within clairvoyant online problems only the release dates of jobs are unknown whereas in non-clairvoyant systems the precise processing times are unknown as well.

Following this introduction of the fundamental notation, possible schedules are classified. This classification is used to describe further constraints of the applied scheduling algorithms.

2.2 Classification of Schedules

A schedule $S \in \mathbb{S}$ represents the final allocation of resources to jobs over time. In the following, we assume that all schedules $S \in \mathbb{S}$ are feasible schedules, that is, all constraints are satisfied. Feasible schedules can be classified as follows.

The first class represents the so called *nondelay* schedules. A schedule is called *nondelay* schedule if and only if no machines are kept idle while a job is waiting to be processed that could run on these idle machines.

The second class defines *active* schedules, where a schedule $S_1 \in \mathbb{S}$ is an *active* schedule if and only if $\nexists S_2 \in \mathbb{S}$ such that $C_j(S_2) \leq C_j(S_1),\ \forall j \in \tau$, with at least one strict inequality.

The class of *semi-active* schedules as defined in the following builds the third class. Let $S_1 \in \mathbb{S}$ be a feasible schedule and $\mathbb{S}^*$ be the set of schedules having the same sequence of jobs on the machines as schedule S_1. S_1 belongs to the class of semi-active schedules if and only if $\nexists S_2 \in \mathbb{S}^*$ such that $C_j(S_2) < C_j(S_1)$, for at least one $j \in \tau$.

In most of the practical applications nondelay schedules are generated. However, those schedules are not necessarily optimal to the given scheduling objective.

2.3 Scheduling Complexity

All scheduling problems are special optimization problems. Therefore, the analysis of the individual complexity gives a clear indication for algorithms that should be used to generate the corresponding solutions. Within this context the complexity of the problem is identical with the time complexity of the scheduling algorithm. This complexity is established by calculating the number of operations that the algorithm needs in the worst case to solve the problem. The number of operations may depend on the size of the input, denoted as *Length*, and possibly on the magnitude or representation of the largest element, noted as *Maximum Value*.

The classification of the complexity of algorithms is well established, see, for instance, Garey and Johnson [67]. Problems are assigned to the classes *P* or *NP*. Informally, *P* includes all problems where an algorithm can be constructed in such a way that the number of operations depends polynomially on the *Length* of the input. The class *NP* consists of all problems where a given solution can be verified in polynomial time. Thus, the class *NP* includes the class *P*. However, the class of *NP-complete* represents the "harder" problems within *NP*. Those problems cannot be solved using a polynomial algorithm assuming that *P≠NP*. Moreover, *NP-complete* problems are further divided into problems that are *NP-complete in the ordinary sense* and problems of the class *NP-complete in the strong sense*. *NP-complete in the ordinary sense* specifies that an algorithm exists that can solve the problem in polynomial time depending on the *Length and the Maximum Value* of the problem. Such algorithms are called *pseudo-polynomial*. The class of *NP-complete in the strong sense* problems represents the "hardest" problems within *NP*. Those problems have a so called exponential time complexity.

If practical problems belong to the class *P*, it is reasonable to search for algorithms that solve those problems in polynomial time. Contrary, if the real problem belongs to *NP-complete*, two different types of algorithms can be generated. The first approach generates an approximation algorithm that produces a solution in polynomial time. This solution should be as close as possible to the optimum. This attempt of using heuristics is used most often in practice. If the problem belongs to the class *NP-complete in the ordinary sense* it might be reasonable for smaller problems to use a pseudo-polynomial algorithm to achieve the optimal solution. However, the usability of a pseudo-polynomial time algorithm highly depends on the given application. In the second approach, an algorithm can be proposed that solves the problem optimally but needs exponential time in the worst case. The application of this second approach is input dependent. Thus, such concepts are only applied if the worst case occurs very rarely or the problem size is limited and the computational time still acceptable.

In theory of scheduling problems often either the proof of *NP-completeness*, for instance, in Du and Leung [37], is given or approximation algorithms are developed (e.g. Shmoys et al. [142]). The approximation quality of certain algorithms for offline scheduling problems is often described by approximation factors. Those factors represent provable bounds for the performance of the algorithms regarding the chosen objective. To this end, the quality of the produced schedule by the heuristic is compared with the optimal schedule quality, see Johnson [87].

The evaluation of online scheduling algorithms is a special case of this general problem. In the online scenario, some characteristics or some constraints or the whole set of jobs are not known at the point in time when a new schedule must be determined. Such online scheduling algorithms are often analyzed by using competitive ratios. To this end, the results of the online generated schedule are compared with the optimal, offline generated schedule that was built using all information of the underlying problem. As the optimal solution for *NP-complete* problems is not precisely achievable in polynomial time, often lower bounds for the optimum are calculated, see e.g. Shmoys et al. [143]. The theoretical analysis of scheduling problems is in general based on the worst case behavior. However, such worst case scenarios occur very rarely in practical situations, see, for instance, Krallmann et al. [94]. Therefore, algorithms with the best known approximation factor do not necessarily work well in practice. Moreover, heuristics that are based on the typical cases are often preferred.

For known scheduling problems a complexity hierarchy has been developed, see, for instance, Graham et al. [70]. This structure can be used to determine the complexity of new scheduling problems, if the complexity of a connected problem is already known and included in this hierarchy. Such lists of well known scheduling problems with a single objective have been collected by e.g. Pinedo [123, p. 544f] or T'kindt et al. [156, p. 42]. Note that even some offline problems with a simple objective are *NP-complete in the strong sense*, see Garey and Johnson [67]. Bruno et al. [16] have shown that even some very simple scheduling problems are *NP-complete*.

Many real life scheduling problems are often more difficult than the scheduling problems for which the *NP-completeness* has been proven. Therefore, the development and the analysis of scheduling algorithms for practical problems can often only be done by using heuristics. Here, the mathematical analysis of bounds in most of the cases is too complex. The problems of online scheduling of real parallel computers and Computational Grids are such complex problems. Therefore, the development of appropriate scheduling algorithms is typically based on heuristics.

Before we describe the development of such scheduling algorithms for real problems, we present some methods to solve single objective scheduling problems. These methods will then be extended to handle multiple objectives.

2.4 Single-Objective Scheduling

The previous section has already introduced the complexity of scheduling problems, as they are mostly *NP-complete*. Therefore, we will not discuss the complexity in more detail here. Within this section, we briefly present the major algorithmic approaches that are used for most of the practical problems. Note that the scheduling algorithms that are nowadays applied to schedule parallel computers will be presented in Chapter 4.

The generation of feasible schedules for real life problems can be done in various ways. Here, the research is focused on the development of heuristics that generate high quality schedules in a reasonable amount of computational time. Hence, the choice of a heuristic to generate a schedule highly depends on the given time constraints of the underlying problem. For some problems, the task of creating a schedule is not time critical and all jobs, machines and constraints are well known. In such scenarios, offline scheduling algorithms can be applied. The scheduling task in other scenarios is time critical as the schedule needs to be generated regularly within short time intervals. Here, the development is more focused on algorithms that generate acceptable schedules in most cases within limited time.

The schedule generation processes can be divided into three main types. The *monolithic* type solves the problem directly by using all information together. The *constructive* type starts without a schedule and gradually constructs a schedule by adding one job at a time. The

improvement type of algorithm uses an arbitrarily generated complete schedule and tries to find a better schedule by manipulating the current schedule. All three types are applied for solving practical problems.
One general monolithic approach is the application of linear programming, see, for instance, Nemhauser [114]. If the problem can precisely be described by linear equations without integer values, the approach of linear programming often is the best choice. Here, the result represents the optimal solution. If the system description includes integer restrictions, methods of integer or mixed-integer linear programming can be used. However, assuming that $P \neq NP$, those problems are much harder to solve than linear programs, as those problems have an exponential worst case behavior. Here, branch and bound methods are often applied. Note that the problem size in the number of jobs, machines and constraints has to be limited if mixed integer linear programming is used.
If the problem cannot be described as a system of linear equations or the problem size is too large, other approaches are applied. If the problem belongs to the class of polynomially solvable problems, a constructive algorithm should be developed. If the problem does not belong to this class, other approaches need to be implemented. Pinedo claims: *"Enumerative branch and bound methods are currently the most widely used methods for obtaining optimal solutions to NP-hard scheduling problems."*, see [123, p. 342]. However, as the enumeration of all possible schedules is impractical for many real applications, heuristics have to be applied to obtain acceptable scheduling solutions. Here, branch and bound methods are not adequate because of their lack of scalability. Furthermore, Hart et al. [77] claim: *"Optimal Schedules are seldom required in an industrial environment, rather something that is resilent to the inevitable environment changes that occur"*.
Some of the most commonly used heuristics for solving real complex problems are Filtered beam search [119], Tabu-search [115], Simulated annealing [1], and Evolutionary Algorithms [77, 5]. All these approaches will be introduced briefly in the remainder of this section.
Filtered beam search is a constructive approach. Here, the main goal is to limit the search of the normal branch and bound method. To this end, the elimination process of using lower bounds is extended as at any time only a limited number of most promising nodes is investigated. All other nodes are discarded permanently. The number of retained nodes is called the *beam width* of the search. The selection of the most promising nodes at each level is the most difficult task. A detailed analysis of each node would lead to a good solution but may consume much time whereas a coarse selection is quick but might result in a bad solution. Here, the filter is used as the results are predicted for all available nodes and only for the best nodes a detailed analysis is computed. All other nodes are already permanently discarded. The number of nodes selected for a detailed analysis is called the *filter width*. This procedure can be adapted for various applications as the beam and filter width can be set individually.
Simulated annealing and *Tabu-search* are examples for improvement types of algorithms. Both approaches start with a valid schedule that could, for example, be generated by random. Then the algorithms try to find better schedules in the *neighborhood*. Two schedules are *neighbors* if one can be obtained through a well-defined modification of the other, see Pinedo [123]. Within each iteration of such algorithms, new schedules in the neighborhood are evaluated and either accepted or rejected as candidate solutions. Within the Simulated annealing algorithm the probability to move to a schedule in the neighborhood that has a poorer objective value than the actual schedule is decreasing during the algorithm's run time. This potentially enables the algorithm to overcome local optima. For a more comprehensive analysis of Simulated anneal-

ing see, for example, Aarts et al. [1]. Simulated annealing was successfully applied to many complex scheduling problems, see, for example, Hart et al. [76]. Tabu-search works similar to Simulated annealing, see, for example, Hertz et al. [78]. Here, moves to neighboring schedules with a lower objective are also possible. Contrary to Simulated annealing the neighborhood search of schedules is not probabilistic. Here, at each iteration a list of moves that are *not* allowed is kept. This list of not allowed moves is limited. Every time a move to a neighbor schedule is made the *reverse* move is entered at the top of the tabu list. This avoids the returning to local optima that have been visited before. The determination of the appropriate size of the not allowed moves list is crucial to the overall performance. If the list is too short, cycling may occur, if the list is too long, the search is too restricted. The algorithm ends after a fixed number of movements.

This section ends with the presentation of Evolutionary Algorithms to solve various practical scheduling problems. Montana [109] states: *"The reason for the success of evolutionary algorithms at a wide and ever-growing range of scheduling problems is a combination of power and flexibility."*. The power derives from the ability to find or approximate global optima with a high quality. Evolutionary Algorithms are flexible as they can be applied to various problems with only very limited modifications. However, if problem specific knowledge can be integrated, the range of problems to which Evolutionary Algorithms can be effectively applied increases. Evolutionary Algorithms for single and multi-objective optimization problems will be introduced in more detail in Chapter 3. Hence, this section only provides a rough idea and some practical scheduling problems.

Scheduling problems are often solved by using Genetic Algorithms as a subset of Evolutionary Algorithms. Here, a population of individuals that represent schedules is processed within every iteration. Normally, each individual reflects the permutation of jobs onto the machines, see Hart et al. [77]. The description of the permutation does not necessarily include real job numbers but representatives. The fitness of each individual can be calculated by applying a pre-defined schedule objective function. The procedure works iteratively. The current individuals are used to generate a next generation of individuals, called children, by applying some operators to change and combine the current individuals. During this process the fitness value of the individuals is used to decide which individual to change or to use in combination with other individuals. This process leads to a next generation that in most of the cases consists of individuals with a higher fitness. The whole process ends after a fixed number of iterations or by reaching a pre-defined fitness level.

The development of appropriate schedules for certain practical schedules by using Evolutionary Algorithm is still a time consuming task. Therefore, such approaches can typically only be applied to offline problems or in cases where the generation of a new schedule is not time critical. Furthermore, most of these practical problems only use a single objective.

An example of offline problems is the generation of a bus transit system as described by Deb et al. [30]. In this example, the only criterion is the minimization of the overall waiting time of all passengers. The generation process of possible schedules for a flow shop scheduling problem is presented by Reeves et al. [128]. For this problem, the authors take the features of the generated landscape into account and present further optimization possibilities. The problem representation of flow shop problems is further analyzed by Cotta et al. [25]. They show that representations in which absolute positions of jobs are coded perform better than all other representations. The generation of a daily schedule for the catching and transportation of a large number of live chickens to a factory with many constraints by using Genetic Algorithms is presented by Hart et al. [76]. Here, the main focus is only generating a system that is able

to automatically produce schedules similar to the schedules generated by human beings. The main difficulties lie in the huge number of constraints. The application of penalty functions has proven to be beneficial since non-feasible solutions are sometimes kept within the population but with a low probability. This enables the algorithm to avoid local optima.
Beside the above mentioned concepts for solving practical scheduling problems, methods like co-evolution or Ant colonies are, for example, described by Hart et al. [77].

2.5 Multi-Objective Scheduling

In the previous section, several single-objective problems were introduced. Some of those problems are already hard to solve. However, most of the real-world search and optimization problems even involve multiple often conflicting objectives. Since no single solution can be called an optimal solution to multiple conflicting objectives, the resulting multi-objective optimization problem generates a number of trade-off optimal solutions. With the set of trade-off solutions in mind, the task of the decision maker is to define which of those solutions should be used.
This general problem structure can be found in many real life problems. Regarding the example of parallel computers, the machine provider does not necessarily treat all users or user groups in the same way. Jobs of preferred user groups might be executed earlier by further delaying jobs of other user groups. In parallel, the machine providers often want to have a high machine utilization. In most cases, this results in a conflict as it will be impossible to satisfy all users and have a high utilization in parallel. Another example is the scheduling of airplanes to gates at airports. Here, one goal is the minimization of the number of flights without gate assignment. In parallel, the total passenger walking distances should be minimized and gate assignment preferences of airlines should be maximized. All those single objectives are conflicting in most cases.
This leads to the definition of a multi-objective scheduling problem. T'kind et al. [156, p. 107] define a multi-objective scheduling problem as: *"the problem which consists of computing a Pareto optimal schedule for several conflicting criteria. This problem can be broken down into three sub-problems:*

- modelling of the problem*, whose resolution leads to the determination of the nature of the scheduling problem under consideration as well as the definition of the criteria to be taken into account.*
- taking into account of criteria*, whose resolution leads to the indication of the resolution context and the way in which we want to take into account the criteria. The analyst finalizes a decision aid module for the multicriteria problem, also called a module for taking accounts of criteria.*
- scheduling*, whose resolution leads us to find a solution of the problem. The analyst finalizes an algorithm for solving the scheduling problem, also called a resolution module for the scheduling problem."*

The modelling of the underlying problem is of course critical and has a high influence on the resulting system. Nevertheless, the modelling is very application dependent and therefore not analyzed in more detail. The consideration of several objectives, however, is important and several approaches will be presented within this work. This incorporation highly influences the resulting schedule decision. Before the precise introduction of multi-objective optimization is presented, we will extend the scheduling notation of Section 2.1.

2.5.1 Extended Notation for Multi-Objective Scheduling Problems

The previously introduced scheduling notation misses the ability to express multiple objectives and dependencies between those objectives. T'kindt et al. [156, p. 109] extended the general notation of Graham et al. [70] by specifying the γ field in more detail. Here, some examples for possible γ entries are presented:

- Z if the objective is to minimize the unique objective Z
- $F_l(Z_1, \dots Z_k)$ if the objective is to minimize a linear convex combination of k objectives
- $\epsilon(Z_u|Z_1, \dots Z_{u-1}, Z_{u+1} \dots Z_k)$ indicates that only the objective Z_u is minimized, subject to all the other objectives being upper bounded by known values
- $F_{Tp}(Z_1, \dots Z_k)$ indicates an objective function which is an expression of a distance to a known ideal solution
- $F_S(Z_1, \dots Z_k)$ indicates a very particular function which takes into account a known ideal solution to find the sought solution
- $GP(Z_1, \dots Z_k)$ if there are goals to reach for each objective in the scheduling problem (goal programming)
- $Lex(Z_1, \dots Z_k)$ indicates that the decision maker does not authorize trade-offs between objectives. The order in which the objectives are given is related to their importance.
- $\#(Z_1, \dots Z_k)$ indicates the enumeration problem of all the Pareto optima. Therefore we associate uniquely to this problem an a posteriori resolution algorithm which does not use any of the aggregation methods.

Later, we will use this extension to classify several scheduling systems. Before this, we introduce the concept of multi-objective optimization.

2.5.2 Multi-Objective Optimization

Within this section, we will first introduce the basic concepts and terminology of multi-objective optimization problems in general. Later, this general structure will be mapped to the scheduling problems. A comprehensive introduction to the field of multi-objective optimization can be found by Miettinen [108], Deb [27], Coello Coello et al. [23], and Osyczka [117]. Without loss of generality, a minimization problem is assumed in the remainder of this thesis. A maximization or mixed maximization/minimization problem can be transformed into a minimization problem by multiplying the maximization problem objectives with -1.
A general *Multi-Objective Optimization Problem* consists of a set of l decision variables and a set of k objective functions. A decision vector $\vec{x} \in X$ of the problem specifies a value for each of the l decision variables. X is denoted as the decision space. The k objective functions are conflicting. Otherwise, the problem degenerates to a single-objective optimization

problem. The outcome of the k objective functions defines the objective space Y of dimension k (sometimes also called the solution space). Thus, the k-th objective function $f_k(\vec{x})$ is described by $f_k(\vec{x}) : \mathbb{R}^l \rightarrow \mathbb{R}$ and the vector function $\vec{f}(\vec{x})$ by $\vec{f}(\vec{x}) : \mathbb{R}^l \rightarrow \mathbb{R}^k$. Furthermore, problem related constraint functions define the *feasible* decision space X_f. In the remainder of this work we will not further specify those constraint functions. For general multi-objective optimization problems they are often described by several inequality functions. For scheduling problems the feasible set simply consists of all schedules that satisfy all given scheduling constraints. The multi-objective problem can now be formulated as:

$$\begin{array}{ll} \text{minimize} & \vec{y} = \vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})) \\ \text{subject to} & \vec{x} = (x_1, x_2, \ldots, x_l) \in X_f \end{array} \tag{2.1}$$

The feasible region within the objective space Y_f can be described by $Y_f = \vec{f}(X_f) = \bigcup_{\vec{x} \in X_f} \vec{f}(\vec{x})$.
As the problem is a multi-objective problem, usually no element of the feasible set $\vec{x}' \in X_f$ exists that minimizes all given simple objective functions:

$$\vec{x}' = \arg \min_{\vec{x} \in X_f} \{f_1(\vec{x})\} = \ldots = \arg \min_{\vec{x} \in X_f} \{f_k(\vec{x})\}. \tag{2.2}$$

During the multi-objective optimization a partial ordering of the given solutions is used (Pareto 1896). Here, a solution $\vec{f}(\vec{x}_1)$ is superior to another solution $\vec{f}(\vec{x}_2)$ if $\vec{f}(\vec{x}_1)$ is better in all objectives than $\vec{f}(\vec{x}_2)$. This observation leads to the concept of Pareto dominance and Pareto optimality.

Definition 1 ***Pareto Dominance:*** *A decision vector $\vec{x}_1$ is said to dominate another decision vector $\vec{x}_2$, iff:*

$$\forall i \in \{1, \ldots, k\} : f_i(\vec{x}_1) \leq f_i(\vec{x}_2) \ \wedge \ \exists i \in \{1, \ldots, k\} : f_i(\vec{x}_1) < f_i(\vec{x}_2).$$

This establishes the so-called Pareto preference relation $\prec_P$. Here, $\vec{x}_1 \prec_P \vec{x}_2$ describes that the decision vector $\vec{x}_1$ dominates the decision vector $\vec{x}_2$. If neither $\vec{x}_1 \prec_P \vec{x}_2$ nor $\vec{x}_2 \prec_P \vec{x}_1$ the decision vectors are called indifferent or $\vec{x}_1 \sim \vec{x}_2$. In such cases, a more detailed specification of preferences would be required to decide which decision vector is better.
Using the given definition of Pareto dominance the Pareto optimality can be defined in the following way.

Definition 2 ***Pareto Optimality:*** *A decision vector $\vec{x} \in X_f$ is said to be non-dominated regarding a set $A \subseteq X_f$ iff $\nexists \vec{a} \in A : \vec{a} \prec_P \vec{x}$. Furthermore, $\vec{x}$ is said to be Pareto optimal iff $\vec{x}$ is non-dominated regarding X_f.*

In *Pareto optimization*, we are searching for the entire set of Pareto optimal decision vectors which is called the *Pareto optimal set* and the corresponding objective vectors the *Pareto front.*

Definition 3 ***Non-dominated Set and Front:*** *Let $A \subseteq X_f$. The function $nd(A)$ gives the non-dominated subset of A:*

$$nd(A) = \{\vec{x} \in A | \nexists \vec{x}' \in A \text{ with } \vec{x}' \prec_P \vec{x}\}$$

The corresponding set of objective vectors $\vec{f}(nd(A))$ is the non-dominated front regarding A. Furthermore, the set $X_P = nd(X_f)$ is called the Pareto optimal set and the set $\vec{f}(X_P)$ is denoted as the Pareto front.

For most real life problems it is impossible to generate the whole Pareto front. Therefore, nearly all problems only use an approximation of $f(X_P)$. At the technical level it is not very easy to efficiently calculate the Pareto front of a given set of objective vectors. The algorithm with the best known processing time was invented by Kung et al. (1975) and is described in detail by Deb [27, p. 38].
After this more formal introduction of multi-objective optimization, we will briefly present some theoretical results for multi-objective scheduling systems, followed by an introduction of various methods to solve multi-objective optimization problems.

2.5.3 Theoretical Analysis of Multi-Objective Scheduling Systems

The theoretical work for single-objective scheduling systems is limited, as already shown in Section 2.3. Hence, the theoretical analysis of multi-objective scheduling problems is even more difficult and occurs rarely. Pinedo presents a representative simple example of the problem structure [123, p. 79]. Here, the single-objective problem $1||\theta_1 \sum w_j C_j + \theta_2 L_{max}$ has been proven to be NP-hard in the strong sense for arbitrary θ_1 and θ_2. However, it is possible to generate the Pareto front for the problem $1||L_{max}, \sum w_j C_j$. Furthermore, the single-objective problems $1|| \sum w_j C_j$ and $1||L_{max}$ can be solved optimally by using the weighted shortest processing time first algorithm and the earliest due date first algorithm respectively.
Nevertheless, several theoretical results have been collected by T'kindt et al. [156] for various machine configurations and objective function combinations. The later presented scheduling problems for high performance parallel computers all belong to the class of NP-hard problems. In the next section, we will therefore discuss possible solution techniques.

2.5.4 Multi-Objective Optimization Techniques

The solution of multi-objective problems is often difficult. Frequently, this results from the lack of well defined objectives. Instead only "orientations" can be provided. Furthermore, the decision maker is rarely a unique individual but a group of people. Even more problematically, the set of possible solutions is rarely fixed, but tends to evolve over time. Finally, the decision maker might not be able to express the optimal solution. This makes various solution techniques to solve multi-objective optimization problems valuable.
Multi-objective optimization techniques have been classified into four categories by Miettinen [108, p. 63]:

1. methods where no articulation of preference information is used (*no-preference methods*),

2. methods where a priori articulation of preference information is used (*a priori methods*),

3. methods where progressive articulation of preference information is used (*interactive methods*), and
4. methods where a posteriori articulation of preference information is used (*a posteriori methods*).

This classification is not complete and in some cases methods may belong to two of the mentioned categories. *No-preference* methods do not assume any information about the importance of the various objectives. A heuristic is used to generate the next single optimal solution. This approach does not try to find several solutions in parallel. Furthermore, the progress of the heuristic cannot really be influenced. The *a priori* attempts assume more information about the optimization problem. Here, it is possible to identify the importance of the various objectives or to specify some ideal solutions. Hence, the problems often degenerate into single-objective problems as only one resulting function has to be optimized. The *interactive* methods use preference information during the optimization process. Such methods can only be applied, if the decision maker has a clear explicit or implicit objective function in mind. Furthermore, the number of interactions has to be limited. The *a posteriori* approach does not assume preference information in advance but generates the Pareto front. After this, the decision maker is able to select certain solutions from this front. The main advantage of this approach is the availability of the Pareto front (or at least an approximation). This enables the decision maker to use higher level information to select the final solution. This higher level information can often not be expressed explicitly. Furthermore, the individual decision might highly depend on the given Pareto front and in this sense from other possible trade-off solutions.
In the following, we will describe some classical solution methods. These methods are called classical in order to differentiate these methods from the later introduced methods that use Evolutionary Algorithms. The classical methods try to aggregate the objectives into a single, parameterized objective function. Note that the parameters are not specified by the decision maker but systematically varied during the optimization process. Within the different optimization runs various parameterizations are used to generate the approximation of the Pareto front. Here, we present only some example methods. By this, we try to assign each method to the a priori approach, the interactive approach or the a posteriori approach. However, this assignment is not unique and some methods can belong to several categories depending on the intension and input of the decision maker. Details on the *no-preference methods* are given by Miettinen [108, p. 67ff]. A complete and detailed introduction of all classical methods is provided by Miettinen [108], Deb [27], and Coello Coello et al. [23].

2.5.4.1 A Priori Methods

This group of techniques consists of all methods that assume either a pre-ordering of the objectives or a pre-defined desired goal for each criterion. The most common methods are the Lexicographic Ordering and Goal Programming.

2.5.4.1.1 Lexicographic Ordering ($Lex(Z_1, \dots, Z_k)$) Within this method the decision maker has to order all objectives corresponding to their absolute importance. This results in an ordered set of all objective functions $(f_{o_1}, f_{o_2}, \dots, f_{o_k})$.
Then the problem is minimized for the objective function f_{o_1}. If a unique solution exists the optimization ends. Otherwise, the objective function f_{o_2} is minimized for all solutions that

minimize f_{o_1}. If this results in a unique solution, the optimization process ends, otherwise the remaining functions will be used until a single solution is found or the procedure ends with several solutions as also the last of the used objective functions does not generate a unique solution.

2.5.4.1.2 Goal Programming ($GP(Z_1, \ldots, Z_k)$) For this optimization technique many derivations exist. Therefore, we will only present the main idea behind this approach. The decision maker has to define a goal or target t_i for each objective $i \in 1, \ldots, k$. This goal might not be reachable if the decision maker is too optimistic. The method then tries to minimize the weighted absolute deviation from the defined targets:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{k} w_i |f_i(\vec{x}) - t_i| \\ \text{subject to} \quad & \vec{x} \in X_f. \end{aligned} \tag{2.3}$$

The weights are used to express the different influences of the various objectives to the overall solution. The final decision highly depends on the pre-defined targets for the objective and on the given weights.

2.5.4.2 Interactive Methods

There is a number of different interactive methods where only very little knowledge about the problem is needed in advance. The main aspect of all these approaches is that at certain points in time the decision maker is involved in the optimization process by defining weight vectors, goals or directions for the next optimization interval. This makes such methods popular in practice. However, they cannot be processed automatically. The most popular interactive procedures are the interactive surrogate worth trade-off method, the step method, the reference point method, the guess method, and the light beam search. More details about these approaches can be found in the book of Miettinen [108, p. 131ff.].

2.5.4.3 A Posteriori Methods

These methods directly generate the Pareto front without prior knowledge about the ranking of the various objectives. After the generation of the Pareto front the decision maker is able to select certain solutions among all alternatives. The problem with such approaches is the typically computational expensive generation of the Pareto front. During this generation, a priori methods are used to generate individual solutions. Furthermore, technical questions like the presentation of the Pareto front in cases with more than three objectives arise.

2.5.4.3.1 Weighted Sum Method ($F_l(Z_1, \ldots, Z_k)$) Using this method all objectives are associated with a weight. Then the weighted sum of all objectives has to be minimized. Often, the weights are normalized.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{k} w_i f_i(\vec{x}) \\ \text{subject to} \quad & \vec{x} \in X_f. \\ \text{where} \quad & \forall i \in \{1,\ldots,k\} : w_i \geq 0 \ \wedge \ \sum_{i=1}^{k} w_i = 1 \end{aligned} \tag{2.4}$$

The Pareto front can then be generated by varying the weights w_i and calculating the corresponding solution. Depending on the function evaluations this might be time consuming. Furthermore, a uniformly distributed set of weights w_i does not need to find a non-uniformly distributed set of Pareto optimal solutions. Therefore, it might become crucial to select appropriate weights in order to generate a well-spread Pareto front.

2.5.4.3.2 epsilon-Constraint Method ($\epsilon(Z_u|Z_1,\ldots,Z_{u-1},Z_{u+1},\ldots,Z_k)$) Within this approach only the most important objective $f_u(\vec{x})$ is minimized. For all other objectives the decision maker must provide an upper bound. The optimization approach can be described by:

$$\begin{aligned} \text{minimize} \quad & f_u(\vec{x}) \\ \text{subject to} \quad & \forall i \in \{1,\ldots,k\} \wedge u \neq i : f_i(\vec{x}) \leq \epsilon_i \\ \text{and} \quad & \vec{x} \in X_f. \end{aligned} \tag{2.5}$$

The Pareto front can be generated by changing the upper bounds for the various objectives. This variation of upper bounds cannot be done arbitrarily as with too small bounds no solution can be found. Otherwise, an upper bound that is too high does not change the resulting solution.

2.5.4.4 Problems of Classical Multi-Objective Optimization Methods

Most of the classical multi-objective optimization approaches convert the original problem into a single-objective problem that can be solved with well-known techniques. The transformation of the problems often requires some problem specific information from the decision maker. This reflects the main disadvantage as the decision maker for complex multi-objective problems does not know the potential trade-offs in advance. Under these circumstances, the definition of weights or certain bounds is often not possible. Furthermore, the generation of the Pareto front with classical optimization methods is difficult if a unique distribution within the decision space does not reflect a well-defined distribution in the objective space. Additionally, the convergence to an optimal solution often depends on the chosen initial solution. Moreover, all algorithms only generate one possible Pareto optimal solution at a time. Hence, the generation of the whole Pareto front can be computationally very time consuming as parallel machines cannot be used efficiently. The multi-objective Evolutionary Algorithms that will be presented in Chapter 3 are able to overcome most of the described restrictions.

2.5.5 First Multi-Objective Scheduling Approaches

This section provides the details of all existing scheduling systems known to the author that use more than two objectives. On the one hand, this collection of problems and the corresponding solutions motivate our methodology to automatically develop multi-objective scheduling algorithms. On the other hand, the presentation emphasizes that only very few scheduling problems are tackled with more than two objectives.

On the theoretical level only simplified problems have been solved and analyzed as already discussed in Section 2.5.3. For these more idealistic problems some deterministic algorithms or heuristics with provable bounds have been proposed.

Most of the realistic problems are solved with the help of heuristics and Evolutionary Algorithms similar to the single-objective case discussed in Section 2.4. Here, we present some examples for multi-objective problems. A collection of real life multi-objective problems that have been solved by using Evolutionary Algorithms can be found by Coello Coello et al. [23] and Ehrgott et al. [38].

The works of Stein and Wein [152] and later Mu'alem and Feitelson [112] analyze the problem $P_m|m_j, r_j, prmp|\sum w_j C_j, C_{max}$ for parallel computers. However, these works only use two objectives to compare several possible algorithms. To this end, upper bounds for competitive factors are derived for the different scenarios and algorithms. Both objectives are not simultaneously optimized.

The problem of generating the assignment of airplanes to gates at airports is presented by Dorndorf et al. [32]. Here, typical objectives and constraints are given. The work includes an overview of the most popular solution techniques in this field. These are branch and bound, Tabu-search, Genetic Algorithms and rule based expert systems. A Simulated annealing approach is used in the work by Drexl et al. [35] to solve the gate assignment problem using three objectives. These are the minimization of the ungated flights, the minimization of the total passenger walking distance and the maximization of the total gate assignment preferences of the different airlines. The Simulated annealing technique has proven to work well by solving this problem.

As an example from the area of scheduling the generation of energy by a collection of power plants is given by Srinivasan et al. [151]. Here, two objectives are used: the produced energy by all power plants and the corresponding ecological emission. The system first generates the Pareto front by using derivation of Evolutionary Algorithms that are similar to the multi-objective Evolutionary Algorithms presented in Section 3.2.

Guntsch et al. [71] use an Ant colony approach to solve the two-dimensional scheduling problem of minimizing the total tardiness and minimizing the total changeover cost for a single machine. To this end, two Ant colonies are used in parallel, each optimizing one of the objectives. The combination of both colonies during the optimization process forms the resulting Pareto front. Therefore, this approach is of type $\#(Z_1, Z_2)$.

Balicki et al. [11] use three objectives during the process of finding appropriate allocations of computer programs modules within a distributed computer system. The algorithm tries to maximize the performance of the bottleneck computer, to minimize the costs of the computation and to maximize the total numerical performance of the selected workstations. The algorithm is based on a multi-objective Evolutionary Algorithm (NSGA-II), which will be presented in Section 3.2.2. Furthermore, the operator that modifies current schedules uses a Tabu-search technique.

The problem of job shop scheduling with the objectives of minimizing the total flow time and the total tardiness has been treated by Tamaki et al. [155]. They use a Genetic Algorithm with a parallel selection strategy. A permutation ordering scheme is used for all jobs. However, the online scenario is not time critical as a new genetic algorithm optimization can be initiated frequently. The work by Bagchi [9] uses the same objectives and furthermore minimizes the makespan. Here the NSGA-II algorithm is again used with a permutation ordering of all jobs. The last work mentioned here is done by Cochran et al. [22]. In this work, the scheduling of parallel machines uses three objectives, the total weighted completion time, the total weighted tardiness, and the makespan. However, in contrast to the scheduling problem of parallel computers, only sequential jobs need to be scheduled. A Genetic Algorithm is applied to vary the different possible schedules. Like in the mentioned work by Tamaki et al. [155], the Genetic Algorithm has enough computational time to generate new solutions in a periodic time interval. A special selection scheme ensures the development of the Pareto front. Again, the underlying scheduling problem uses a permutation ordering of all jobs.

The list of first attempts to apply multi-objective optimization techniques could be extended, see, for instance, Coello Coello [23]. None of the solutions that deal with real life problems is based on classical multi-objective optimization approaches. In addition, most of them aim at generating the Pareto front in order to solve the underlying problem. In those cases, often Evolutionary Algorithms are applied that have proven to work well in such scenarios. Therefore, some of those algorithms will be presented in Section 3.2. All of the shown problems do not use more than three objectives. As mentioned above, real life problems, like the scheduling of massively parallel processing systems, might need more than three objectives to model the relationship to different user groups. Thus, the presented techniques do not fit to our problem. Within this work we will exemplarily incorporate seven objectives, see Chapter 6. This will highly increase the flexibility of the decision maker to specify the required system behavior.

2.6 General Design Process of Scheduling Systems

Within this section some general design recommendations for the development of appropriate scheduling systems will be given. During the generation of our multi-objective scheduling approach for parallel computers we will in some sense follow this general guideline. Krallmann et al. [94] have introduced and discussed this subject in more detail.

During the development of scheduling systems the task of exploring the machine configurations and all necessary constraints (physical and logical scheduling constraints) might be time consuming. However, the difficult task is to describe the objectives of the machine providers in a way that can automatically be evaluated. Furthermore, the machine providers respectively owners may not be able to express their preferences with mathematical equations.

The development of scheduling strategies is based on the given objective functions. Therefore, the scheduling objective should be expressed as precisely as possible. The goal of the resulting scheduling strategies is the best achievable allocation of machines to jobs regarding this objective. Thus, the system might be able to reduce the number of required machines to process a certain amount of jobs. Furthermore, more jobs could be solved on the available machines within the same time interval. Therefore, the quality of a good scheduling system can often be compared with providing more machines to process the available jobs. Note that this point of view is simplified as the machine provider might have other preferences than processing as many jobs as possible on the local machines.

Krallmann et al. [94] divide the scheduling algorithm development into three parts: the scheduling policy, the objective function, and the scheduling algorithm.

2.6.1 Scheduling Policy

At the highest level of the scheduling system development the machine provider or owner needs to specify a set of rules. Those rules are used to determine the machine allocation of all jobs. Hence, the scheduling policy is able to define which job is started on which machine subset in the case of conflicts between several jobs. Such conflict situations occur frequently as we assume the allocation of rare machines.
The set of rules may be described in a non-formal way. However, using those rules it is possible to distinguish between *good* and *bad* schedule outcomes. Some rules need to be explicit like *"Finish every job as soon as possible if no other rule is affected"*. Furthermore, those scheduling policies enable the machine provider to specify certain rules for specific users or user groups in order to specify certain preferences.
Krallmann et al.[94] define two properties of a good scheduling policy:

1. The scheduling policy is able to settle conflicts between rules.
2. The scheduling policy can be implemented within an algorithm.

2.6.2 Scheduling Objective

At the next level of the development the scheduling policy is refined to a scheduling objective. Here, the goal is to determine an objective function that for every possible schedule calculates a single scalar value. Then, different schedules can automatically be compared by using these scalar values. To derive this scheduling objective the following approach can be used, see Krallmann et al. [94]:

1. For a typical set of jobs determine the Pareto front of possible schedules (using the scheduling policy).
2. Define a partial order of these schedules.
3. Derive an objective function that generates this order.
4. Repeat this process for other sets of jobs and refine the objective function accordingly.

The problem with this approach is the generation of the Pareto front of possible schedules by using the scheduling policy. This policy can be described in a non-formal way. However, during the generation of the Pareto front it is necessary to automatically decide whether a solution dominates other solutions and vice versa.

2.6.3 Scheduling Algorithms

The scheduling algorithm is responsible for generating a valid schedule for all incoming jobs. A *good* scheduling algorithm is able to generate optimal or nearly optimal schedules for various job streams with respect to the previously defined objective function. Furthermore, the time and resource consumption to calculate the schedule should be limited. The development of scheduling algorithms depends on the actual application. For some problems, a standard algorithm, taken from the literature, might be appropriate. In other cases, it might be necessary to develop or at least adapt new algorithms.

Chapter 3

Evolutionary Optimization

This chapter introduces the concepts of Evolutionary Algorithms in more detail. Note that all presented details are used in the remainder of this thesis. Furthermore, it is not sufficient to just reference existing publications as our work differs in many details from these works. Thus, we detail our approaches in order to provide all information that is necessary to reproduce the later presented results.

As already mentioned in Chapter 2 most real life scheduling problems are solved by using such techniques. Within this work, we will also apply several Evolutionary Algorithms to solve single-objective as well as multi-objective optimization problems. To this end, a brief introduction to the general concept of Evolutionary Algorithms will be given followed by specialized techniques for single-objective problems. Those methods are then extended to solve multi-objective optimization problems.

Evolutionary Algorithms mimic basic principles of biological evolution in order to solve complex problems. Typically, they are applied as randomized search heuristics for solving black-box optimization problems. The common term *Evolutionary Algorithms (EA)* comprises several techniques such as *Genetic Algorithms (GA)* (see Holland [80]), *Evolution Strategies (ES)* (see Schwefel [134] and Rechenberg [125]), and *Evolutionary Programming (EP)* (see Fogel [60]). Other, also nature driven methods, like Fuzzy Logic (e.g. Cordón et al. [24]), Neural Nets (see Peterson et al. [122]), Ant colonies (e.g. Guntsch et al. [71]), Simulated annealing (e.g. Aarts [1]), and Particle Swarm Optimization (e.g. Kennedy et al. [90]) are not included in the group of Evolutionary Algorithms as in these cases the problem adaptation does not operate on an additional abstraction level that encodes the underlying problem (genotype).

Evolutionary Algorithms have clearly demonstrated their capabilities to yield good approximate solutions in many real life scenarios. They are often applied for optimization problems with discontinuous, non-differentiable, and even noisy or moving solution spaces, see Bäck et al. [7, 8]. Classical optimization techniques, like direct methods or gradient based methods, tend to fail in such scenarios.

All of these techniques use at generation t a *population* $P_{t,\mu}$ of μ *individuals*. An individual is an encoded solution (i.e. a decision vector) to some problem. Each individual can be represented in various ways, e.g. by a string corresponding to a biological *genotype*. This genotype defines the characteristics of the individuals. The *phenotype* is the decoded *genotype* and reflects the observable behavior of the specific individual. An Evolutionary Algorithm requires both, an *objective* and a *fitness* function. The objective function reflects the problem domain and defines the quality of a given solution of the real problem. To this end, the phenotypic

representation of the specific individuals is used. The fitness function measures how well a particular solution solves the optimization problem. In most cases, objective and fitness functions are identical. However, in the context of multiple objectives they differ fundamentally.
Just as in nature, Evolutionary Algorithms modify the individuals of a given population $P_{t,\mu}$ to generate solutions with a higher fitness. This process is repeated until a defined stop criterion is reached. A typical stop criterion is, for example, a pre-defined number of generations of the population. The three major evolutionary operators to modify the individuals of the population are *mutation, recombination*, and *selection*. The mutation slightly modifies the genotype of the individuals. The recombination generates new individuals by combining the genotypic representation of several existing individuals. The selection reduces the number of individuals that form the next generation by choosing the individuals with higher fitness values. Using selection, the population gradually moves to regions of the search space in the proximity of optimal solutions.
The selection of an appropriate Evolutionary Algorithm for a specific problem is important. Often, two conflicting goals can be observed, the *exploration* and the *exploitation*. The exploration tries to find solutions in regions of the search space that have not been analyzed before. In opposite, the exploitation aims at identifying solutions nearby existing solutions in order to improve the resulting solution quality. Different Evolutionary Algorithms provide different abilities to explore and exploit the solution space of a given problem. On the one hand, this is caused by different mutation and recombination operators that are mainly responsible for the exploration task. On the other hand, the various selection operators are mainly focused on the exploitation.
To put things into more concrete terms, Algorithm 3.1 outlines the general structure of Evolutionary Algorithms, as, for example, presented by Bäck et al. [7, 8].

Algorithm 3.1 Outline of an Evolutionary Algorithm.

```
$P_{0,\mu} \leftarrow$ initialization, $P_{0,\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
$P_{0,\mu} \leftarrow$ evaluation;
$t \leftarrow 0$;
while (termination criterion not fulfilled) do
    $P'_{t,\lambda} \leftarrow$ recombination($P_{t,\mu}$), $P'_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
    $P''_{t,\lambda} \leftarrow$ mutation($P'_{t,\lambda}$), $P''_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
    $P''_{t,\lambda} \leftarrow$ evaluation;
    $P_{(t+1),\mu} \leftarrow \begin{cases} \text{select}(P''_{t,\lambda}) & \text{for } (\mu,\lambda) \text{ selection} \\ \text{select}(P_{t,\mu} \cup P''_{t,\lambda}) & \text{for } (\mu+\lambda) \text{ selection} \end{cases} \quad \{P_{(t+1),\mu} \in \mathcal{M}_\mu(\mathbb{I})\}$
    $t \leftarrow t + 1$
end while
```

The algorithm starts with the initialization and evaluation of a parent population $P_{0,\mu}$. This population consists of μ individuals of the individual space $\mathbb{I}$. In the following, we denote a multi set of μ individuals of the individual space $\mathbb{I}$ by $\mathcal{M}_\mu(\mathbb{I})$. Thus, the population $P_{0,\mu}$ is element of $\mathcal{M}_\mu(\mathbb{I})$. After the initialization and evaluation of the first parent population, the evolutionary loop is repeated until a given termination criterion is fulfilled.
First, λ offspring individuals that form $P''_{t,\lambda}$ are generated by means of the random variation operators from the current parent population. This is done by employing recombination and/or mutation operators, which can be specific to the problem at hand. Then, the objective function values of all new individuals are calculated. Based upon the evaluation results, a new

population of individuals is selected ($P_{(t+1),\mu}$). In the case of a comma strategy (μ, λ), the new parents are chosen from the offspring ($P''_{t,\lambda}$) only. This is normally the case for Genetic Algorithms. The plus strategy $(\mu + \lambda)$ takes also the parents into account ($P_{t,\mu} \cup P''_{t,\lambda}$). This kind of selection is often limited to Evolution Strategies. Each loop incorporating the procedures above represents a generation.
Evolutionary Algorithms are applied to single- and multi-objective optimization problems. However, the selection strategy differs completely as algorithms for multi-objective optimizations must ensure a high diversity between the generated solutions. Here, we present Evolution Strategies as they are applied in the remainder of this work. First, we introduce algorithms for the single-objective case followed by a presentation of extensions for the multi-objective scenario.

3.1 Single-Objective Evolutionary Algorithms

The set of existing single-objective Evolutionary Algorithms is large. It consists, for example, of the already mentioned Genetic Algorithm, the Evolutionary Programming, and Evolution Strategies. In this section, we only introduce Evolution Strategies in more detail as only those strategies are applied in the remainder of this work. Although originally Evolution Strategies were designed for discrete parameter settings, they are nowadays mainly used in the area of numerical parameter optimization.
A new notation for Evolution Strategies is $(\mu, \kappa, \lambda, \rho)$ [137]. This describes that μ parents produce λ offsprings within each generation. To this end, the recombination of ρ parents and mutation are applied. Further, each individual can participate in at most κ successive generations. This general notation includes the older variants (μ, λ) and $(\mu + \lambda)$ as $\kappa = 1$ is equivalent to the comma strategy and $\kappa = \infty$ specifies the plus strategy. All four parameters, μ, λ, κ, and ρ are called *exogenous* strategy parameters and are kept constant during the evolution.
The population $P_{t,\mu} \in \mathcal{M}_\mu(\mathbb{I})$ consists of μ individuals. An individual $a_k \in \mathbb{I}$ is described by an *object parameter set (or vector)* $\vec{o}_k = (o_1, o_2, \dots, o_u)$, a set of *strategy parameters* $\vec{s}_k = (s_1, s_2, \dots, s_v)$ and its objective function value $F_k = F(\vec{o}_k)$. The length of the objective vector which is equal to the number of elements of this vector is u and the length of the strategy parameter is v. However, in most cases, only one strategy parameter is used or every object parameter consists of a corresponding strategy parameter $(u = v)$, see Section 3.1.3.1. For the remainder of this chapter, we will therefore assume that $u = v$.

$$a_k = (\vec{o}_k, \vec{s}_k, F(\vec{o}_k)) \tag{3.1}$$

The strategy parameters $\vec{s}_k$ are called *endogenous* parameters. They influence the genetic operators, like the mutation. Further, endogenous strategy parameters can evolve during the evolution process and are needed in *self-adaptive* Evolution Strategies, see Section 3.1.5.
The basic algorithm of an Evolutionary Algorithm as described in Algorithm 3.1 is extended to describe the working concept of Evolution Strategies in Algorithm 3.2. For further details, see Beyer and Schwefel [12].
During the initialization of the Evolution Strategy a first population is randomly generated and evaluated. Then, λ offsprings are created. To this end, a parent pool is built by randomly choosing ρ parents. If $\rho = 1$ no recombination of parents is used. Then the strategy parameters of the parents are recombined followed by a recombination of the object vectors. The

calculated strategy parameters then undergo a mutation. This is followed by the mutation of the object parameters using the already mutated strategy parameters. Hence, the sequence during the mutation must not be changed. Note that the resulting offspring parameters are marked by using a tilde. Finally, the fitness value of the generated offsprings is calculated. The selection of the next generation is influenced by κ. If $\kappa = 1$ the selection is only based on the generated offsprings. Otherwise, if $\kappa = \infty$, the combined set of parents and offsprings undergoes a selection process. In this case, the algorithm ensures that the best solution is not lost. Hence, such algorithms are called *elitist*. For all other values of κ, the selection incorporates all parent individuals that participated in less than the last κ generations. The termination criterion can be of various types, see Beyer and Schwefel [12]. In this work, we always specify a maximum number of generations as our termination criterion.

Algorithm 3.2 $(\mu, \kappa, \lambda, \rho)$-Evolution Strategy

```
$P_{0,\mu} \leftarrow$ initialization, $P_{0,\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
$P_{0,\mu} \leftarrow$ evaluation;
$t \leftarrow 0$;
while (termination criterion not fulfilled) do
  for ($i = 1$ to $\lambda$) do
    {i is a simple counter variable}
    $Pool_i \leftarrow$ marriage $(P_{t,\mu}, \rho)$;
    $\vec{s}_i \leftarrow$ strategy_recombination $(Pool_i)$;
    $\vec{o}_i \leftarrow$ object_recombination $(Pool_i)$;
    $\tilde{\vec{s}}_i \leftarrow$ strategy_mutation $(\vec{s}_i)$;
    $\tilde{\vec{o}}_i \leftarrow$ object_mutation $(\tilde{\vec{s}}_i, \vec{o}_i)$;
    $\tilde{F}_i \leftarrow F(\tilde{\vec{o}})$
  end for
  $P'_{t,\lambda} \leftarrow \left\{\left(\tilde{\vec{o}}_i, \tilde{\vec{s}}_i, \tilde{F}_i\right), i = 1, \ldots, \lambda\right\}$; $\{P'_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})\}$
  if ($\kappa = 1$) then
    $P_{(t+1),\mu} \leftarrow$ selection $(P'_{t,\lambda}, \mu)$;
  else if ($\kappa = \infty$) then
    $P_{(t+1),\mu} \leftarrow$ selection $(P'_{t,\lambda} \cup P_{t,\mu}, \mu)$;
  else
    $P_{(t+1),\mu} \leftarrow$ selection $(P'_{t,\lambda} \cup P_{t,\mu}, \mu, \kappa)$;
    {$P_{(t+1),\mu} \in \mathcal{M}_\mu(\mathbb{I})$ includes only individuals that participate in less than the last $\kappa$
    generations.}
  end if
  $t \leftarrow t + 1$
end while
```

3.1.1 Different Encoding Techniques

Evolution Strategies can be applied to many different kinds of optimization problems. Depending on the type of problem, different encoding techniques for the object parameters are used, see, for instance, Beyer and Schwefel [12]. In this work, we use real-value encoded approaches as well as encoding techniques for combinatorial search spaces. In the following, we will present the three main operators namely the selection, the mutation, and the recombination. During the discussion of these operators, the two different encoding techniques will be introduced.

3.1.2 Selection

The selection operator directs the evolutionary process to promising regions of the object parameter space. In Evolution Strategies normally a deterministic process is used that selects the individuals with the highest fitness values of the current generation to be the parents of the next generation. That is, only those individuals are chosen that provide the highest fitness values, see Bäck and Schwefel [7]. This technique is called *truncation* or *breeding* selection.

In the case of a (μ, λ) selection $(\kappa = 1)$, only the newly generated offspring individuals participate during the selection. Consequently, this selection strategy is non-elitist. Contrary, the $(\mu + \lambda)$ selection $(\kappa = \infty)$ is elitist as the best individuals of all parents and offsprings are used to build the next generation. In all other cases $(1 < \kappa < \infty)$ the selection is not necessarily elitist as only individuals survive the selection process that participated in at most the last κ generations.

Beyer and Schwefel [12] recommend to apply a (μ, λ) selection strategy in unbounded search spaces while the $(\mu + \lambda)$ strategy should be used in discrete finite size search spaces.

As already mentioned, the truncation selection is normally used for Evolution Strategies, see Bäck and Schwefel [7]. However, in some exceptional cases other selection operators, like tournament selection, proportionate selection, and ranking selection are applied. Normally, those selection strategies are often only applied to Genetic Algorithms. Those possible operators are collected by Deb [27]. Here, we will introduce the *binary tournament selection* and the *fitness proportional selection scheme* as both will be used within the multi-objective Evolutionary Algorithms.

The binary tournament selection process chooses two individuals of the current population and the individual with the better fitness is selected to participate within the next generation, see Deb [27, p. 89]. Then, this process is repeated until enough individuals have been selected. The various tournament selection operators differ in the way they choose the next two individuals for a tournament. However, if some bookkeeping is implemented, it can be ensured that each individual participates exactly twice in such a tournament. In this case, the best individual is copied twice to the next generation.

During the fitness proportional selection, individuals are copied to the next generation multiple times. The number of copies is proportional to the fitness of the individuals. Thus, individuals with higher fitness generate more offsprings than individuals with worse fitness, see Bäck et al. [6, C2.2]. We assume that the average fitness over all individuals within the population is F_{avg} and the fitness of a special individual $a_k \in \mathbb{I}$ is F_k. Then the probability of copying this individual to the next generation is F_k/F_{avg}. To this end, two versions of the fitness proportional selection scheme exist, the *roulette wheel selection* and the *universal stochastic sampling*. During the roulette wheel selection, the wheel is divided into partitions corresponding to the population size where the size of each partition is proportional to the fitness of the corresponding individual. Then, the next generation of μ individuals is chosen by sampling μ-times over this generated roulette wheel. Contrary, the stochastic universal sampling determines μ points on the wheel with equal distances between any two neighboring points while ensuring that the whole wheel is used. This leads to μ-points on the wheel and to the corresponding individuals.

The fitness proportional selection scheme has the risk of quickly losing the diversity within the population, if the fitness function is not well chosen.

3.1.3 Mutation of Object and Strategy Parameters

The mutation operator is the main variation operator for Evolution Strategies. Following the guide by Bäck and Schwefel [12] this operator should provide the three characteristics of *reachability*, *unbiasedness*, and *scalability.*
Reachability describes that every solution within the search space can be found after a finite number of mutations or generations. Unbiasedness characterizes the variation operators as the exploring operator. To this end, the variation operators should not have any preferences of directions within the search space. The scalability condition states that the average length of a mutation step should be tuneable in order to adapt to the properties of the fitness landscape [12].
In the following, we will present some mutation operators for real-value as well as combinatorial search spaces that are used in the remainder of this thesis. The mutation operator can be applied to object and strategy parameters in a similar way.

3.1.3.1 Real-valued Coding

The mutation operator for optimization problems that are encoded by using real values frequently uses a normal distribution. Within a population not all individuals are mutated but with a probability of p_m. This is an exogenous parameter and must be specified before the evolutionary process. In most cases, the mutated objective vector $\tilde{\vec{o}_i}$ is generated by adding a vector $\vec{z}$ that consists of samples from a normal distribution. Additionally, a standard deviation σ, which is also called *mutation strength*, for all samples is given as an endogenous strategy parameter. Thus, we can calculate the mutated object vector by:

$$\begin{aligned}\tilde{\vec{o}} &= \vec{o} + \vec{z}, \text{ with} &(3.2)\\ \vec{z} &= \sigma(\mathcal{N}_1(0,1), \dots, \mathcal{N}_u(0,1)). &(3.3)\end{aligned}$$

In this case, the $\mathcal{N}_i(0,1)$, $i \in \{1, \dots, u\}$ are independent random samples from the standard normal distribution, see Beyer and Schwefel [12]. This standard normal distribution has the density function

$$p(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}. \tag{3.4}$$

Hence, each component $\tilde{o}_i$ has the density function

$$p(\tilde{o}_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(\tilde{o}_i - o_i)^2}{\sigma^2}}. \tag{3.5}$$

As the same mutation strength σ is used for all object parameter components, the mutation is *isotropic.*
Depending on the characteristics of the search space, a mutation operator that provides different mutation strengths for the different object vector components is beneficial. In this *non-isotropic* case, the Evolution Strategy consists of v endogenous strategy parameters instead of one. As already mentioned, in most cases $u = v$. Hence, the additive vector is calculated by:

$$\vec{z} = (\sigma_1 \mathcal{N}_1(0,1), \dots, \sigma_u \mathcal{N}_u(0,1)). \tag{3.6}$$

As mentioned above, several possible approaches exist to implement the mutation for real-value optimization problems. In our work, we use the *polynomial mutation*, see Deb [27, p.124] and Deb et al. [31], as well. In this case, a polynomial function is applied as the probability distribution function instead of a normal distribution. Thus, the components of an objective vector are varied by:

$$\tilde{o}_i = o_i + \left(o_i^{(U)} - o_i^{(L)}\right)\delta_i. \tag{3.7}$$

Here, $o_i^{(U)}$ and $o_i^{(L)}$ correspond to the upper and lower bound of the values of component i of the object value. These bounds are exogenous parameters and must be specified before the optimization process. The parameter δ_i is calculated for a random variable $r_i \in [0,1]$ (rectangular distribution):

$$\delta_i = \begin{cases} (2r_i)^{1/(\eta_m+1)} - 1, & \text{if } r_i < 0.5 \\ 1 - [2(1-r_i)]^{1/(\eta_m+1)}, & \text{if } r_i \geq 0.5 \end{cases}. \tag{3.8}$$

The parameter η_m is problem specific and has to be chosen carefully.
The *self-adaptation* of the strategy parameter is an important aspect of Evolution Strategy and has a high influence on the mutation operator. This will be introduced in detail in Section 3.1.5.

3.1.3.2 Coding as Permutation

The mutation of permutations is not trivial. It must be ensured that at any time the mutation generates a valid permutation. To this end, we use three elementary operations: *move*, *swap*, and *jump*. During the mutation of an individual only one of these elementary operations is applied.

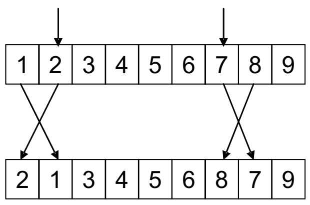


Figure 3.1: The effect of the move mutation operator. Each numbered cell indicates an object component. Jobs number 2 and 7 are moved.

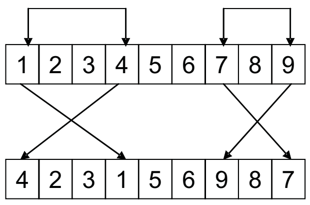


Figure 3.2: The effect of the swap mutation operator. Each numbered cell indicates an object component. The jobs 1 and 4 are swapped as well as the jobs 7 and 9. In the first case, the sampling from the mutation jump length distribution produces the value 3 and in the second case 2.

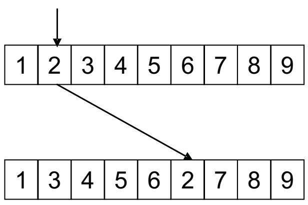


Figure 3.3: The effect of the jump mutation operator. Each numbered cell indicates an object component. Job number 2 jumps 4 positions.

The move operation exchanges a randomly chosen element with one of its neighbors, see Figure 3.1. The swap operation swaps two arbitrarily chosen entries as displayed in Figure 3.2. The jump operation moves a randomly selected element of the sequences for a given number of jobs, see Figure 3.3.
For all three basic operators we use the mutation number (MN) as an endogenous strategy parameter. As we will show later, this parameter influences the number of basic mutation operations that are applied to one individual. Furthermore, the distance between jobs during the *swap* operation and the distance a job should be moved during the *jump* operation are parameterized by the endogenous strategy parameter mutation jump length (MJL). Note that the direction of the movement of a job within the sequence for all three basic operators is chosen randomly.
In detail, the number of mutations as well as the distance to move or swap jobs are geometrically distributed as recommended by Rudolph [129] for such permutation problems. The means of these two distributions are described by the *mutation number* (MN) and the *mutation jump length* (MJN). Both endogenous parameters are subject to the self-adaptation described in Section 3.1.5.
The number of mutations for an individual are calculated by

$$number\_of\_mutations = \left\lfloor \frac{ld(1-\varsigma_1)}{ld(1-p_{MN})} \right\rfloor, \text{ with} \tag{3.9}$$

$$p_{MN} = 1 - \frac{MN}{1+\sqrt{1+MN^2}}. \tag{3.10}$$

The value ς_1 is a random variable: $\varsigma_1 \in [0, 1[$ (rectangular distribution). Correspondingly, the number of jobs to move or swap a job (*distance*) is calculated as

$$distance = \left\lfloor \frac{ld(1-\varsigma_2)}{ld(1-p_{MJL})} \right\rfloor, \text{ with} \tag{3.11}$$

$$p_{MJL} = 1 - \frac{MJL}{1+\sqrt{1+MJL^2}}. \tag{3.12}$$

Again, the value ς_2 is a random variable in the interval of $[0, 1[$ (rectangular distribution).

3.1.4 Recombination of Object and Strategy Parameters

During the recombination, a new offspring is generated by combining ρ parents. Contrary to the recombination operator of Genetic Algorithms, the recombination in Evolution Strategies only produces one offspring. Evolution Strategies normally use the *discrete recombination* or the *intermediate recombination*. Both types are displayed in Figure 3.4.

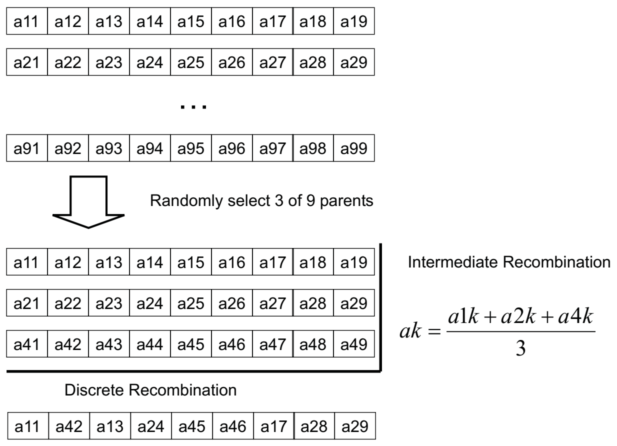

a11 a12 a13 a14 a15 a16 a17 a18 a19
a21 a22 a23 a24 a25 a26 a27 a28 a29
...
a91 a92 a93 a94 a95 a96 a97 a98 a99
Randomly select 3 of 9 parents
a11 a12 a13 a14 a15 a16 a17 a18 a19
a21 a22 a23 a24 a25 a26 a27 a28 a29
a41 a42 a43 a44 a45 a46 a47 a48 a49
Intermediate Recombination
ak = (a1k + a2k + a4k)/3
Discrete Recombination
a11 a42 a13 a24 a45 a46 a17 a28 a29

Figure 3.4: Standard (multi-)recombination operators in Evolution Strategies (taken from Beyer and Schwefel [12]). In this case we assume to have a population of $\mu = 9$ individuals where $\rho = 3$ individuals are selected for recombination.

In the following, we denote each individual $\vec{a}_k$ as a vector of object and/or strategy of D parameters: $\vec{a}_k = (a_{k1}, a_{k2}, \ldots, a_{kD})$.
A *discrete recombination* first selects ρ from the μ individuals of the current population. Then, the object vector components at position i, $\tilde{a}_{ki}$, of the offspring $\tilde{\vec{a}}_k$ are determined by randomly selecting one a_{ji}, $j \in 1, \ldots, \rho$ from the selected ρ parents.

$$(\tilde{a}_k)_i = (a_j)_i, \text{ with } j \text{ randomly in } \{1, \ldots, \rho\} \tag{3.13}$$

The *intermediate recombination* determines the object vector component $\tilde{a}_{ki}$ of the offspring $\tilde{\vec{a}}_k$ by averaging the a_{ji} values of all ρ selected individuals.

$$(\tilde{a}_k)_i = \frac{1}{\rho} \sum_{j=1}^{\rho} (a_j)_i \tag{3.14}$$

The *Simulated Binary Crossover (SBX)* as described by Deb et al. [28, 31] is uncommon for standard Evolution Strategies. However, this recombination operator is applied in multi-objective Evolutionary Strategies. Thus, it will be introduced briefly.
The SBX operator uses $\rho = 2$ parents to produce two offsprings. Further, an exogenous parameter $\eta_c \geq 0$ has to be chosen. The higher this value the nearer the offsprings lie beside one of the parents, see Deb [27, p. 113]. Assuming that we have selected two parent individuals a_1 and a_2, we need to apply Algorithm 3.3 for each vector component to generate two offsprings $\tilde{a}_1$ and $\tilde{a}_2$.
In most cases, the discrete recombination is applied to the object parameters whereas the intermediate recombination is used for the strategy parameters, see Beyer and Schwefel [12]. These three recombination operators directly fit to real-value coded optimization problems. However, for permutation coded problems the recombination operator is difficult to establish. In this case, neither the random selection of object vector components nor the averaging of these components is possible.
For permutation coded approaches, it must be ensured that each element exists exactly once. In this work we therefore use a different recombination, displayed in Figure 3.5.

Algorithm 3.3 Simulated Binary Crossover (SBX).

```
Choose a random number $u_i \in [0, 1[$ {rectangular distribution}
Calculate $\beta = \begin{cases} (2u_i)^{\frac{1}{\eta_c+1}}, & \text{if } u_i \leq 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise.} \end{cases}$
$\tilde{a}_{1i} = 0.5\,[(1+\beta)a_{1i} + (1-\beta)a_{2i}]$
$\tilde{a}_{2i} = 0.5\,[(1-\beta)a_{1i} + (1+\beta)a_{2i}]$
```

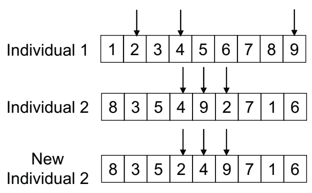


Figure 3.5: Recombination of individuals for permutation coded approaches.

In this case, the recombination is restricted to $\rho = 2$ parents that are randomly selected over all μ individuals. Then, we select MN components of the object vector of the first individual. The number of selected components is equivalent to the number of components that are varied during the mutation operator, that is, we use the same endogenous parameter MN. The second object vector includes the same elements, as both of these vectors are just permutations. Hence, the position of the components that are selected in the first object vector can be determined in the second object vector. The recombination then orders the determined components in the same order as they occur in the first object vector. Thus, the recombination operator tries to establish the same sequence of object vector components as in the first object vector.

3.1.5 Adaptation of Endogenous Strategy Parameters

The success of applying Evolution Strategies is highly connected with the adaptation of the strategy parameters. This enables the algorithm to react on changing solution spaces. Especially the mutation operator is influenced by the adaptation of the mutation strength(s) σ of Equations 3.3 and 3.6. The right setting for these mutation strengths is important for the success of the Evolution Strategy. Too high mutation strength values lead to the effect that optima cannot be found as the algorithm is not sensitive enough to search in the near environment of existing solutions. In contrast to this, if the mutation strength is too low, the process of reaching optima is very slow and sometimes the process is not able to overcome local optima. Thus, the mutation strengths need to be adapted corresponding to the local environment within the search space.
The adaptation of endogenous strategy parameters is not limited to the mutation strengths. The mutation number of Equation 3.10 and the mutation jump length of Equation 3.12 are also adapted.

3.1.5.1 The 1/5th Rule

The 1/5th rule was developed by Rechenberg [126] for a (1+1)-Evolution Strategy. This rule expresses that the mutation strength σ must be adapted according to the success of the last

mutations.
In detail, the success of the last mutations is counted while the mutation strength σ is constant for several generations. A mutation is successful if the mutated individuals have better fitness values compared to the parents. Then, the success probability P_s is calculated by

$$P_s = \frac{\text{Number of successful mutations}}{\text{Number of all mutations}}. \tag{3.15}$$

The mutation strength σ is changed according to

$$\tilde{\sigma} = \begin{cases} \sigma/a, & \text{if } P_s > 1/5 \\ \sigma \cdot a, & \text{if } P_s < 1/5 \\ \sigma, & \text{if } P_s = 1/5 \end{cases}. \tag{3.16}$$

Schwefel [134] recommends the use of a value $0.85 \leq a < 1$. Then the mutation strength is again kept constant for several generations before the adaptation process is restarted.
The 1/5th rule has several disadvantages as it is normally restricted to applications with a single strategy parameter. Further, it is in most cases only used for (1+1)-Evolution Strategies.

3.1.5.2 Self-Adaptation

The main idea of self-adaptation is the direct combination between object and strategy parameters. Hence, each individual has its own strategy parameters, see Equation 3.1. The strategy parameters are sometimes recombined and always mutated. Then, the mutation strategy parameters control the mutation of the individual's object parameters. As the selection of individuals is based on their fitness and higher fitness values correspond with better object vectors that are influenced by their strategy parameters, the probability of individuals with better strategy parameters to survive the evolution is also higher, see Beyer and Schwefel [12].
The adaptation of the mutation strength σ is realized by multiplying σ by a positive number. This ensures that the resulting mutation strength $\tilde{\sigma}$ is also positive.
In general, two different cases are analyzed in the literature. In the first case, a single mutation strength is used to adapt all object parameters, see Equation 3.3. In the second case, each object parameter has its own mutation strength, see Equation 3.6.

3.1.5.2.1 Single Mutation Strength (Isotropic Mutation) Schwefel [134] suggest to use a sample from a normal distribution $\mathcal{N}(0,1)$ and calculating the new mutation strength $\tilde{\sigma}$ using the logarithmic normal distribution with the result

$$\tilde{\sigma} = \sigma \cdot e^{\tau \mathcal{N}(0,1)}. \tag{3.17}$$

The exogenous strategy parameter τ is called *learning rate* and should be set to

$$\tau \propto \frac{1}{\sqrt{u}} \tag{3.18}$$

for u object parameters, see Beyer and Schwefel [12]. For highly multimodal fitness landscapes, see Beyer and Schwefel [12], a smaller learning rate should be tried

$$\tau \propto \frac{1}{\sqrt{2 \cdot u}}. \tag{3.19}$$

Another even simpler approach is Rechenberg's *two-point rule* [127]. In this case, a random variable $r \in \,]0,1]$ is used to calculate the new mutation strength $\tilde{\sigma}$

$$\tilde{\sigma} = \begin{cases} \sigma \cdot \alpha, & \text{if } r \leq 0.5 \\ \sigma / \alpha, & \text{if } r > 0.5 \end{cases}, \ \alpha > 1. \tag{3.20}$$

The parameter α must be chosen individually.

3.1.5.2.2 Mutation Strength Vector (Non-Isotropic Mutation) The basic concept of adapting the mutation strength can be extended to strategy parameter vectors. The new strategy parameter vector $\tilde{\vec{s}}_k = \tilde{\vec{\sigma}} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_u)$ (assuming $u = v$) is calculated by

$$\tilde{\vec{\sigma}} = e^{\tau_0 \mathcal{N}(0,1)} \cdot \left(\sigma_1 \cdot e^{\tau_1 \mathcal{N}_1(0,1)}, \ldots, \sigma_u \cdot e^{\tau_1 \mathcal{N}_u(0,1)} \right). \tag{3.21}$$

In this case, each mutation strength is mutated individually and the resulting strategy parameter vector is scaled by a random factor. Note that the general term $e^{\tau_0 \mathcal{N}(0,1)}$ is determined once for all individuals of a generation by sampling the normal distribution $\mathcal{N}(0,1)$. Contrary, all normal distribution samples $\mathcal{N}_i(0,1)$, $i \in \{1, \ldots, u\}$ (remember that we assume $u = v$), are determined separately for each individual. Beyer and Schwefel [12] claim that "this technique allows for learning axes-parallel mutation ellipsoids". The two exogenous learning rates τ_0 and τ_1 should be set to

$$\tau_0 = \frac{1}{\sqrt{2 \cdot u}}, \text{ and } \tau_1 = \frac{1}{\sqrt{2\sqrt{u}}}. \tag{3.22}$$

3.2 Multi-Objective Evolutionary Algorithms

In this section, the concept of using single-objective Evolutionary Algorithms to solve optimization problems with a single-objective is extended to the optimization of multi-objective problems. As described in Section 2.5, the development of scheduling strategies has to incorporate multiple objectives. The goal during the multi-objective optimization process is to find as many Pareto optimal solutions as possible. Further, those solutions should be spread over a wide area of the solution space. Thus, the goal of the multi-objective optimization is to solve problems with the objective $\#(Z_1, Z_2, \ldots, Z_k)$ as defined in Section 2.5.1. The classical approaches, presented in Section 2.5.4, have several disadvantages. First, they can only generate one solution at a time. Second, they need some pre-defined weights or ϵ-values for the optimization process. However, those parameters are unknown and may lead to a bad approximation of the Pareto front. Further, the usage of such parameters in fact transforms the multi-objective problem into a single-objective problem.

Multi-objective Evolutionary Algorithms use a population of individuals each representing a solution. Hence, multiple potential Pareto optimal solutions are included within a population. In contrast to the classical methods that result in a single solution, the whole optimization process generates an approximation of the Pareto front. Furthermore, multi-objective Evolutionary Algorithms do not require the definition of weights which leads to a more robust generation of the Pareto front.

All non-dominated solutions are processed in the same way by using Multi-objective Evolutionary Algorithms. So, they do not assume any prioritizations of individual simple objectives. Further, they gradually move the population towards the Pareto front.

Zitzler et al. [167] present the major problems that must be addressed when multi-objective Evolutionary Algorithms are applied:

- How to accomplish fitness assignment and fitness selection in order to guide the search towards the Pareto optimal set (convergence)?
- How to maintain a diverse population in order to prevent premature convergence and achieve a well distributed trade-off front (coverage)?

Several multi-objective Evolutionary Algorithms have been published, see, for example, the collection by Deb [27]. They are often divided into two main groups. The first group consists of non-elitist algorithms where the selection operator does not ensure that non-dominated solutions are transferred to the next generation. The second group describes all elitist algorithms that enforce the transmission of non-dominated solutions to the next generations. Both groups will be presented in the following as our scheduling strategy development will use both attempts.

3.2.1 Non-Elitist Multi-Objective Evolutionary Algorithms

Non-elitist algorithms do not ensure that the best individuals of the current population participate in the next generation. However, the probability for a good individual to survive the selection procedure is higher than for individuals with a low fitness.
Several different non-elitist algorithms have been proposed to generate the Pareto front. Examples are the Vector Evaluated Genetic Algorithm (VEGA) by Schaffer [132] which was the first multi-objective Evolutionary Algorithm. Other approaches are the Vector-Optimized Evolution Strategy (VOES) by Kursawe [95], the Weight-Based Genetic Algorithm (WBGA) by Hajela and Lin [72], the Random Weighted Genetic Algorithm (RWGA) by Murata and Ishibuchi [113], the Multiple Objective Genetic Algorithm (MOGA) by Fonseca and Fleming [61], the Niched-Pareto Genetic Algorithm (NPGA) by Horn et al. [81], the Predator-Prey Evolution Strategy by Laumanns [97], and the Non-dominated Sorting Genetic Algorithm (NSGA) by Srinivas and Deb [150].
The NSGA algorithm has proven to be a powerful multi-objective Evolutionary Algorithm which is easy to use. It is used as our first approach of generating the Pareto front. Thus, we present this algorithm in more detail.

3.2.1.1 Non-Dominated Sorting Genetic Algorithm (NSGA)

The Non-dominated Sorting Genetic Algorithm (NSGA) varies from a simple Evolutionary Algorithm only in the way the selection operator is used to direct the search to the Pareto front while maintaining a high diversity of the generated solutions within the population. The recombination and mutation operators remain as usual.
Algorithm 3.4 describes the whole NSGA algorithm. First, μ initial individuals are randomly generated and further evaluated for each of the k objective functions f_i, $i \in \{1, \ldots, k\}$. Then, the fitness of each individual is calculated (see Algorithm 3.5). This procedure incorporates the non-domination level of the individuals and their corresponding diversity. In the following, the evolutionary loop is started for a fixed number of generations.
Within this loop a fitness proportional selection, see Section 3.1.2, is used to generate λ individuals. Then, the strategy parameters of the individuals are mutated by using Rechenberg's

Algorithm 3.4 Non-Dominated Sorting Genetic Algorithm (NSGA).

```
$P_{0,\mu} \leftarrow$ initialization, $P_{0,\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
for ($i = 1$ to $k$) do
  $P_{0,\mu} \leftarrow$ evaluate objective function $f_i$;
end for
$P_{0,\mu} \leftarrow$ NSGA fitness assignment {See Algorithm 3.5};
$t \leftarrow 0$;
for ($t = 0$ to (Number of Generations - 1)) do
  $P'_{t,\lambda} \leftarrow$ fitness proportional selection($P_{t,\mu}$), $P'_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $P''_{t,\lambda} \leftarrow$ two point mutation of strategy parameters($P'_{t,\lambda}$), $P''_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $P'''_{t,\lambda} \leftarrow$ recombination of object parameters($P''_{t,\lambda}$), $P'''_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $P''''_{t,\lambda} \leftarrow$ mutation of object parameters($P'''_{t,\lambda}$), $P''''_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  for ($i = 1$ to $k$) do
    $P''''_{t,\lambda} \leftarrow$ evaluate objective values $f_i$;
  end for
  $P''''_{t,\lambda} \leftarrow$ NSGA fitness assignment {See Algorithm 3.5};
  $P_{(t+1),\mu} \leftarrow$ select($P''''_{t,\lambda}$), $P_{(t+1),\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
  $t \leftarrow t + 1$;
end for
```

two-point rule, see Section 3.1.5.2.1. In the following, the individuals are in some cases recombined and always mutated. Subsequently, the modified individuals within the population are evaluated by using all k objective functions and the NSGA fitness assignment procedure is used to determine the individual's fitness values, see Algorithm 3.5. Next, the NSGA algorithm selects the μ best individuals from all modified individuals to become the parents of the next generation. As NSGA only uses the modified individuals, the whole procedure is non-elitist.

The presented main structure of the NSGA does not specify the fitness assignment of individuals. This fitness assignment uses dominance levels and the contribution of individuals to the diversity of the overall population. Algorithm 3.5 describes this fitness assignment. By $F(a)$ we denote the fitness of individual a. The k objective function values of individual a are denoted by $f_i(a)$ for $k \in \{1, \ldots, k\}$. First, a small positive number ϵ must be chosen. Then, the default fitness level F_{min} is set to the number of individuals λ plus ϵ. Next, the current population P_t is divided into Υ subsets that are mutually non-dominated by using Definition 2 (on page 16). Thus, the subset $P_t^{(1)}$ is the Pareto front of P_t and rank 1 is assigned to this subset. $P_t^{(2)}$ is the Pareto front of $P_t \backslash P_t^{(1)}$ with rank 2. The same concept is applied to all following subsets. Within the resulting subsets each solution from a subset with a higher rank is dominated by at least one solution of every lower ranked subset. The solutions within one subset are preferable compared to solutions with higher rank numbers (remember that rank 1 consists of the best solutions). Figure 3.6 shows an example of such a ranking of the various solutions. Furthermore, the selection process needs a differentiation between solutions within the same subset. Thus, the contribution to the population's diversity is used to distinguish between those solutions.

In detail, the algorithm starts at rank 1 and processes each rank separately until the last rank Υ is reached. For all individuals a the dummy fitness F^r is assigned which is the lowest fitness of the lower ranks decreased by the chosen ϵ. Then the niche count for each individual is

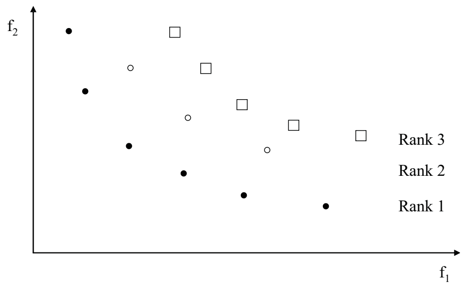


Figure 3.6: Illustration of the non-dominated sorting procedure. The specified rank is assigned to the solutions by the non-dominated sorting.

Algorithm 3.5 NSGA Fitness Assignment.

```
Choose a small positive number $\epsilon$;
$F_{min} = \lambda + \epsilon$;
Classify population $P_t$ according to non-domination: $\left(P_t^{(1)}, P_t^{(2)}, \ldots, P_t^{(\Upsilon)}\right) = Sort\,(P_t, \preceq_p)$
{We assume that $\Upsilon$ fronts exist};
for ($j = 1$ to $\Upsilon$) do
  $F_{newMin} = F_{min}$;
  for all ($a \in P_t^{(j)}$) do
    Assign rank fitness $F^r(a) = F_{min} - \epsilon$;
    Calculate niche count $nc_a$ among solutions of $P_t^{(j)}$ only; {See Algorithm 3.6.}
    Calculate fitness $F(a) = \frac{F^r(a)}{nc_a}$;
    $F_{newMin} = \min(F(a), F_{newMin})$;
  end for
  $F_{min} = \min(F_{newMin}, F_{min})$;
end for
```

calculated. The smaller this niche count the higher the contribution of the particular individual to the diversity of the overall population. Hence, the previously assigned dummy fitness is adjusted with this niche count nc_a of each individual a. After the fitness values of all individuals in the current subset are determined, the lowest fitness is readjusted.
The calculated fitness values can then be used for the selection process of the NSGA algorithm, see Algorithm 3.4. The last detail, namely the calculation of the niche count nc, is presented next. The original work by Srinivas and Deb [150] uses a concept of calculating distances between solutions that are filtered by a pre-defined σ_{share} value. Only if the distance between the solutions is smaller than this value, the pair is used to calculate the niche count. However, this value σ_{share} is critical as it is problem dependent. Hence, we exchanged this method to calculate the niche count and used the method by Laumanns [96] that does not require any pre-defined values. Algorithm 3.6 presents the main idea behind this approach. However, here we will not describe this procedure in detail. For further information, see Laumanns [96].

3.2.2 Elitist Multi-Objective Evolutionary Algorithms

The second category of multi-objective Evolutionary Algorithms consists of all elitists methods. Here, several possible approaches have been published. These are Rudolph's Elitist

Algorithm 3.6 Laumann's niche count estimator, see Laumanns [96].

```
Part I: Compute niche radius $q$
$minNiches = k$ {k is the number of objectives};
$maxNiches = \lambda$ {$\lambda$ is the number of individuals};
$niches = \lfloor (maxNiches - 0.5 \cdot (maxNiches - minNiches)) + 0.5 \rfloor$;
$divisions = \left\lfloor \left( (niches/k)^{1/(k-1)} \right) + 0.5 \right\rfloor$;
$h = 0$ {h is a help variable};
for ($i = 1$ to $k$) do
  $V_l$ = lowest value of all individuals regarding objective $f_i$;
  $V_h$ = highest value of all individuals regarding objective $f_i$;
  $h = h + ((V_h - V_l)/divisions)^2$;
end for
$q = \sqrt{h}$;
Part II: Compute niche count $nc$ of individuals
for ($j = 1$ to $\lambda$) do
  $d = 0$;
  for ($l = 1$ to $\lambda$) do
    if ($|\vec{f}(j) - \vec{f}(l)|/q < 1$) then
      $d = d + 2\pi(1 - (|\vec{f}(j) - \vec{f}(l)|/q)^2)$ {$| \cdot |$ specifies the Euclidean Distance.}
    end if
  end for
  $nc_j = d/(\lambda/q^k)$
end for
```

Multi-Objective Evolutionary Algorithm [130], the Distance-Based Pareto Genetic Algorithm (DPGA) by Osyczka and Kundu [118], the Strength Pareto Evolutionary Algorithm (SPEA2) by Zitzler et al. [169], the Termodynamical Genetic Algorithm by Kita et al. [92], the Pareto-Archived Evolution Strategy (PAES) by Knowles and Cornes [93], the Multi-Objective Messy Genetic Algorithm (mGA) by Veldhuizen and Lamont [160], the S metric selection Evolutionary Multi-Objective Evolutionary Algorithm (SMS-EMOA) by Emmerich et al. [39], the Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) by Deb at al. [29], and the Indicator Based Evolutionary Algorithm (IBEA) by Zitzler and Künzli [168].
In our work we have used the NSGA-II algorithm as this extends the main ideas of NSGA. However, the IBEA algorithm has proven to work better than NSGA-II on most multi-objective problems. Unfortunately, IBEA was published after we applied NSGA-II to our scheduling problem and hence IBEA has not been used. In the following, we will present the main concept of NSGA-II.

3.2.2.1 Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II)

The non-elitist NSGA Algorithm, see Section 3.2.1.1, has three major disadvantages. First, the non-dominated sorting of NSGA has a complexity of $O(k\mu^3)$, where k objectives and μ individuals are used (see Deb et al. [29]). Second, the algorithm is non-elitist and hence might loose good solutions. Third, the original procedure to calculate the niche count needs a pre-defined parameter σ_{share} (we have exchanged this method already).
The further development of NSGA-II uses a *fast non-dominated sorting* procedure with com-

Algorithm 3.7 Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II).

```
$P_{0,\mu} \leftarrow$ initialization, $P_{0,\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
for ($i = 1$ to $k$) do
  $P_{0,\mu} \leftarrow$ evaluate using objective function $f_i$;
end for
Assign ranks to all individuals of population $P_{0,\mu}$ according to non-domination;
$P'_{0,\lambda} \leftarrow$ binary tournament selection based on the rank($P_{0,\mu}$), $P'_{0,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
$P''_{0,\lambda} \leftarrow$ recombination($P'_{0,\lambda}$), $P''_{0,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
$Q_{0,\lambda} \leftarrow$ mutation($P''_{0,\lambda}$), $Q_{0,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
for ($i = 1$ to $k$) do
  $Q_{0,\lambda} \leftarrow$ evaluate using objective function$f_i$;
end for
$t \leftarrow 0$;
for ($t = 0$ to (Number of Generations-1)) do
  $R_{t,\mu+\lambda} = P_{t,\mu} \cup Q_{t,\lambda}$;
  Classify population $R_{t,\mu+\lambda}$ according to non-domination:
  $\left(R_t^{(1)}, R_t^{(2)}, \ldots, R_t^{(\Upsilon)} = Sort(R_{t,\mu+\lambda}, \preceq_p\right)$ {We assume $\Upsilon$ fronts exist};
  $P_{(t+1),\mu} = \emptyset$;
  $c = 1$;
  while $((|P_{(t+1),\mu}| + |R_t^{(c)}| \leq \mu))$ do
    crowding distance assignment($R_t^{(c)}$) {see Algorithm 3.8};
    $P_{(t+1),\mu} = P_{(t+1),\mu} \cup R_t^{(c)}$;
    $c = c + 1$;
  end while
  $Sort(R_t^{(c)}, \preceq_n)$ {sort by crowded comparison operator, see Algorithm 3.9};
  $P_{(t+1),\mu} = P_{(t+1),\mu} \cup R_t^{(c)}[1 : (\mu - |P_{(t+1),\mu}|)]$ {the best $(\mu - |P_{(t+1),\mu}|)$ elements of $R_t^{(c)}$};
  $P'_{(t+1),\lambda} \leftarrow$ binary tournament selection($P_{(t+1),\mu}$), $P'_{(t+1),\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $P''_{(t+1),\lambda} \leftarrow$ mutation of strategy parameters($P'_{(t+1),\lambda}$), $P''_{(t+1),\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $P'''_{(t+1),\lambda} \leftarrow$ recombination($P''_{(t+1),\lambda}$), $P'''_{(t+1),\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  $Q_{(t+1),\lambda} \leftarrow$ mutation($P'''_{(t+1),\lambda}$), $Q_{(t+1),\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
  for ($i = 1$ to $k$) do
    $Q_{(t+1),\lambda} \leftarrow$ evaluate using objective function $f_i$;
  end for
  $t \leftarrow t + 1$;
end for
```

Algorithm 3.8 NSGA-II Crowding distance assignment procedure for set R.

```
$\Gamma = |R|$ {We assume $\Gamma$ individuals in set $R$};
for all $(i \in R)$ do
    $R[i]_{distance} = 0$ {for each individual $i$ set the crowding distance to 0};
end for
for $(o = 1$ to $k)$ do
    Sort $R$ according to increasing objective function $f_o$: $(R[1_{f_o}], R[2_{f_o}], \ldots, R[\Gamma_{f_o}]) = Sort(R, o)$;
    $V_{f_o}^{max} = R[\Gamma_{f_o}]_{f_o}$;
    $V_{f_o}^{min} = R[1_{f_o}]_{f_o}$;
    $R[1_{f_o}]_{distance} = R[\Gamma_{f_o}]_{distance} = \infty$ {first and last element have distance $\infty$};
    for $(j = 2$ to $(\Gamma - 1))$ do
        $R[j_{f_o}]_{distance} = R[j_{f_o}]_{distance} + \frac{R[(j-1)_{f_o}]_{f_o} - R[(j+1)_{f_o}]_{f_o}}{V_{f_o}^{max} - V_{f_o}^{min}}$;
    end for
end for
```

plexity $O(k\mu^2)$, see Deb et al. [29]. Further, a plus strategy is used that ensures that the best individuals survive the selection process. Moreover, a combined ranking and crowding distance approach is used to ensure the diversity of the individuals within the population. Hence, it is not necessary to pre-define any parameters.
Algorithm 3.7 presents the main idea of NSGA-II. Initially, a random population $P_{0,\mu}$ is generated and then evaluated using all k objectives. Next, all individuals are ranked according to the non-domination, see Definition 2 (on page 16). Subsequently, the binary tournament selection, see Section 3.1.2, followed by recombination and mutation is applied to generate the first child population $Q_{0,\lambda}$. This child population is evaluated by again using all k objectives. Following, the main loop for a pre-defined number of generations is started. Within this loop, the parent and child populations are combined to $R_{t,\mu+\lambda}$. This combined population is classified into Υ fronts using Definition 2. This concept is displayed in Figure 3.6. Then, the next parent generation $P_{(t+1),\mu}$ is generated by selecting the best individuals of $R_{t,\mu+\lambda}$ that is choosing all individuals from $R_t^{(c)}$ for increasing c as long as the number of individuals in $P_{(t+1),\mu}$ is smaller than or equal to μ. Furthermore, for all individuals the crowding distance, see Algorithm 3.8, is calculated. Within this algorithm we denote by $R[i]_{distance}$ the crowding distance of the i-th individual of the given set R. Further, we specify by $R[i]_{f_o}$ the objective function value of individual i regarding function f_o.
If the number of individuals in the last $R_t^{(c)}$, in Algorithm 3.7, is larger than $\mu - |P_{(t+1),\mu}|$ the individuals of $R_t^{(c)}$ are sorted by using the crowded comparison operator, see Algorithm 3.9. Next, the best $\mu - |P_{(t+1),\mu}|$ individuals, regarding the crowded comparison operator, are added to $P_{(t+1),\mu}$.
Subsequently, the child generation $Q_{(t+1),\lambda}$ is generated by applying a binary tournament selection, mutating the strategy parameter, recombining the parent individuals, and mutating the outcomes. At the end, this child generation is again evaluated by using all k objective functions. Then, the next iteration of the loop is started.
The main differences between NSGA and NSGA-II are the crowding distance assignment and the comparison operator $\preceq_n$. The crowding distance uses the cuboid around each solution, see Figure 3.7. The solutions with the biggest cuboids have the highest impact on the diversity

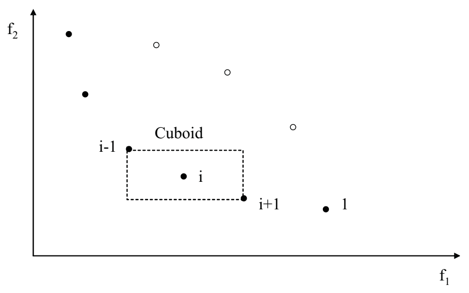


Figure 3.7: Illustration of the crowding distance sorting of members on the Pareto front. The circumference of the box touching neighboring solutions is the ranking criterion. Note that extremal solutions are always preferred to non-extremal solutions.

of the population and hence get a higher fitness. Furthermore, the extreme solutions always receive a high fitness as they reflect the explored solution space.
In detail, the crowding distance assignment procedure is shown in Algorithm 3.8. For each objective function f_o, $o = 1$ to k, the individuals of a current set are sorted. The first and the last individual of the sorted set receive a very high crowded distance value as they generate a high diversity of the given set. All individuals j in between calculate the distance of the neighboring individuals $(j-1)$ and $(j+1)$ regarding the objective o. This distance is normalized by the maximum distance between two individuals within the given set, regarding f_o. The resulting value is added to the crowding distance value of individual j.
As the crowding distances are summed up for all given objectives k the resulting sum for each individual reflects its contribution to the diversity of the whole set. The rank corresponding to the non-domination definition together with this crowding distance is used to sort individuals. The resulting operator is called the crowded comparison operator $\preceq_n$.

Algorithm 3.9 NSGA-II Crowded Comparison Operator $\preceq_n$ for two individuals a_1 and a_2.

```
$a_{1,rank}$ denotes the rank of $i$;
$a_{1,distance}$ denotes the crowding distance, see Algorithm 3.8;
if ($a_{1,rank} < a_{2,rank}$) then
    $a_1 \preceq_n a_2$
else if ($a_{2,rank} < a_{1,rank}$) then
    $a_2 \preceq_n a_1$
else if ($a_{1,distance} > a_{2,distance}$) then
    $a_1 \preceq_n a_2$
else
    $a_2 \preceq_n a_1$
end if
```

This operator is given in detail in Algorithm 3.9. For two individuals of a population a_1 and a_2, the individual with the lower rank has a higher fitness. If both individuals share the same rank, the individual with the higher crowding distance is preferred. This operator is used to select the individuals to form the next parent population within the evolutionary loop in Algorithm 3.7.

Chapter 4

Parallel Job Scheduling

The general problem of scheduling high performance parallel computers was already introduced in Chapter 1. Hence, this chapter will not repeat the general problem structure but present the state of the art of scheduling on Massively Parallel Processors (MPP).
The scheduling system is an important component of a parallel computer. Here, the applied scheduling strategy has direct impact on the overall performance of the computer system with respect to the scheduling policy and objective. For example, the number of idle resources could be reduced and furthermore the systems utilization and job throughput could be increased. The increase in efficiency can be compared with saving additional hardware.
As the scheduling component within massively parallel processors highly influences the overall system performance, the machine providers are often involved in the development process of better working algorithms, see Krallmann et al. [94]. Contrary, the manufacturers of such massively parallel systems show only a limited interest in the development of new scheduling algorithms. In most cases they argue that the customers only buy their systems as the integrated hardware is very powerful. Nevertheless, the machine providers are still not satisfied with the existing scheduling algorithms. Therefore, the research within this area is still ongoing and growing.
In the following, the scheduling environment will be specified in detail. This is followed by the introduction of scheduling algorithms that are nowadays used at real installations.

4.1 Machine Environment

Existing massively parallel processor installations differ in the available hardware. However, most systems consist of several computational nodes with one or more processors, main memory and a local hard disc. A collection of the most powerful systems in 2004 can be found by Ernemann et al. [45].
As the development of the underlying scheduling system is mainly independent of the real hardware specifications of individual computational nodes, we assume in this work m parallel identical nodes. All nodes inside a machine are connected with a fast interconnection network that does not favor any communication pattern inside the machine. This means that a parallel job can be allocated on any subset of nodes of a machine. This comes reasonably close to real systems like an IBM RS/6000 scalable parallel computer, a Sun Enterprise 10000, or an HPC cluster. Furthermore, all nodes support space-sharing and run the jobs in an exclusive fashion.

Many real systems use resource partitions in order to provide different services for different users. In such environments, some user groups can only access a limited partition of the whole system, see, for example, Feitelson et al. [58]. This implements some kind of prioritization. However, partitions normally decrease the overall system performance, see Feitelson [55]. We therefore assume that all nodes can be used by all users in any combination. The scheduling algorithm itself has to implement the local prioritization scheme.

4.2 System Characteristics

The scheduling system at massively parallel computers has to establish node allocations for computational jobs of the job system τ. Within this *online* scenario, different users are submitting jobs $j \in \tau$ over time. The release date $r_j \geq 0$ of job j is not known in advance. This is denoted by $\bar{r}_j$. Additionally, the processing time $p_j > 0$ is unknown. The users only provide a rough estimate $\bar{p}_j$ for p_j. In practice, this estimate is used to cancel job j if $p_j > \bar{p}_j$, i.e. if the actual execution length of job j exceeds the estimate. In order to make sure that a job without errors is not cancelled just before its completion, users tend to overestimate the execution time [98]. Therefore, $\bar{p}_j$ is of limited help for the purpose of schedule prediction unless users gain a benefit from a correct estimate. As the release and processing times are unknown, the problem is often classified as *non-clairvoyant online scheduling*, see Motwani et al. [110].
Further, job j requires the concurrent and exclusive availability of $m_j \leq m$ nodes during the whole execution time. The number of required nodes m_j is specified by the users and does not change during the execution. Therefore, those jobs are classified as *rigid*, see Feitelson et al. [58]. As the network does not favor any subset of nodes, the execution time p_j is invariant of the subset of m_j nodes allocated to j. Each node is used by at most one job at any time (space-sharing). Note that a time-sharing of nodes within a massively parallel processor system is not used, as many jobs require a large amount of memory. In case of a time-sharing, this would result in memory swapping. Therefore, all jobs $j \in \tau$ with $p_j \leq \bar{p}_j$ run to completion, that is, we do not allow any preemption [106, 105]. The completion of job j in schedule S is denoted by $C_j(S)$. Contrary to other online scheduling models, see e.g. Albers [2], we do not require that machines must be allocated to a job at time r_j. Instead, the scheduler decides just before the start time $C_j(S) - p_j$ which machines are used to execute job j. In practice, this scheduling model is used in many parallel computers [82, 111], like, for instance, the IBM SP2.

4.3 Scheduling Policy and Objective

Local scheduling policies highly depend on the specific local environments. For example, some systems only require a high system utilization. Other installations might prefer a short response time for some user groups while delaying the jobs of all other users. The goal of a higher system utilization is implemented in all current systems. That is, machines are not left idle while some jobs are waiting for execution that fulfill all scheduling constraints to be immediately started. By this, all nowadays systems generate *nondelay* schedules as introduced in Section 2.2.
So far, the prioritization of certain users or user groups is implemented by using certain system partitions, see Feitelson and Jette [55], or special job queues, see Talby and Feitelson [154].

Within such systems users are only allowed to submit jobs to these job queues. Furthermore, the jobs from those job queues can only be started within pre-defined system partitions. Hence, it is possible to assign a larger partition to preferred user groups. However, as already stated, this normally results in a waste of available resources [55]. Another method to implement some kind of prioritization is the usage of quotas at real installations. In such environments, the amount of processing time for individual users can be limited. This ensures that the available machines can be used by groups with a higher priority. However, in some cases, the machines are left idle even if some jobs are waiting for execution as the corresponding quotas are reached [55]. Within this thesis, neither partitions nor quotas will be used. The whole scheduling system only consists of a single job queue in which all users submit their jobs.

The scheduling algorithms that are implemented in nowadays scheduling systems for parallel computers mainly try to minimize the makespan $C_{max} = \max\limits_{j \in \tau} C_j$ for a given job system τ. Additionally, the applied algorithms use job queues that lead to some kind of fairness that prevents job starvation. Therefore, existing scheduling systems for massively parallel processor systems can be classified as $P_m|\bar{r}_j, \bar{p}_j, m_j|C_{max}$. In Chapter 6, we will extend those systems to incorporate multiple objectives.

The development of new scheduling algorithms focuses more on the users of such machines. This is reasonable as the users tend to choose other parallel computers for their tasks if the local system does not calculate the computational jobs in an appropriate time. Therefore, the machine providers need to incorporate those objectives as a migration of local users reduces their profit. The list of possible evaluation objectives from the users' point of view is long. However, most of those objectives are somehow correlated.

For our evaluation, we have chosen the utilization of the parallel computer and the average weighted response time for all jobs $j \in \tau$ as the main objectives. Several other objectives like the slowdown or the average weighted wait time can easily be derived from them. Most of the evaluation objectives use some kind of weighting to specify the influence of different jobs on the result. Accounting to Schwiegelshohn et al. [139], a job weight should be used that reflects the resource consumption $RC_j = p_j \cdot m_j$ of the job j. This weight selection ensures that neither a splitting of jobs into several sequential jobs nor a combination of several independent jobs to a single job is beneficial regarding the objective function. Subsequently, those objectives are presented in detail.

The utilization (UTIL) of the parallel computer after having completed all jobs j from a given job system τ can be calculated as defined in Equation 4.1. The utilization is calculated beginning with the earliest job start time $(\min\limits_{j \in \tau}\{C_j(S) - p_j\})$ and ending with the latest completion time $(\max\limits_{j \in \tau} C_j(S))$. The utilization objective describes how effectively the available machines are used. Therefore, it reflects mainly the machine provider point of view.

$$\text{UTIL} = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j}{m \cdot \left(\max\limits_{j \in \tau} \{C_j(S)\} - \min\limits_{j \in \tau}\{C_j(S) - p_j\} \right)} \tag{4.1}$$

The second objective is the average weighted response time (AWRT) over all jobs of all users as defined in Equation 4.2. Note that the jobs are weighted by the previously defined resource consumption. A short AWRT describes that in average the users do not wait long for their jobs to complete. Therefore, this objective highly reflects the users' point of view.

$$\text{AWRT} = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum\limits_{j \in \tau} p_j \cdot m_j} \tag{4.2}$$

The average weighted wait time (AWWT), as defined in Equation 4.3, is also frequently used during the evaluation of scheduling systems. Here, the same weight is applied as for the average weighted response time. Both objectives are highly correlated as they only differ in the processing times of all jobs. However, some algorithms that will be presented in Section 4.4 require a certain amount of computation that is waiting for execution. Hence, the average weighted wait time is often used to prove that the resource consumption of all waiting jobs is sufficiently large.

$$\text{AWWT} = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - p_j - r_j)}{\sum\limits_{j \in \tau} p_j \cdot m_j} \tag{4.3}$$

In Chapter 6, we will extend the presented objective function definitions to emphasize user group dependent goals.

4.4 Scheduling Algorithms

As noted in Section 2.6.3 the scheduling algorithm is responsible for generating a valid schedule. The nowadays applied scheduling algorithms for massively parallel processor systems are rather simple. Most of them use a single job queue that holds all released jobs ordered by their submission time. Note that all jobs can be started directly after submission. Hence, the submission and the release times are identical. Whenever a new job is released or another job finishes execution, a scheduler tries to start jobs from the waiting queue on the currently idle nodes. The various scheduling algorithms for parallel computers mainly differ in the way the jobs are picked from the waiting queue. In the following, four different algorithms in increasing order of algorithmic complexity will be presented that are used as references in the remainder of this work. Note that the first three algorithms use a statically sorted waiting queue while the last algorithm dynamically reorders this queue.

4.4.1 First Come First Serve (FCFS)

FCFS starts the first job of the waiting queue whenever enough idle resources are available. Hence, this algorithm has a constant complexity as the scheduler always only tests whether the first job can be started immediately if a job in the schedule has completed its execution or a new job is on top of the waiting queue.

This algorithm ensures that no job is started after any later released job. Furthermore, this method does not require any estimations of job processing times. The application of First Come First Serve leads to a very low resource utilization in the worst case scenario, see, for example, Turek et al. [157]. However, for realistic workloads FCFS produces acceptable results, e.g. Ernemann et al. [47].

4.4.2 List Scheduling

List Scheduling as introduced by Graham [69] is not applied in this work. However, it serves as the basic concept for the two backfilling variants. By applying List Scheduling, the scheduler tries to find the first job within the waiting queue that can be started on the currently idle machines. Again, the algorithm uses the sorted queue. The complexity is higher than in the case of FCFS as in the worst case the whole queue is tested each time the scheduling procedure is initiated.

4.4.2.1 EASY Backfilling

EASY (**E**xtensible **A**rgone **S**cheduling S**y**stem) is similar to the original List Scheduling. However, if the first job within the waiting queue cannot be started immediately, the algorithm estimates the completion time of this job. To this end, a runtime estimation provided by the user is needed. Then, EASY tries to bring forward some jobs of the waiting queue by starting them on the currently idle machines while ensuring that the *first job* is not further delayed, see Lifka [99]. This algorithm requires more computational time than List Scheduling, as the scheduler needs to estimate the processing of the first job in case it cannot be started directly.

4.4.2.2 Conservative Backfilling (CONS)

Conservative Backfilling extends the concept of EASY. Here, the scheduler tries to find the next job within the waiting queue that can be started immediately while ensuring that *no previous job* within the queue is further delayed, see Mu'alem et al. [111]. This results in a much higher complexity of the scheduling algorithm as in the worst case, the completion time of all jobs within the waiting queue except of the last job must be estimated each time the scheduling process is initiated. Note that in both backfilling strategies the completion time of some jobs may increase compared to FCFS as in the online scenario some jobs have a shorter processing time than estimated by the users.

4.4.3 Greedy Scheduling (Greedy)

Greedy uses a dynamically sorted waiting queue contrary to the already introduced scheduling algorithms. To this end, the algorithm defines a complex sorting criterion. Each time the Greedy scheduling process is started, the queue is sorted according to this function. Then, a simple FCFS is applied. The complexity of this algorithm can be high as the calculation of the sorting criterion for each job within the waiting queue may be computationally expensive. Furthermore, the necessary sorting of all jobs has to be taken into account. Greedy provides the advantage to specify user or user group dependent preferences within the complex sorting criterion.

4.5 Evaluation

The evaluation of scheduling algorithms is important to identify the appropriate algorithms and the corresponding parameter settings. The results of theoretical worst-case analysis are often only of limited help as typical workloads on production machines do normally not exhibit the specific structure that will create a really bad case, see Section 2.3. In addition, theoretical analysis is often difficult to apply to scheduling strategies. Theoretical analysis of

random workloads will also not provide the desired information as the job parameter values are not distributed randomly, see e.g. Feitelson and Nitzberg [56]. Instead, the job parameters depend on several patterns and relations.
A trial and error approach on a commercial machine is tedious and significantly affects the system performance. Thus, it is usually not practicable to use a production machine for the evaluation except for the final testing. This just leaves simulation for all other cases. Simulations may either be based on real trace data or on a workload model. We will therefore discuss both possible approaches in detail.

4.5.1 Workload Models

Many existing approaches on workload modelling for massively parallel computers mainly focused on methods that use statistical distributions to fit the overall workload characteristics, see, for example, Lublin et al. [103] or Cirne et al. [21].
Generally, such statistical models use distributions or a collection of distributions to describe the important features of real workload attributes and the correlations among them. Then synthetic workloads are generated by sampling from the probability distributions, e.g. Jann et al. [84] or Feitelson [52]. Such statistical workload models have the advantage that new sets of job submissions can be generated easily. The consistence with real traces depends on the knowledge about the different examined parameters in the original workload. Many factors contribute to the actual process of workload generation on a real system. Some of them are known, some are hidden and hard to deduce. For example, it is difficult to find rules for job submissions by individual users. The analysis of workloads shows several correlations and patterns of the workload statistics. For example, jobs on many parallel computers require job sizes of a power of two, see Lo et al. [102], Feitelson [50], Lublin et al. [103], and Song et al. [148]. Other examples are the job distribution during the daily cycle obviously caused by the individual working hours of the users, or the job distribution of different week days, see Ernemann et al. [47]. Most approaches consider the different statistical moments isolated. However, it is very difficult to identify whether the important rules and patterns are extracted. In the same way it is difficult to tell whether the inclusion of the result is actually relevant to the evaluation and therefore also relevant for the design of an algorithm.
Besides the isolated modelling of each attribute, the correlations between different attributes were addressed as well. Lo et al. [102] demonstrated how the different degrees of correlation between job size and job runtime might lead to discrepant conclusions about the evaluation of scheduling performance. To consider such correlations, Jann et al. [84] divided the job sizes into subranges and then created a separate model for the inter-arrival time and the service time in each range, which may have a risk of over-fitting too many unknown parameters. Furthermore, Lublin et al. [103] considered the runtime attribute according to a two-stage hyper-gamma distribution with a linear relation between the job size and the parameters of the runtime distribution so that the longer runtime can be emphasized by using the distribution with the higher mean. Although these models can provide an overall description of a workload, they cannot give a deeper insight into the individual job submission behavior. In general, the global and summarized characterization does not provide a model for users or user groups and thus cannot give hints to relate the performance metrics to user groups.
In order to integrate additional characteristics into the workload models, some research projects focused on the sequential dependencies between jobs. For example, Feitelson et al. [50] showed that users tend to submit jobs which are similar to their predecessors. Therefore,

more realistic models are needed which incorporate the correlation within the sequence of job submissions. Hence, Song et al. [147] have analyzed job dependencies in more detail. The resulting workload model incorporates temporal dependencies and the correlation between job parameters. To this end, individual Markov chains have been created for runtime and node requirements of jobs. The correlation between job parameters requires the combination of such individual Markov chains. To this end, a novel approach of transforming the states in the different Markov chains has been proposed. With this model the sequential dependencies of jobs regarding the node requirements and the actual runtimes can be described. However, the model neither includes the release times of individual jobs nor a relation between jobs and individual users or user groups.
Consequently, Song et al. [148] generated a workload model that is based on individual user groups. The particular generation of a workload model is based on a specific existing workload from a real system. To this end, jobs are first partitioned into clusters such that the jobs with similar characteristics are placed in the same cluster. Then the users are grouped by the contribution of their submission into these job clusters. The output of this user group model has been statistically compared with original and not-used reference workloads.
Although such complex workload models were generated, most of the evaluations of scheduling algorithms nowadays use workload traces from real existing parallel computers. This results from the limited complexity of existing models. A precise evaluation requires the availability of a model that incorporates all relevant characteristics and dependencies of the jobs like the release date, the estimated processing time, the real processing time, and the node requirements. So far, none of the available workload models can fulfill all of these requirements.

4.5.2 Workload Traces

For the above mentioned reasons it is difficult to use statistical workload models for our research work. Therefore, real workload traces have been used. The standard parallel workload archive, see Feitelson [54] and Table 4.1, is a source for job traces of massively parallel processor systems. In comparison to statistical workload models, the use of actual workload traces is simpler as they inherently include all submission patterns and underlying mechanisms. The traces exactly reflect the real workload. However, as the data basis is limited it is difficult to perform many simulations.
Furthermore, the total number of available processors differs in those workloads. Hence, the freedom of selecting different configurations and scheduling strategies is limited as a specific job submission depends on the original circumstances. Each trace is only valid on a similar machine configuration and the same scheduling strategy. For instance, trace data taken from a 128 processor parallel machine will lead to unrealistic results on a 256 processor machine. Therefore, the selection of the underlying data for the simulations depends on the circumstances determined by the MPP architecture as well as the scheduling strategy.
Some of the workload traces taken from the archive [54] do not provide estimates for the job processing time, see Table 4.1. As most of the scheduling algorithms, see Section 4.4, require such information, Song et al. [146] developed a method to recover the estimated job processing time. This method first groups all jobs from a workload regarding their processing time. Then, for each group a Beta distribution is used to describe the dependencies between real and estimated processing time. The models are extracted from several real traces and tested on other traces that also include both parameters. The results clearly demonstrated the usability of this model.

Identifier	NASA	CTC	KTH	LANL	SDSC00	SDSC95	SDSC96
Machine	iPSC/860	SP2	SP2	CM-5	SP2	SP2	SP2
from to	10/01/93 12/31/93	06/26/96 05/31/97	09/23/96 08/29/97	04/10/94 09/24/96	04/28/98 04/30/00	12/29/94 12/30/95	12/27/95 12/31/96
Processors	128	430	100	1024	128	416	416
Jobs	42264	79302	28490	201387	67667	76872	38719
Estimated runtime	no	yes	yes	partially	yes	no	no
Users	69	679	214	213	428	97	59

Table 4.1: Used Workloads from the Standard Parallel Workload Archive, see Feitelson [54].

This model has been applied to all workloads that did not provide the corresponding estimated processing times. All of the different workload traces presented in Table 4.1 have already been analyzed in detail in other publications, see the given references below. Furthermore, the development of scheduling algorithms is often based on at least some of them. All workload traces are available in the Standard Workload Format defined by Feitelson [54]. Within this format each job is described by 18 parameters like the release date, the estimated processing time (if available), the real processing time, the node requirements, the memory consumption and the user and user group identification numbers. Hence, the different scheduling systems can include many parameters for their decision process.
The NASA workload comes from a 128-node iPSC/860 hypercube system. This workload was first analyzed in detail by Feitelson et al. [56]. The CTC trace origins from the Cornell Theory Center with an IBM/SP2 parallel computer. Hotovy examined this trace in more detail [82]. The workload trace from the Los Alamos National Lab (LANL) is based on a 1024 nodes Connection Machine CM-5. Feitelson [51] describes the characteristics in detail. This parallel computer consists of the largest number of nodes of all available traces (1024) and hence will be used as a reference later. The recorded trace from the Swedish Royal Institute of Technology (KTH) is based on a 100 node IBM/SP2. This trace was examined in detail by Mu'alem et al. [111]. The two workload traces (SDSC95, SDSC96) both origin from a 416 node Intel Paragon at the San Diego Supercomputer Center. Both traces are recorded at subsequent years and hence give the opportunity to analyze changes within the user behavior. The traces were, for example, examined by Windisch et al. [164]. The SDSC00 workload trace was also recorded at the San Diego Supercomuting Center. However, this time an IBM/SP2 with 128 nodes was used. A deeper analysis is provided by Feitelson [53].

Part II

Optimizing the Scheduling of a Parallel Machine

Chapter 5

Scaling of Workload Traces

The evaluation and optimization of the scheduling strategy for massively parallel processor systems is based on simulations as described in Section 4.5. Those simulations help to evaluate the possible scheduling algorithms and are used to identify the corresponding parameter settings. The evaluation of scheduling strategies can be based on workload models or real workload traces. As discussed in Section 4.5.1, workload models can be used to generate any kind of workload for various machine configurations. However, none of the models is able to express the whole complexity of a real workload. The real workload traces presented in Section 4.5.2 restrict the freedom of selecting different configurations and scheduling strategies as a specific job submission depends on the original circumstances. A trace is only valid on a similar machine configuration and the same scheduling strategy.

Our research on job scheduling strategies for parallel computers as well as for Computational Grid environments leads to the requirement of considering different resource configurations. Furthermore, the development of strategies that can be applied to various workload traces with similar results requires the existence of several workload traces for a single machine configuration. The development of a robust scheduling algorithm is based on evaluations by using several of such workload traces. However, the number of available traces is limited, see Section 4.5.2. Most of those workloads are observed on different supercomputers. Mainly, the total number of available processors differs in those workloads, see Table 4.1. Therefore, our goal was to find a reasonable method to scale such workload traces to fit to a standard supercomputer. However, special care must be taken to keep the new workload as consistently as possible to the original trace. To this end, criteria for measuring the validity must be chosen for the examined methods to scale the workload.

In general, the applicability of workload traces to other machine configurations with a different number of processors is complicated. For instance, this could result in a high workload and an unrealistically long wait time for a job. Contrary, the machine is not fully utilized if the amount of computational work is too low. However, it is difficult to change any parameter of the original workload trace as it has an influence on its overall validity. For example, the reduction of the inter-arrival time destroys the distribution of the daily cycle. Therefore, modifications of the job length are also inappropriate. Modifications of the requested processor number of a job change the original job size distribution. For instance, we might neglect an existing preference of jobs with a power of 2 processor requirement. In the same way, an alternative scaling of the number of requested processors by a job would lead to an unrealistic job size submission pattern. For example, scaling a trace taken from a 128 node massively parallel processor system to a 256 node system by just duplicating each job preserves the temporal distribution of job submissions. However, this transformation also leads to an unrealistic distribution as no jobs with a higher processor demand are submitted.

Note that the scaling of a workload to match a different machine configuration always alters the original distribution whatsoever. Therefore, as a trade-off special care must be taken to preserve the original time correlations and job size distribution. This chapter presents such a scaling strategy.

In the following, we present the examined methods to scale the workload traces, see also Ernemann et al. [47]. To this end, we briefly discuss the different methods as the results of each step motivate the next.

First, it is necessary to select quality criteria for comparing workload modifications. Distribution functions could be used to compare the similarity of the modified with the corresponding original workloads. This method might be valid, but, it is unknown whether the new workload has a similar effect on the resulting schedule as the original workload. Here, the same arguments as in Section 4.5.1 can be used, as the simple similarity of several independent distributions does not necessarily correspond with a similar scheduling behavior.

As mentioned above, the scheduling strategy that has been used on the original parallel machine also influences the submission behavior of the users. If a different scheduling system is applied and causes different response times, the submission pattern of later arriving jobs will most certainly be influenced. This is a general problem, see Feitelson and Downey [34] or Downey [33], that has to be kept in mind if workload traces or statistical models are used to evaluate new scheduling systems. This problem can be solved if the feedback mechanisms of prior scheduling results on new job submissions are known. However, such a feedback is very difficult to extract as the underlying mechanisms vary between individual users and single jobs. Hence, we neglect those feedback influences within our scaling process.

For our evaluation, we have mainly chosen the Average Weighted Response Time (AWRT) and the Average Weighted Wait Time (AWWT), which are already defined in Equations 4.2 and 4.3 on page 46. Several other scheduling objectives, for instance the slowdown (see e.g. Ernemann et al. [44]), can be derived from AWRT and AWWT. Furthermore, the utilization (UTIL) as defined in Equation 4.1 was used. To match the original scheduling systems, we used First Come First Serve and EASY Backfilling (see Sections 4.4.1 and 4.4.2.1) for generating the corresponding schedules. These scheduling strategies are well known and used for most of the original workloads. Note that the focus of this paper is not to compare the quality of both scheduling strategies. Instead, we use the results of each algorithm to compare the similarity of each modified workload with the corresponding original workload.

In addition, the makespan (see Section 2.1 on page 7) is considered, which is the completion time of the last job within the workload. The *overall Resource Consumption* (*RC*) is given as a measurement for the amount of consumed processing power for all jobs within the individual workloads by summing the resource consumption of all jobs, see Equation 5.1.

$$RC = \sum_{j \in \tau} p_j \cdot m_j \tag{5.1}$$

Note that in the following we refer to jobs with a higher number of requested processors as *bigger* jobs, while calling jobs with a smaller processor demand *smaller* jobs.

Scaling only the number of requested processors of a job results in the problem that the whole workload distribution is transformed by a factor. In this case the modified workload might not contain jobs requesting 1 processor or a small number of processors. In addition, the favor of jobs requesting a power of 2 processors is not modelled correctly for most scaling factors. Alternatively, the number of jobs can be scaled. Each original job is duplicated to several jobs in the new workload. Using only this approach has the disadvantage that the ratio of the number of requested processors to the machine size of each job within the new workload is smaller compared to the ratio of the original workload. For instance, if the biggest job in the original workload uses the whole machine, a duplication of each job for a machine with twice the number of processors leads to a new workload in which no job requests the maximum number of processors at all.

5.1 Precise Scaling of Job Size

Workload	NASA	CTC	KTH	LANL	SDSC00	SDSC95	SDSC96
Number of jobs (n)	42264	79302	28490	201387	67667	76872	38719
Number of nodes (m)	128	430	100	1024	128	416	416
Size of the biggest job	128	336	100	1024	128	400	320
Static factor f	3	8	10	1	8	3	3

Table 5.1: Details of the Examined Workload Traces.

Based on the considerations above, a factor f is determined for combining the scaling of the requested processor number of each job with the scaling of the total number of jobs. In Table 5.1 the requested maximum number of processors of a single job is given as well as the total number of available processors.
As explained above multiplying solely the number of processors of a job or the number of jobs by a constant factor is not reasonable. Therefore, a combination of both strategies has been applied in the following. In order to analyze the individual influence of both possibilities the workloads were modified by using a probabilistic approach: a probability factor p is used to specify whether the requested number of processors is multiplied for a job or whether with probability $(1-p)$ copies of this job are created. So, during the scaling process each job of the original workload is modified by only one of the given alternatives. A rectangular distribution between 0 and 100 is sampled to generate p. A decision value d is used to discriminate which alternative is applied for a job. If p produced by the probabilistic generator is greater than d, the number of processors is scaled by f for the job. Otherwise, f identical, new jobs are included in the new workload. So, if d is a high value, the system prefers the creation of smaller jobs while resulting in fewer bigger jobs.
As a first approach, integer scaling factors have been chosen based on the relation to a 1024 processor machine. We restricted ourselves to integer factors as it would require additional considerations to model fractional job parts. For the KTH a factor f of 10 is chosen, for the NASA and the SDSC00 workloads we used a factor of 8, and for all other workloads a factor of 3 is applied. Note that for the SDSC95 workload one job yields more than 1024 processors if multiplied by 3. Therefore, this single job is reduced to 1024.

For the examination of the influence of d, we created 99 modified workloads for each original workload with d varying between 1 and 99. However, with exception of the NASA trace, our method did not produce satisfying results for the workload scaling. The imprecise factors increased the relative overall amount of workload up to 26 % which lead to a jump of several factors for AWRT and AWWT. This shows how important the precise scaling of the overall amount of workload is. Second, if the chosen factor f is bigger than the precise scaling factor the generated workloads that in the majority include smaller jobs (d is high) scale better than the generated workloads with bigger jobs. If f is smaller than or equal to the precise scaling factor, the modified workloads scale better for smaller values of d.
Based on these results, we introduced a precise scaling for the number of processors m_j of all jobs $j \in \tau$ (in the following also called job size). As the precise scaling factors for the workloads CTC (2.38), KTH (10.24), SDSC95 (2.46) and SDSC96 (2.46) are not integer values, an extension to the previous method was necessary. In case a single large job being created the number of processors is multiplied by the precise scaling factor f and rounded, if necessary.
The scheduling results for the modified workloads are presented in Table 5.2. Only the results for the original workload (ref) and the modified workloads with the parameter settings of $d = \{1, 50, 99\}$ are shown. The modified CTC based workloads are close to the original workload in terms of AWWT, AWRT and UTIL if only bigger jobs are created ($d = 1$). For increasing values of d, also the AWRT, AWWT and UTIL increase. Overall, the results are closer to the original results in comparison to using an integer factor. A similar behavior can be observed for the SDSC95 and SDSC96 workload modifications. For KTH the results are similar with the exception that AWRT and AWWT increase for decreasing values of d.
The results for all modified NASA workloads are very similar. The AWRT and AWWT for the derived and original workloads does not vary much independently from the used settings. Note that the NASA workload itself is quite different in comparison to the other workloads as it includes a high percentage of interactive jobs. Furthermore, nearly no job ever waits for execution.
In general, the results for this method are still not satisfying. Using a factor of $d = 1$ is not realistic as mentioned at the beginning of this chapter.

5.2 Precise Scaling of Number and Size of Jobs

Consequently, the precise factor is also used for the duplication of jobs. However, as mentioned above, it is not trivial to create fractions of jobs. To this end, a second random variable p_1 has been introduced with values between 0 and 100. The variable p_1 is used to decide whether the lower or upper integer bound of the precise scaling factor is considered. For instance, the precise scaling factor for the CTC workload is 2.3814. We used the value of p_1 to decide whether to use the scaling factor of 2 or 3. If p_1 is smaller than 38.14 the factor of 2 will be applied, 3 otherwise. The average results in a scaling factor of around 2.3814. For the other workloads we used the same scaling strategy with the decision values of 24.00 for the KTH workload and with 46.15 for the SDSC95 and SDSC96 workloads.
This enhanced method improves the results significantly. In Table 5.3 the main results are summarized. Except for the simulations with the SDSC00 workload, all results show a clear improvement in terms of similar utilization for each of the according workloads. The results for the CTC workload again show that only small values of d lead to a convergence of the

AWRT and AWWT to the original workload. The same qualitative behavior can be observed for the workloads which are derived from the KTH and SDSC00 workloads.
The results of the modifications for the SDSC95 and SDSC96 derived workloads are already acceptable as the AWRT and AWWT between the original workloads and the modified workloads with a mixture of smaller and bigger jobs ($d = 50$) are already very close. For these two workloads the scaling is acceptable.
In general, it can be summarized that the modifications still do not produce matching results for all original workloads. Although we use precise factors for scaling the number of jobs and for the number of requested processors, some of the scaled workloads yield better results than the original workload. This is probably caused due to the fact that according to the factor d the scaled workload is distributed over either more but smaller ($d = 99$) or fewer but bigger jobs ($d = 1$). As mentioned before, the existence of more smaller jobs in a workload usually improves the scheduling result. The results show that a larger machine leads to smaller AWRT and AWWT values. Or contrary, a larger machine can execute relatively more workload than an according number of smaller machines for the same AWRT or AWWT. However, this applies only for the described workload modifications. Here, we generate relatively more smaller jobs in relation to the original workload.

workload	m	d	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
CTC	430	ref	EASY	79285	29306750	66	13905	53442	8335013015
	1024	1		82509	29306750	66	13851	53377	19798151305
		50		158681	29306750	75	21567	61117	22259040765
		99		236269	29306750	83	30555	70083	24960709755
	430	ref	FCFS	79285	29306750	66	19460	58996	8335013015
	1024	1		82509	29306750	66	19579	59105	19798151305
		50		158681	29306750	75	28116	67666	22259040765
		99		236269	29306750	83	35724	75253	24960709755
KTH	100	ref	EASY	28482	29363625	69	24677	75805	2024854282
	1024	1		30984	29363625	69	25002	76102	20698771517
		50		157614	29363625	68	17786	68877	20485558974
		99		282228	29363625	67	10820	61948	20258322777
	100	ref	FCFS	28482	29381343	69	400649	451777	2024854282
	1024	1		30984	29373429	69	386539	437640	20698771517
		50		157614	29376374	68	38411	89503	20485558974
		99		282228	29363625	67	11645	62773	20258322777
NASA	128	ref	EASY	42049	7945421	47	6	9482	474928903
	1024	1		44926	7945421	47	6	9482	3799431224
		50		190022	7945421	47	5	9481	3799431224
		99		333571	7945421	47	1	9477	3799431224
	128	ref	FCFS	42049	7945421	47	6	9482	474928903
	1024	1		44926	7945421	47	6	9482	3799431224
		50		190022	7945421	47	5	9481	3799431224
		99		333571	7945421	47	1	9477	3799431224

Continued on next page

workload	m	d	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
SDSC00	128	ref	EASY	67655	63192267	83	76059	116516	6749918264
	1024	1		72492	63201878	83	74241	114698	53999346112
		50		305879	63189633	83	54728	95185	53999346112
		99		536403	63189633	83	35683	76140	53999346112
	128	ref	FCFS	67655	68623991	77	2182091	2222548	6749918264
	1024	1		72492	68569657	77	2165698	2206155	53999346112
		50		305879	64177724	82	516788	557245	53999346112
		99		536403	63189633	83	38787	79244	53999346112
SDSC95	416	ref	EASY	75730	31662080	63	13723	46907	8284847126
	1024	1		77266	31662080	63	14505	47685	20439580820
		50		151384	31662080	70	19454	52652	22595059348
		99		225684	31662080	77	25183	58367	24805524723
	416	ref	FCFS	75730	31662080	63	17474	50658	8284847126
	1024	1		77266	31662080	63	18735	51914	20439580820
		50		151384	31662080	70	24159	57357	22595059348
		99		225684	31662080	77	28474	61659	24805524723
SDSC96	416	ref	EASY	37910	31842431	62	9134	48732	8163457982
	1024	1		38678	31842431	62	9503	49070	20140010107
		50		75562	31842431	68	14858	54305	22307362421
		99		112200	31842431	75	22966	62540	24410540372
	416	ref	FCFS	37910	31842431	62	10594	50192	8163457982
	1024	1		38678	31842431	62	11175	50741	20140010107
		50		75562	31842431	68	18448	57896	22307362421
		99		112200	31842431	75	26058	65632	24410540372

Table 5.2: Results for Precise Scaling for the Job Size and Estimated Scaling for Job Number.

workload	m	d	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
CTC	430	ref	EASY	79285	29306750	66	13905	53442	8335013015
	1024	1		80407	29306750	66	13695	53250	19679217185
		50		133981	29306750	66	12422	51890	19734862061
		99		187605	29306750	66	10527	50033	19930294802
	430	ref	FCFS	79285	29306750	66	19460	58996	8335013015
	1024	1		80407	29306750	66	18706	58261	19679217185
		50		133981	29306750	66	15256	54724	19734862061
		99		187605	29306750	66	12014	51519	19930294802
KTH	100	ref	EASY	28482	29363625	69	24677	75805	2024854282
	1024	1		31160	29363625	69	24457	75562	20702184590
		50		159096	29363625	69	18868	70002	20725223128
		99		289030	29363625	69	11903	62981	20737513457

Continued on next page

workload	m	d	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
KTH	100	ref	FCFS	28482	29381343	69	400649	451777	2024854282
	1024	1		31160	29381343	69	383217	434322	20702184590
		50		159096	29371792	69	41962	93097	20725223128
		99		289030	29363625	69	12935	64013	20737513457
NASA	128	ref	EASY	42049	7945421	47	6	9482	474928903
	1024	1		44870	7945421	47	6	9482	3799431224
		50		188706	7945421	47	2	9478	3799431224
		99		333774	7945421	47	1	9477	3799431224
	128	ref	FCFS	42049	7945421	47	6	9482	474928903
	1024	1		44870	7945421	47	6	9482	3799431224
		50		188706	7945421	47	3	9479	3799431224
		99		333774	7945421	47	1	9477	3799431224
SDSC00	128	ref	EASY	67655	63192267	83	76059	116516	6749918264
	1024	1		77462	63192267	83	75056	115513	53999346112
		50		305802	63189633	83	61472	101929	53999346112
		99		536564	63189633	83	35881	76338	53999346112
	128	ref	FCFS	67655	68623991	77	2182091	2222548	6749918264
	1024	1		77462	68486537	77	2141633	2182090	53999346112
		50		305802	64341025	82	585902	626359	53999346112
		99		536564	63189633	83	38729	79186	53999346112
SDSC95	416	ref	EASY	75730	31662080	63	13723	46907	8284847126
	1024	1		76850	31662080	63	14453	47641	20411681280
		50		131013	31662080	63	13215	46319	20466656625
		99		185126	31662080	62	11635	44739	20446439351
	416	ref	FCFS	75730	31662080	63	17474	50658	8284847126
	1024	1		76850	31662080	63	18511	51698	20411681280
		50		131013	31662080	63	15580	48684	20466656625
		99		185126	31662080	62	12764	45867	20446439351
SDSC96	416	ref	EASY	37910	31842431	62	9134	48732	8163457982
	1024	1		38459	31842431	62	9504	49084	20100153862
		50		66059	31842431	62	9214	49087	20106192767
		99		92750	31842431	62	8040	47796	20171317735
	416	ref	FCFS	37910	31842431	62	10594	50192	8163457982
	1024	1		38459	31842431	62	11079	50658	20100153862
		50		65627	31842431	62	10126	49823	20106192767
		99		92750	31842431	62	8604	48360	20171317735

Table 5.3: Results using Precise Factors for Job Number and Size.

workload	m	f	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
CTC	430	ref	EASY	79285	29306750	66	13905	53442	8335013015
	1024	2.45		136922	29306750	68	14480	54036	20322861231
	430	ref	FCFS	79285	29306750	66	19460	58996	8335013015
	1024	2.45		136922	29306750	68	19503	59058	20322861231

Continued on next page

workload	m	f	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
KTH	100	ref	EASY	28482	29363625	69	24677	75805	2024854282
	1024	10.71		165396	29363625	72	24672	75826	21708443586
	100	ref	FCFS	28482	29381343	69	400649	451777	2024854282
	1024	10.71		165396	29379434	72	167185	218339	21708443586
NASA	128	ref	EASY	42049	7945421	47	6	9482	474928903
	1024	8.00		188706	7945421	47	2	9478	3799431224
	128	ref	FCFS	42049	7945421	47	6	9482	474928903
	1024	8.00		188258	7945421	47	4	9480	3799431224
SDSC00	128	ref	EASY	67655	63192267	83	76059	116516	6749918264
	1024	8.21		312219	63204664	86	75787	116408	55369411171
	128	ref	FCFS	67655	68623991	77	2182091	2222548	6749918264
	1024	8.58		323903	69074629	82	2180614	2221139	58020939264
SDSC95	416	ref	EASY	75730	31662080	63	13723	46907	8284847126
	1024	2.48		131884	31662080	63	13840	46985	20534988559
	416	ref	FCFS	75730	31662080	63	17474	50658	8284847126
	1024	2.48		131884	31662080	63	17327	50472	20534988559
SDSC96	416	ref	EASY	37910	31842431	62	9134	48732	8163457982
	1024	2.48		66007	31842431	62	8799	48357	20184805564
	416	ref	FCFS	37910	31842431	62	10594	50192	8163457982
	1024	2.48		66007	31842431	62	10008	49566	20184805564

Table 5.4: Results for Increased Scaling Factors with $d = 50$.

5.3 Adjusting the Scaling Factor

In order to compensate the above mentioned scheduling advantage of having more jobs with the requirements of only a few processors in relation to the original workload, the scaling factor f has been modified to increase the overall amount of workload. The goal is to find a scaling factor f so that the results in terms of the AWRT and AWWT match the original workload for $d = 50$. In this way, a combination of bigger as well as more smaller jobs exists. To this end, additional simulations have been performed with small increments of f.

In Table 5.4, the corresponding results are summarized. More extended results are shown in Table A.1 in the appendix on page 182. It can be observed that the scheduling behavior does not strictly linear correspond to the incremented scaling factor f. The precise scaling factor for the CTC workload is 2.3814, whereas a slightly higher scaling factor corresponds with an AWRT and AWWT close to the original workload simulation results. The actual values slightly differ e.g. for the EASY Backfilling ($f = 2.43$) and the FCFS strategy ($f = 2.45$). Note that the makespan stays constant for different scaling factors. Obviously the makespan is dominated by a later job and is therefore independent of the increasing amount of computational tasks (resource consumption, utilization and the number of jobs). This underlines that the makespan is predominantly an off-line scheduling objective [57]. Analogous results were produced by using the KTH, SDSC95 and SDSC96 workloads.

The increment of the scaling factor f for the NASA workloads leads to different effects. A marginal increase causes a significant change of the scheduling behavior. The values of the

AWRT and AWWT are drastically increasing. However, the makespan, the utilization and the workload stay almost constant. This proves that the original NASA workload has almost no wait time. A new job is started whenever the previous job is finished.
The approximation of an appropriate scaling factor for the SDSC00 workload differs from the previously described process as the results for the EASY and FCFS strategies differ much. Here, the AWRT and the AWWT of the FCFS are more than a magnitude higher than by using EASY Backfilling. Obviously, the SDSC00 workload contains highly parallel jobs as this causes FCFS to suffer in comparison to EASY backfilling. In our opinion, it is more reasonable to use the results of the EASY strategy for the workload scaling, because the EASY strategy is more representative for many current systems and for the observed workloads. However, as discussed above, if the presented scaling methods are applied to other traces, it is necessary to use the original scheduling method that caused the workload trace.
The described method has been applied to all presented workload traces. The modified traces that will be used in the remainder of this thesis are presented in Table 5.5.

Identifier	NASA	CTC	KTH	LANL	SDSC00	SDSC95	SDSC96
Processors	1024	1024	1024	1024	1024	1024	1024
Jobs	188986	136471	167375	201378	310745	131762	66185

Table 5.5: Modified workload traces (only the modified values).

Chapter 6

Developing a Multi-Objective Scheduling System

Chapter 1 provides already a good motivation for using multi-objective scheduling systems for many real life problems. Further, also scheduling systems for parallel processing systems need the ability to optimize more than one objective as the different users have different business relationships to the machine providers and hence must be satisfied differently. However, in Chapter 4, we have shown that nowadays used scheduling systems in such environments only optimize a single objective. Nevertheless, Chapter 2 and especially Section 2.5.5 provide some examples of scheduling systems that incorporate more than one objective. This chapter describes the main concept of our approach to extend the existing scheduling systems for parallel processing so that they can also be applied to scenarios with multiple objective.
Hence, the main goal of this work is to develop a methodology to automatically generate such multi-objective online scheduling strategies. Further, we aim to generate robust scheduling strategies in the sense that the same strategy can be applied to several different workload traces with a similar scheduling outcome.
The application of a multi-objective scheduling system requires a provider defined objective function. However, so far the different machine providers are often not able to express their preferences in detail. Thus, a method must be provided for them to define more complex objective functions that are based on several simple objectives.
To this end, we will follow the concept of Krallmann et al. [94] presented in Section 2.6.2 in order to derive such a scheduling objective. However, this concept will be modified to emphasize the generation of robust scheduling systems.
This chapter presents the main ideas of our approach to derive appropriate scheduling objective functions from the machine providers' point of view as well as the development of the corresponding scheduling strategies. Based on these general ideas, the following chapters contain the details of the different development steps.

6.1 Solution Details

Here, we will not repeat the general problem structure of the problem, as this has already been done within the introduction and Figure 1.1. Corresponding to the general problem, we first describe our simple evaluation objectives. Note that the methodology is not limited to the presented objectives. Subsequently, we introduce the main development steps of our scheduling strategy development approach in detail.

6.1.1 Our Scheduling Objectives

Our model is able to pay more attention to certain users or user groups in order to achieve a higher degree of satisfaction for them. Unfortunately, the available workload data do not provide any user group information nor define any complex scheduling objective. To address the user group problem, we are using the work of Song et al. [148], who have shown that users can be reasonably well partitioned into 5 groups for all available MPP workload traces. Those groups are differentiated with respect to job characteristics and frequency of job submissions. Within this work, we will also use 5 different user groups. However, we will use the user's resource consumption as the differentiation criterion. Here, we generate the 5 user groups such that user group 1 represents all users with a higher resource consumption whereas all users in group 5 have a very low resource demand. Equation 6.1 defines a function $\varpi_u(j)$ that is used to describe the relationship between job j and user u.

$$\varpi_u(j) = \begin{cases} 1, & \text{if job } j \text{ belongs to user } u \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

Table 6.1 consists of the ratio between the total resource consumption of user u:

$$RC_u = \sum_{j \in \tau} RC_j \cdot \varpi_u(j) \tag{6.2}$$

and the overall resource consumption (RC), see Equation 5.1, which is needed for a single user u to become a member of the specified user group.

User Group	1	2	3	4	5
RC_u/RC	> 8 %	> 2 %	> 1 %	> 0.1 %	≤ 0.1 %

Table 6.1: The assignment of users to user groups depending on the corresponding resource consumption ratio.

This leads to Equation 6.3, which defines a function $\varrho_i(j)$ that is used to introduce different objectives for different user groups.

$$\varrho_i(j) = \begin{cases} 1, & \text{if job } j \text{ belongs to user group } i \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

Based on this user group definitions we are able to define all of our scheduling objectives. In our work, we exemplarily use seven objectives. To our knowledge, no other research on scheduling algorithms has used more than three objectives, see Section 2.5.5.
All of the seven individual objectives to evaluate the achieved scheduling results can only be calculated after all jobs from the given workload have finished their processing.
The first objective is the overall system utilization (UTIL) that is specified in Equation 4.1 in Section 4.3. The average weighted response time (AWRT) as defined in Equation 4.2 is our second objective. Note that the weight of a job is its total resource consumption, see

Schwiegelshohn et al. [139]. This assumes that there is no preference of jobs due to their resource consumption. If desired, the weight can be adjusted to implement preferences. The last five scheduling objectives are the average weighted response times (AWRTi) for user groups $i \in \{1, 2, \ldots, 5\}$, see Equation 6.4. The definition is similar to Equation 4.2 and uses the same weight.

$$\mathrm{AWRT}i = \frac{\sum\limits_{j \in \tau} m_j \cdot p_j \cdot (C_j(S) - r_j) \cdot \varrho_i(j)}{\sum\limits_{j \in \tau} m_j \cdot p_j \cdot \varrho_i(j)} \tag{6.4}$$

These objectives are used as in real installations of parallel computers different user groups have different relationships to the machine providers. In our work, the complex objective function can be described by combining the presented seven simple objectives.

6.1.2 Used Workload Traces

During the development as well as during the evaluation of our scheduling strategies we use simulations with the scaled workload traces, see Chapter 5, that are extended by the introduced user group information, see Table 6.1. Further, we will not use preemption. Hence, the job characteristics do not vary as jobs cannot be split. The main advantage of non-preemptive schedules is the smaller amount of schedules that can be generated compared with preemptive schedules. For the same job set a preemptive scheduling system can produce many more different schedules. Hence, the limitation to non-preemptive schedules within this work reduces the number of necessary simulations. Note that the usage of non-preemptive schedules does not result in a conceptual limitation as the same methods can be applied to preemptive scheduling systems. Furthermore, preemptive schedules are not frequently used in practice.

6.2 Description of the Main Steps during the Scheduling Strategy Development

In the following, we describe the main steps to automatically develop multi-objective scheduling strategies for massively parallel processing systems. Then, each of those steps is detailed in the next chapters.

6.2.1 Generating the Pareto Fronts for all Workloads

As already introduced, the machine providers need to define appropriate scheduling objectives. To this end, they need to know which solutions are achievable (Pareto front) and where the solutions are located that can be produced by applying existing standard scheduling strategies. Using this information the providers are able to run the existing scheduling systems in the case that the outcome fits to the local demand. Otherwise, the providers can define appropriate objectives that reflect their preferences, and an appropriate scheduling strategy needs to be developed.

As already mentioned, a goal of our work is the development of robust scheduling strategies. Hence, the scheduling strategy development should be based on all available workloads. Con-

sequently, the Pareto fronts must be generated for all those workloads that are combined to a representative front, see Section 6.2.2.
As presented in Section 3.2, multi-objective Evolutionary Algorithms are appropriate for the task to generate Pareto fronts. Note that these algorithms do not guarantee to produce the true Pareto front. Nevertheless, they have proven to approximate the Pareto front with a high quality. The Pareto front of possible schedules is generated in an offline fashion. Hence, information about future jobs can already be used during the scheduling process. However, we assume that the scheduling constraints are observed as, for example, no job can be started before its release date. Consequently, the Pareto front describes all schedules that can be generated by speculation on future job submission. Hence, some of those scheduling solutions cannot be reached in the online scenario where typically nondelay schedules are generated. Figure 6.1 shows the outcome of this step exemplarily for a single workload and two objectives. Note that we assume a minimization of the objectives. The details of our Pareto front generation procedure are presented in Chapter 7.

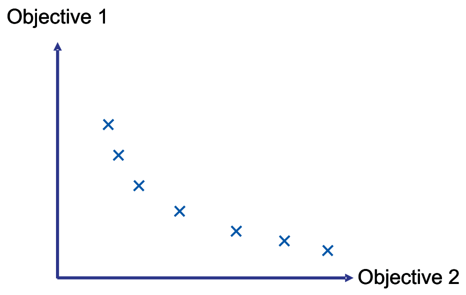
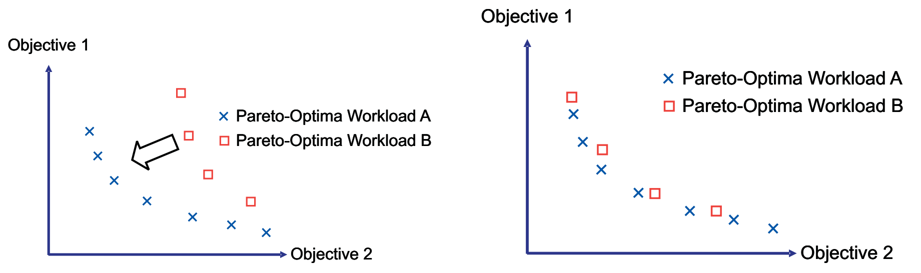


Figure 6.1: A Pareto front of a single workload trace.

6.2.2 Extraction of the Multi-Objective Evaluation Function

The generation of an objective function that expresses the provider's preferences and leads to a robust scheduling strategy can be based on the generated Pareto fronts of the different workloads. The provider could select certain regions within each individual front and try to generate a scheduling strategy that produces such schedules for the workloads. However, this procedure might result in a strategy that fits to a single workload but cannot necessarily be used for other traces.
In this work, we use two approaches to derive a robust scheduling strategy. In the first approach, the hyper planes that describe the Pareto fronts of the different workload traces are scaled and moved to generate a maximum overlapping. That is, the distance between the fronts must be minimized. The resulting representative front enables the machine provider to select certain regions of preferences that are valid for all underlying workloads. Then, the distances of any possible schedule to these preferred regions define the objective function that must be minimized. We assume that this objective function definition leads to scheduling strategies that work well for most of the workloads as the representative Pareto front is based on all traces. Figure 6.2a shows two exemplary Pareto fronts of two workloads that must be transformed to the representative Pareto front displayed in Figure 6.2b. Figure 6.3 displays the selection of preferred regions within the resulting representative Pareto front. Note that we assume convex priority regions that are ranked according to the providers preferences. Region 1 represents the best schedules.



(a) Original Pareto fronts. (b) Resulting representative Pareto front after scaling and moving.

Figure 6.2: Pareto fronts of two workload traces.

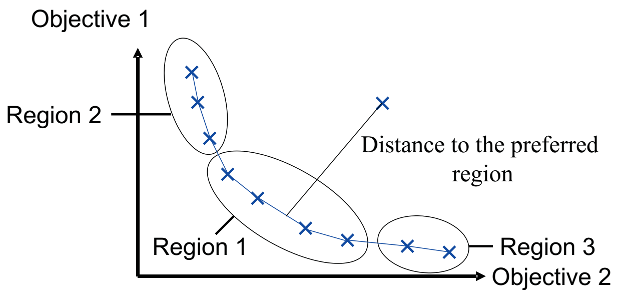


Figure 6.3: Definition of preference regions with the resulting distance measurement for an arbitrary schedule.

The second approach chooses one of the generated Pareto fronts as a reference. Based on this Pareto front and the corresponding provider's preferences a scheduling strategy can be developed. If the generated strategy applied to all other workloads produces similar schedules compared to the reference Pareto front and the results for the standard scheduling strategies, the algorithm can be seen as being robust. During this second approach we can define the scheduling objective as in the first approach by using preference regions or by specifying a linear objective function that is based on the seven basic scheduling objectives. Note that this linear function cannot be defined without knowing the Pareto front, as the relations between the possible solutions and solutions generated by existing standard algorithms are not known in advance. The machine providers may change their preferences in the awareness of their possibilities. This attempt is not as powerful as the region based selection as only a single preference can be specified without variation. However, the calculation of a schedule objective is computationally easier and quicker as no distances to hyper planes in seven dimensions must be calculated.
Chapter 8 includes all details of our approaches to generate a complex objective function that reflects the machine providers' preferences.

6.3 Generating Appropriate Scheduling Strategies

Based on the extracted multi-objective evaluation function and on the available workload traces scheduling strategies must be developed. Here, we use two different approaches. The first approach is a Greedy scheduling strategy while the second approach is the generation of rule based scheduling systems.

6.3.1 Development of a Greedy Scheduling Strategy

In our first approach, we develop a Greedy scheduling strategy as already introduced in Section 4.4.3. Each time a new job arrives or another job finishes execution the waiting queue is reordered using job weights. Then the simple First Come First Serve strategy, see Section 4.4.1, is used to schedule the jobs in the given sequence onto the available machines. Note that the job weights are *not* given by the users. Further, they are independent of the current schedule. For each job a complex function determines the weight based on the job parameters. So, the function can, for example, use the estimated resource consumption, the estimated run time, the number of requested machines and the user group information. Furthermore, the dynamic waiting time of each job within the waiting queue is used as a parameter to determine the job weight.

The complex weight function is generated by parameterizing those job characteristics. The optimization of the weight function parameters is done by using single-objective Evolutionary Algorithms, see Section 3.1.

As already mentioned, we follow two approaches of generating a robust scheduling system. In our first approach, online schedules for all available workload traces will be generated by using the same parameter settings. Then, the previously extracted provider objective which determines the distance to the preferred region within the multiple objective space is used for each workload trace. The results are combined to build a global evaluation value for the used parameters of the complex sorting criterion. In our second approach, the optimization process is done for a single workload trace using the extracted provider objective. Then, the extracted weight parameters are used to schedule the jobs for all other workloads. If the result is similar, the whole strategy is robust.

This Greedy scheduling approach leads to a scheduling strategy that can be implemented easily as only a sorting criterion with static parameters is used. Further details are given in Section 9.1.

6.3.2 Development of Rule Based Scheduling Systems

Our second approach is the development of a rule based scheduling system. Such a system is divided into two steps. In the first step, the waiting queue is reordered according to a sorting criterion. Then a scheduling algorithm is selected which assigns the jobs in the given sequence onto the idle machines. To this end, the system classifies all possible scheduling states and assigns a corresponding sorting criterion and a scheduling algorithm to each of them. The classification is based on the system state, that is the actual schedule, the current waiting queue, and additional external factors, like the time of day.

As for the Greedy scheduling approach, the available workload traces are used offline to generate such a rule based system. As already mentioned, local scheduling decisions influence the allocation of future jobs. Hence, the effect of a single decision cannot be determined individually. Therefore, the whole rule base is only evaluated after the complete scheduling of all jobs belonging to a workload trace. This has a significant influence on the learning method to generate this rule base as the evaluation prevents the application of a supervised learning algorithm. Instead, the reward of a decision is delayed and determined by a critic.

The actual optimization of the rule based scheduling system regarding the extracted provider objective is implemented in different ways. We use five different approaches, which are explained in detail in Section 9.2.
Following our procedure for generating a Greedy scheduling strategy, we apply two different methods to generate each of the five different rule based scheduling systems. First, we use all workload traces together with the same rule base and evaluate the combined outcome. This general rule base is then modified by again using single-objective Evolutionary Algorithms, see Section 3.1. Second, the rule base is optimized for one workload trace and the result is applied to all other workloads.
The usage of a rule based scheduling system is expected to generate better schedules compared with the Greedy scheduling approach. However, the algorithmic complexity is higher and so is the amount of computational time to generate such rule bases. The details are given in Chapter 10.

Chapter 7

Generation of the Pareto Front

This chapter presents our method to generate the Pareto front for the available workloads by using the already introduced seven objectives, see Section 6.1.1. This enables the machine provider in a later step to establish scheduling objectives that fit to their environment. For further details, see Ernemann et al. [46].

7.1 Exploring the Scheduling Solution Space

The generation of the Pareto fronts for the available workload traces, see Chapter 5, is done by using several versions of multi-objective Evolutionary Algorithms, introduced in Section 3.2. As mentioned above, they can be applied to the underlying problem of solving

$$P_m|\bar{r}_j, \bar{p}_j, m_j|\#(\text{AWRT, AWRT1, AWRT2, AWRT3, AWRT4, AWRT5, UTIL}), \qquad (7.1)$$

regarding Section 2.5.1, best.
A normal scheduling strategy, like FCFS, itself is not able to generate the Pareto front as the behavior of the scheduler is deterministic and the scheduling result only depends on the given workload trace. Therefore, one deterministic scheduling strategy generates exactly one result for a single workload.
Figure 7.1 shows the simplified scheduling system. The scaled workload trace with the user group information is used as an input. The waiting queue includes all jobs that are already released but not yet started. So, the scheduler determines which of the jobs from the waiting queue is assigned to the idle machines next.

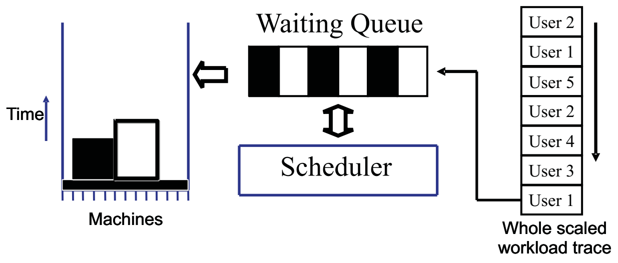


Figure 7.1: The figure shows the simplified scheduling system.

Within this work, we will use two different approaches to generate the Pareto front of possible schedules for each workload trace. However, the underlying concept of both approaches is the same. Here, we use a simple FCFS scheduling algorithm. Additionally, the following three constraints must be satisfied:

1. For all jobs $j \in \tau$: j cannot start earlier than its release date r_j.

2. All jobs are ordered: a job at position l cannot be started earlier than a job at position k if $k < l$.

3. Following the first two constraints, every job is started as soon as possible.

Using this strategy, each schedule can be represented by one permutation of the incoming job stream. Hence, it is sufficient to change the job sequence in order to generate different schedules. As no job can be started earlier than its release date, all produced schedules are valid and could be used in a real life scenario. Furthermore, this process might generate schedules that have a low utilization and longer periods of time with idle machines. This is caused by sequences where jobs with a late release date are placed before jobs with a lower release date. This reflects the effects of speculation in an online real life scenario.

Nevertheless, even these artificial cases might be optimal for the response times of jobs of a certain user group and therefore represent a Pareto optimum within the objective space. Both different approaches that are used within this work to generate the Pareto front vary the job sequence.

The first attempt is based on reordering the jobs within the scaled workload trace before this trace is used for a simulation of a schedule generation. In this case, the sequence of jobs in the waiting queue is not further modified. Contrary, the second attempt uses the original workload traces as an input and only modifies the sequence of jobs within the waiting queue. This modification of the waiting queue uses a parameterized sorting criterion. Hence, a variation of these parameters results in different sequences of jobs in the waiting queue. Both approaches will be explained in the following.

As mentioned above, our work is based on workload traces from real installations. They are used to represent the incoming job sequences. In our first approach we use a reordering of the given workload trace before a simulation by using multi-objective Evolutionary Algorithms with a representation of permutation sequences. During the execution of a single simulation, the sequence of jobs is not further modified. In the second approach, a different multi-objective Evolutionary Algorithm that uses a real-value coding modifies coefficients of a pre-defined weight function that is used to calculate a weight for each job in the waiting queue. This queue is dynamically reordered according to these weights afterwards.

The outcome of the second approach of changing the order of all jobs within the waiting queue is limited in comparison to the first approach. Theoretically, every possible sequence of jobs can be created by using the first method. The second method is restricted to local modifications of the job sequence. However, it can be implemented easier. Furthermore, the corresponding Evolutionary Algorithm uses a real-value coding of just 36 parameters. Thus, the coding of the individuals for the second approach is easier than the coding of the permutation of the whole workload trace, as done in the first approach.

Having applied both of the described approaches, the overall Pareto set is extracted. This enables the machine providers to define their preferences. This results in a complex objective function. This function can then be used to develop appropriate scheduling strategies later.

7.2 Modifying the Sequence of Jobs in the Scaled Workload Traces

The first approach of modifying the sequence of jobs within the scaled workload trace uses a permutation coded approach as described in Section 3.1.3.2. So, the multi-objective Evolutionary Algorithm operates on a sequence of jobs. The design of variation operators for this approach has taken into account some general design principles, e.g. that small changes are more probable than large changes and that by a concatenated application of the operator any element in the search space can be reached from any other element. Moreover, variations should be reversible, meaning that a single operation should have the same probability as its inverse operation.

7.2.1 Mutation Operators

Following these guidelines for permutation based optimization problems, three elementary mutation operations were identified. These are the *move*, *swap*, and *jump* operations as described in Section 3.1.3.2.

The number of operations that are carried out within one mutation can be influenced by the mutation number (MN) parameter that is part of each individual. As described in Section 3.1.3.2, this parameter describes the mean of a geometrical distribution. Each time the mutation operator is initiated, a sampling on this distribution specifies the precise number of mutations.

The mutation jump length (MJL), see Section 3.1.3.2, specifies the mean of a geometrical distribution. Independent samples of this distribution are used to specify the distance between jobs during the swap operation and the distance that a single job should jump within the sequence of jobs.

The parameters MN and MJL are initialized externally by user defined values. Within the Evolutionary Algorithm, those parameters are subjected to self-adaptation. Here, we use Rechenberg's two-point rule with $\alpha = 1.3$, see Section 3.1.5.2.1.

In detail, the *move* operator exchanges two adjacent jobs within the job sequence. The number of moved operations is determined by the geometrical distribution using the mutation number (MN) parameter. This reflects the smallest possible variation of the job sequence. Further, many move operations are necessary as several of the used workload traces consist of more than 100,000 jobs.

The second operator *swaps* two distant randomly chosen elements. Again, the mutation number (MN) is used for the number of swap operations. The mutation jump length (MJL) serves as the mean distance between the two elements. The swap operator allows for greater changes with fewer mutations compared with the move operator.

The operator *jump* moves several jobs within the sequence. The number is again determined by using the geometric distribution with the mean mutation number (MN). The distance is calculated by using the geometric distribution with the mean mutation jump number (MJN). In a first approach, the direction of the jump operation is chosen randomly. However, in a problem specific implementation of this mutation operator a job is only moved towards the end of the job sequence. This is motivated as the movement of jobs towards the end of the whole job sequence only delays these individual jobs. Contrary, a job movement towards the front of the sequence delays all jobs that are in-between. Thus, a movement of jobs towards the beginning of the job sequence artificially delays several jobs. This results in schedules of

a low quality (low utilization and high average weighted response times). Furthermore, we restricted the type of job that can be moved. Such a job must follow some rules, concerning its job properties. For example, the jobs need to have a minimal number of requested machines m_j as well as a minimal estimated processing time $\bar{p}_j$. This means that only "big" jobs are moved, which are assumed to have a greater impact on the resulting schedule.
During the generation of the Pareto front, we modified the jump operator in order to introduce problem knowledge. Within this approach, we use the information about the user group information that is given for each job. In an outer loop, every time a user group is chosen randomly. In an inner loop, the jobs of the other user groups are selected randomly and moved some positions towards the end. At the same time, only jobs with a certain amount of requested processors and a higher estimated processing time are moved. As before, the number of the jobs as well as the mean moving distance are controlled by the already introduced endogenous strategy parameters mutation number and mutation jump number which are adapted by self-adaptation. Again, Rechenberg's two point rule with $\alpha = 1.3$ has been used. By means of this new mutation operator, it is likely that the mutation operator generates an improvement in at least one of the objectives as the objectives are also based on certain user groups. Since another user group is chosen each time, improvements can be obtained in each direction of the solution space leading to a well-spread Pareto front.

7.2.2 Recombination

For the recombination of two randomly chosen individuals we selected the recombination for permutation sequences as described in Section 3.1.4. The geometric distribution with mean mutation number was also used for the recombination in order to determine the number of jobs that are involved during a recombination.
For two selected individuals within the population the recombination is only applied with a specified probability. This limits the influence of the recombination.
As our results demonstrate, the application of a recombination was not successful. Hence, we used this operator only at the beginning.

7.2.2.1 Initialization of Population

In general, we have two possibilities to initialize the starting population. On the one hand, the population can be initialized with random elements. On the other hand, we can start with some heuristically generated solutions. These start solutions are only used for generating the first population and cannot survive the first generation. Both alternatives were tested for modifying the sequence of jobs within the workload traces.
Next, we describe our approach to generate initial solutions. Here, the user group information is used to derive initial solutions as five of the seven objectives are directly connected to a special user group.
In detail, we developed a deterministic permutation rule which was statically applied to the original workload trace before the simulation. To this end, the jobs are ordered according to their corresponding user group. However, we used a more complex permutation as also combined job subsets were used. This will be explained using a small example of three user groups. One permutation is described as a tuple of ordered jobs. The notation $\{x, y\}$ describes that all jobs from both user groups x and y are interleaved as in the original sequence. With (x, y) we denote that all jobs of user group x are placed before the jobs of user group y. For

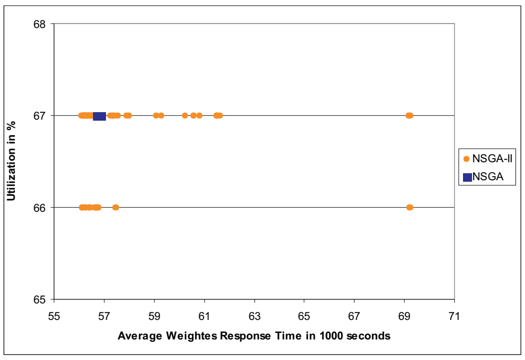
example, 13 different permutations can be generated for three user groups: (1,2,3), ({1,2},3), (1,{2,3}), {1,2,3}, (1,3,2), ({1,3},2), (2,1,3), (2,{1,3}), ({2,3},1), (2,3,1), (3,1,2), (3,{1,2}), and (3,2,1). For our five different user groups 540 orderings are possible.
As each permutation represents a valid schedule, we also incorporate those solutions within our final determination of the Pareto front for each workload trace. Further, we apply dynamically the same mechanism to the waiting queue. This means that released jobs are inserted into the waiting queue according to the described permutation scheme. This again generates valid schedules that are used to generate the final Pareto fronts.

7.2.2.2 Computational Limitations

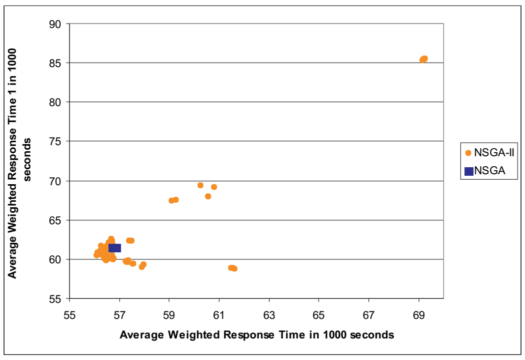
Both approaches to generate the Pareto front of possible schedules have been applied to all mentioned workload traces described in Chapter 5. During these attempts some technical restrictions exist. Each workload trace has a size of 10-15 MB. As all individuals represent a permutation of the original workload, the size of an individual is also around 10-15 MB. Therefore, the number of offsprings within the Evolutionary Algorithms is restricted to about 60 individuals as this results in a memory usage of 600 MB to 900 MB for a whole population. Otherwise, the physical memory requirements of the simulations exceed the existing physical memory of the computer (1 GB) which performs the evolutionary operations. Further, each individual scheduling simulation has been performed on a separate computer within a computing cluster. Depending on the original workload trace each simulation took about 40-120 minutes. This limits the number of trials to generate the Pareto front.

7.2.2.3 Applying NSGA to the Permutation of the Workload Trace

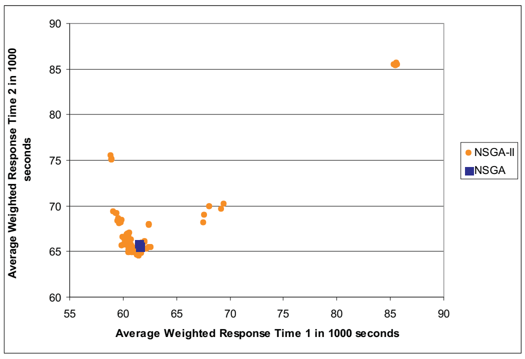
In the following, only results for the CTC workload will be presented. Except for the NASA workload all traces show a similar behavior. The NASA workload is in some sense artificial as nearly no job ever waits for the execution, that is, the jobs are started directly after submission [47]. Therefore, a reordering of the jobs is not meaningful in this case. Consequently, we will not further use the NASA workload trace.
Furthermore, we present several representative diagrams to demonstrate the outcome of our approaches to generate the Pareto fronts. In detail, we display figures for UTIL in combination with the overall AWRT. Moreover, figures for the overall AWRT and the AWRT1 are shown. The outcome for all other user groups with the corresponding AWRT values is similar. Also the AWRT1 and AWRT2 will be compared. The combination of other user group AWRT values is similar and hence not presented in detail.
First, we applied NSGA as described in Section 3.2.1.1, to modify the sequence of jobs within the workload traces. We have chosen MN=5000, MJL=1, and a randomly generated initial population. Further, we only used *move* as mutation operator. As in all following optimizations, we run 100 generations with $\mu = 30$ and $\lambda = 60$. This size is motivated by our computational limitations, see Section 7.2.2.2. These first optimizations resulted in populations with objective values worse than FCFS.
Next, we applied NSGA with MN=5000, MJL=1, and starting solutions. However, we generated only 120 such starting solutions by using the information about the five user groups and generating all job sequences without any interleaving of jobs from different user groups. Hence, 5!=120 such sequences were generated. Further, the original job sequence within the workload traces was used. This resulted in 121 initial individuals within the Evolutionary Al-



(a) UTIL versus AWRT.



(b) AWRT versus AWRT1.



(c) AWRT1 versus AWRT2.

Figure 7.2: Comparison of all results achieved by using NSGA and NSGA-II using the CTC workload trace.

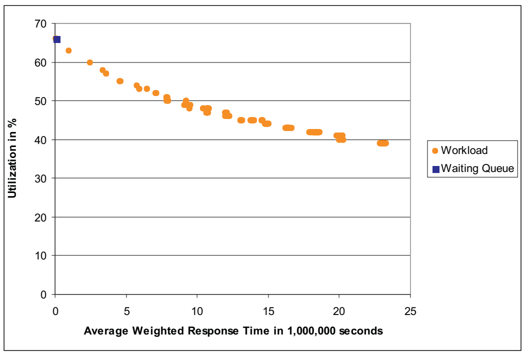
gorithms. As mentioned above, each NSGA optimization approach used only $\mu = 30$. Hence, three runs of NSGA with different initial solutions were performed by choosing $\mu = 30$ and $\lambda = 60$. As we have 121 initial individuals, a fourth run used $\mu = 31$ and $\lambda = 62$ in order to keep the ratio of $\mu/\lambda = 1/2$ constant between all evolutionary optimizations.

The use of starting solutions increased the quality, but still the number of achieved solutions was very small. The deterministically generated start solutions provide already a higher diversity than the solutions found by NSGA with randomly generated start solutions. The utilization of the start solutions varies between about 38 % and 68 %. Also the AWRT values for all user groups vary between several magnitudes.

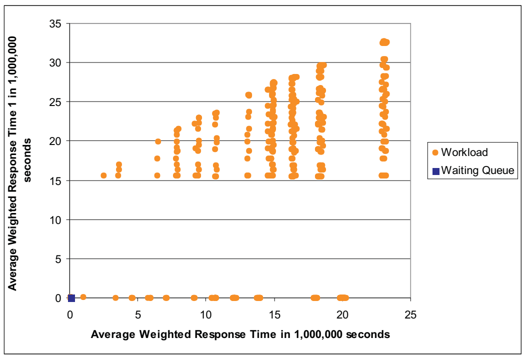
The quality of the achieved solutions was increased by changing the mutation operator by applying the swap and jump operators instead of move, see Section 3.1.3.2. This reflects the characteristics of the underlying scheduling problem as some jobs should be moved so that several later jobs can overcome this job in order to increase the scheduling quality for certain objectives. In detail, we used MN=5000 and MJL=4000. However, the results for this permutation approach using NSGA were of poor quality as only a few non-dominated solutions were found with a bad diversity compared with the deterministically generated initial solutions. This outcome might be explained by the non-elitist character of NSGA. Existing non-dominated solutions might get lost during the optimization.

In Figures 7.2a to 7.2c all achieved non-dominated solutions by modifying the job sequence in

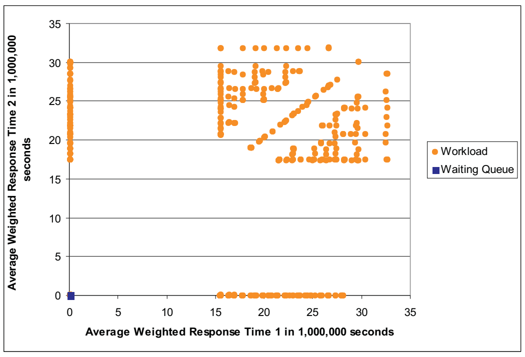
the workload traces for the NSGA and NSGA-II variants, see the next section, are presented. The results clearly show, that the number of found solutions for both approaches is very small. In the sum, we only found three non-dominated solutions using NSGA, see Table 7.1 on page 79.
Compared with the deterministically generated initial solutions, see Figures 7.3a to 7.3c, the number of non-dominated solutions as well as the diversity of the whole population is worse.



(a) UTIL versus AWRT.



(b) AWRT versus AWRT1.



(c) AWRT1 versus AWRT2.

Figure 7.3: Start solutions generated by the permutations of the whole workload (Workload) and the local waiting queue (Waiting Queue).

7.2.2.4 Applying NSGA-II to the Permutation of the Workload Trace

In the following, NSGA-II was used with the same subset of the deterministically generated start generation as used for the NSGA approach (121 solutions) and the changed mutation operator. The number of solutions and the diversity increased. Obviously the introduced elitism of NSGA-II is necessary to generate better multi-objective solutions compared with NSGA. Furthermore, the results demonstrate the advantage of using the crowding distance sorting within NSGA-II in comparison to the sharing method used in NSGA. The crowding distance sorting method results in a better diversity of the whole population. This is especially true for solutions that are not in the non-dominated set. A higher diversity in such regions results in a better diversity of the non-dominated solutions in later generations. Nevertheless, the overall number of found non-dominated solutions at about 100 was small (see Table 7.1). The diversity of the non-dominated set was also small. Therefore, we applied three modifications to the system. First, we restricted the direction of the mutation in the way that jobs can only jump into the direction of the end of the sequence. Our second change, which was motivated from the previous simulations, results in the deselection of the recombination through the multi-objective Evolutionary Algorithm as all results which were created using recombination were of low quality. Third, the usage of good starting solutions was helpful to improve the quality of the solutions. Nevertheless, a widely spread front of non-dominated solutions could not be processed. Therefore, multiple copies of the original job sequence were used as the only start solutions instead of the deterministically generated job sequences. All three modifications were applied to the NSGA-II algorithm. This did not affect the number of found solutions, but the diversity increased. The last additional modification that was applied to the NSGA-II algorithm was the selection of certain jobs during the mutation. Here two restrictions were introduced as only jobs with a requested number of machines of more than 31 and/or jobs with an estimated processing time of more than 1 hour were chosen for mutation. We applied both the "and" and the "or" condition. This again increased the diversity of the achieved results. Further, it reflects the influence of those jobs with a higher resource demand on the resulting scheduling objectives.

Nevertheless, the number of found non-dominated solutions at about 130 and the achieved diversity in terms of the variation of the seven simple objectives were not satisfying, see Table 7.1 on page 79. The final set only consists of solutions with 66 % or 67 % utilization and the AWRT does not vary much in comparison to the given start solutions.

7.3 Modifying the Job Sequence within the Waiting Queue

To reduce the problem dimension, we developed an alternative representation. Instead of directly determining the positions of the jobs in the workloads, we optimize the parameters of a complex sorting criterion. This sorting criterion is applied to the waiting queue. Then an FCFS scheduler is used to assign the jobs in the given sequence to the idle machines. As mentioned above, this approach is limited as only the jobs within the waiting queue are reordered.

By means of this indirect representation of job sequences, it is possible to reduce the number of variables that need to be optimized to 36 real-value parameters. Of course, the suggested representation has the disadvantage that not all possible job sequences can be coded. But even in the former strategy, due to the huge search space it is impossible to generate all possible schedules within the given computational time. However, it can be assumed that the represented solutions comprise an interesting part of the search space with regard to the problem at hand.

All situations where a new job is released or a running job finishes execution are divided into three classes. This is the outcome of other studies [47] where significant patterns of workloads were extracted. The three mentioned classes are weekends, week days between 8 am and 6 pm and week days between 6 pm and 8 am. Within each of these classes, we use one of four different sorting criteria to calculate job weights that are used during the sorting. The four complex criteria that determine the weight of job j are:

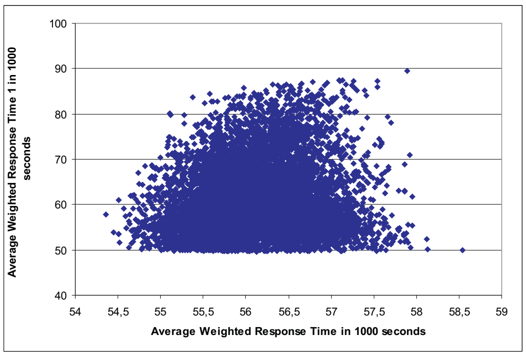
$$f_1(j) = \sum_{i=1}^{5} \left[\varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot \frac{t - r_j}{\bar{p}_j} + b \cdot \frac{\bar{p}_j}{m_j} \right) \right] \quad (7.2)$$

$$f_2(j) = \sum_{i=1}^{5} \left[\varrho_i(j) \cdot w_i \cdot (K_i + a \cdot (t - r_j) + b \cdot \bar{p}_j \cdot m_j) \right] \quad (7.3)$$

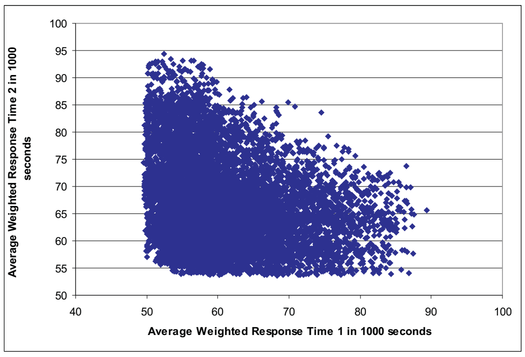
$$f_3(j) = \sum_{i=1}^{5} \left[\varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot \frac{t - r_j}{\bar{p}_j \cdot m_j} \right) \right] \quad (7.4)$$

$$f_4(j) = \sum_{i=1}^{5} \left[\varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot (t - r_j) + b \cdot \frac{\bar{p}_j}{m_j} \right) \right] \quad (7.5)$$

Here, t denotes the actual time. The time is needed to calculate the actual wait time of job j as $(t - r_j)$. Exemplarily, we will discuss the sorting criterion 7.2, which determines the weight for job j. Depending on the user group i of job j, a different parameter w_i is used to scale the resulting weight of this job. The parameter a scales the ratio of the actual waiting time of job j and its estimated processing time. The ratio of the estimated processing time of j and the number of requested processors is multiplied by b. The parameters a and b are user group independent. Furthermore, K_i increases or decreases the importance of $(t - r_j)/\bar{p}_j$ and $\bar{p}_j/m_j$ depending on the user group i that job j belongs to. All other functions are very similar. The NSGA-II algorithm, described in Section 3.2.2.1, modifies the static parameters w_i, K_i, a, and b in order to adapt the resulting scheduling behavior. Note that for each user group i individual w_i and K_i are defined. This results in 12 parameters (a, b, w_i, K_i, $i \in \{1, \ldots, 5\}$) that are needed to parameterize one of the introduced sorting criteria. As the scheduling states are divided into 3 classes (weekend, weekday 8 am - 6 pm, weekday 6 pm - 8 am) each having a corresponding sorting criterion, 36 parameters are sufficient to describe the resulting job sequence precisely. Furthermore, the assignment of the sorting criteria to the three different classes is varied. The definitions of the criteria 7.2 to 7.5 are based on the adjustment of parameters w_i and K_i depending on the user group i that job j belongs to. This is motivated as the Pareto front represents the different goals of the participating user groups.



(a) AWRT versus AWRT1.



(b) AWRT1 versus AWRT2.

Figure 7.4: Results of the real-value coded approach using the CTC workload trace.

As mentioned above, a real-value instantiation of an NSGA-II was used to optimize the

function parameters. We employed an NSGA-II with $\mu = 30$, $\lambda = 60$, $\rho = 2$, and $\kappa = \infty$. Further, 100 generations have been used during the optimization. The parameters are initialized with values chosen uniformly at random between problem specific lower and upper bounds. In detail, the value intervals used are: $w_i \in [0, 1]$, $a \in [0, 1]$, $b \in [0, 1]$, and $K_i \in [0, 5]$. After the μ individuals of the initial population have been generated randomly, they are evaluated. The Greedy scheduler reorders the jobs in the waiting queue depending on the calculated job weights. The functions to calculate those weights are parameterized by NSGA-II. Then the seven simple objective functions are evaluated and these values are returned to NSGA-II. From the perspective of the Evolutionary Algorithm, the combination of the used scheduling strategy and the evaluation of a job sequence serves as a black-box. Thus, the evaluation of each individual is just a function call of the scheduling strategy.
After having evaluated the initial population, the evolution process starts. Again, we use 100 generations. Now, λ offsprings are generated from the current population by means of variation operators.

7.3.1 Recombination

With a probability of 90 % we applied the Simulated Binary Crossover operator, see Section 3.1.4. The setting of the necessary parameter $\eta_c = 20$ is directly taken from Deb et al. [29].

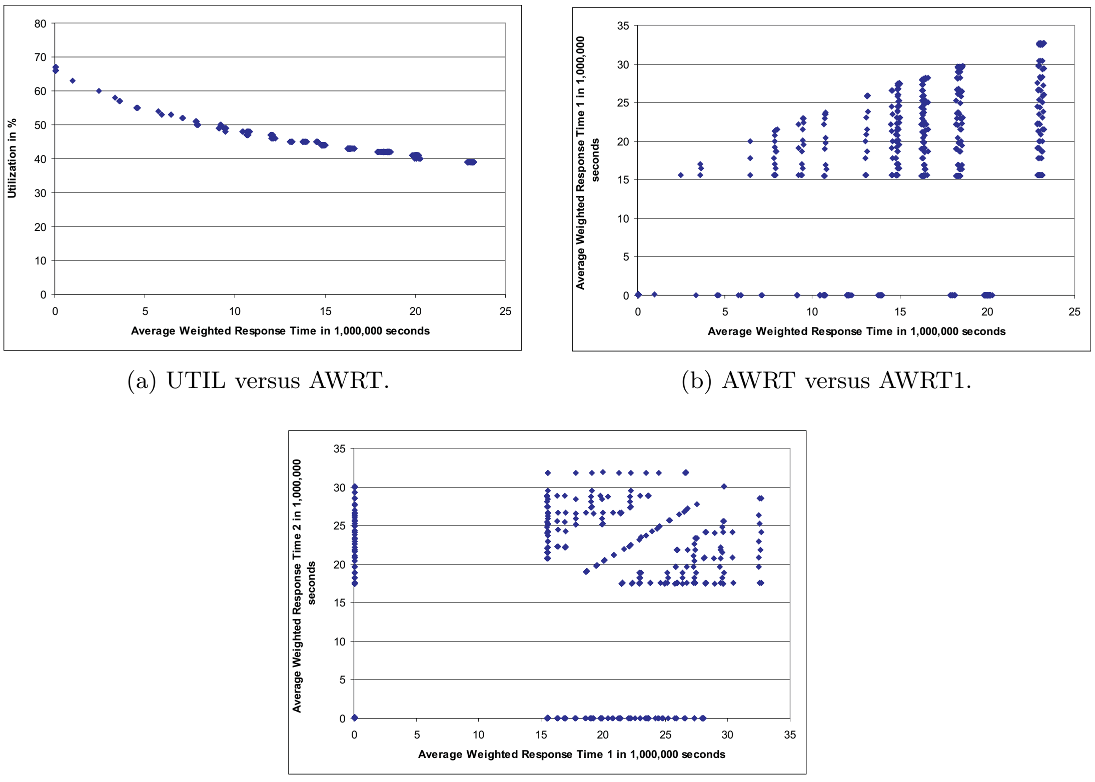
7.3.2 Mutation

After the recombination, an offspring individual is changed by the polynomial mutation operator, see Section 3.1.3.1. Here with probability equalling the reciprocal number of parameters, each gene is slightly modified by an additive term that is distributed according to a polynomial function. Again, the parameterization of this variation operator ($\eta_m = 20$) is exactly chosen as given in the literature [29].

7.3.3 Achieved Results

The results for the second approach by dynamically modifying the waiting queue with a real-value coded Evolutionary Algorithm are presented in Figures 7.4a and 7.4b. This time, we used $\mu = 50$ and $\lambda = 50$. As stated above, the influence of this attempt is limited as only a limited number of jobs are reordered. As a result, the utilization does not vary much and is nearly constant at about 66 %. However, the diversity of the overall AWRT and the AWRT values of all user groups is of high quality. Figure 7.4a represents the relative outcome of a special user group (here user group 1) to all users for the AWRT. Here, the AWRT1 has a lower bound of about 50,000 seconds. The overall AWRT does not have such a clear bound. The comparison of the AWRT values between two user groups presents a well spread Pareto front as it can be seen in Figure 7.4b. For both representative user groups a clear lower bound can be observed.
The diversity of the starting solutions is higher than the diversity of the solutions generated by the parameter value coded approach. Nevertheless, this second approach has a much better diversity than the permutation coded approach. Additionally, all solutions are in the "most" interesting area with a high machine utilization but a higher variation of waiting times between the different user groups. Furthermore, the second approach provides about 13,000 non-dominated solutions and therefore highly covers the solution space in this restricted area.

7.4 Final Pareto Front



(a) UTIL versus AWRT.

(b) AWRT versus AWRT1.

(c) AWRT1 versus AWRT2.

Figure 7.5: Final Pareto front for the CTC workload trace.

The final approximation of the Pareto front is calculated by using all solutions of the permutation coded approach, the real-value coded approach, and all starting solutions. Figures 7.5a to 7.5c present this set which in the case of the CTC workload trace consists of about 14,000 solutions. The complete approximated Pareto front is displayed in Appendix B on page 183. For all objectives the front has an acceptable diversity as the parameters for the waiting times are varying between several magnitudes and the utilization is varying between about 38 % and 67 %. This set is used for the further development of an appropriate scheduling strategy. Table 7.1 summarizes all results. The deterministic permutation of the whole workload and the local waiting queue produce 540 non-dominated solutions each. The overall performance of the permutation coded Evolutionary Algorithms is very low in comparison to the real-value coded approach. In this scenario, the NSGA-II algorithm performs better than NSGA and the applied modifications increase those effects. The real-value coding solves the problem of generating the approximation to the Pareto front best. Here the number of found solutions is very high. Therefore, this strategy should be applied in such high dimensions and with workloads of a similar size.

Representation	EMOA	Mutation	Adaptation	Recombination	StS	NoS	FC
Workload Permutation	-	-	-	-	-	540	540
Queue Permutation	-	-	-	-	-	540	540
Permutation Coded	NSGA	MJL=1, MN=5,000	-	0.3	121	3	24,000
Permutation Coded	NSGA	MJL=4,000, MN=5,000	-	0.3	121	1	24,000
Permutation Coded, All NSGA Results	NSGA	-	-	-	-	3	48,000
Permutation Coded	NSGA-II	MJL=4,000, MN=5,000	-	0.3	121	106	24,000
Permutation Coded	NSGA-II	MJL=1,500, MN=5,000	Movement of jobs towards the end.	0.0	1	108	30,000
Permutation Coded	NSGA-II	MJL=1,500, MN=5,000	Randomly chosen user group. Movement of jobs (with 32 processors and/or 1 hour run time) of other groups towards the end.	0.0	1	125	30,000
Permutation Coded, All NSGA-II Results	NSGA-II	-	-	-	-	131	84,000
Real-Value Coded	NSGA-II	Polynomial	Real-Value Coded	SBX	-	12,832	20,000
Final Solution	-	-	-	-	-	13,954	153,080

Table 7.1: Summarized evaluation results. EMOA $\hat{=}$ Evolutionary Multi-Objective Algorithm. MJL $\hat{=}$ Mutation Jump Length. MN $\hat{=}$ Mutation Number. Recombination describes the probability to use the recombination operator for two randomly selected individuals. StS $\hat{=}$ number of start (initial) schedules. NoS $\hat{=}$ number of non-dominated solutions. FC $\hat{=}$ function calls, which is equal to the number of generated schedules.

Chapter 8

Establishing of the Complex Evaluation Function

In the area of scheduling high performance parallel computers, the machine providers are usually not able to describe their objectives precisely in advance. Hence, the previously generated approximation of the Pareto front of possible schedules in combination with the schedules generated by commonly applied standard strategies are used to define the provider's preferences. During this process, we focus on the main goal of generating a robust scheduling strategy. Here, we have chosen two different approaches to derive the complex provider objective.

In the first approach, we scale and move the existing approximations of the Pareto fronts, based on the different workloads, in order to generate a single representative Pareto front. Then, we establish a partial ordering of convex regions within this representative front. These partially ordered regions reflect the preferences of the providers. Further, the condition of convexity is needed to easily assign a unique preference to every possible schedule. The complex provider objective is then built by incorporating the distances of a given schedule to the specified convex regions with different preferences. Consequently, it is not possible to directly establish a mathematical formulation of the provider's objective that only depends on several simple objectives. However, the definition within the objective space based on distances to the Pareto front and certain regions reflects the preferences of the providers. As we neither have special knowledge about certain provider preferences nor want to implement a pre-defined objective, we will exemplarily select the preference regions. Here, the previously generated representative Pareto front is used as this prevents contradicting definitions for the different workload traces. The resulting objective function is then used for generating an appropriate scheduling strategy.

In the second approach, the providers choose the corresponding complex objective function within one of the available Pareto fronts. Then a scheduling strategy is developed for the underlying single workload trace. After this, the extracted strategy is applied to all other workloads. If the results show a similar behavior, the scheduling strategy can be called robust. Otherwise, the application of the extracted strategy is limited to the individual workload trace that was selected during the strategy development.

Within this chapter, we present both approaches in detail. The optimization of scheduling strategies by using the extracted objectives will be part of the next section.

8.1 Generation of a Representative Pareto Front

As mentioned above, the goal of the first approach is the generation of a representative Pareto front that is created by transforming all available approximations of the Pareto fronts of the various workload traces to a unique front.
During this process, which will be described later in more detail, we only scale and move the various fronts within the multidimensional objective space. We have chosen not to use rotation or some other transformations as the machine providers need to select preferred regions within the resulting Pareto front. However, this is only possible, if the relative relationship between the various generated schedules holds. The usage of rotation or interval wise scaling would change those relationships and is therefore excluded from our work.
From the mathematical point of view, each Pareto front consists of a set of grid points $\vec{x}_i$ within a seven dimensional space. The transformation of those Pareto fronts can be based on three different representations of the corresponding sets of grid points.

1. The original set of grid points for each front can be used or
2. an interpolation of each Pareto front or
3. an approximation of each Pareto front.

Both the interpolation and the approximation of Pareto fronts would result in a mathematical expression that can be modified easily in most cases. Furthermore, the calculation of distances between various fronts might become easier in those cases. An interpolation includes all grid points of a Pareto front exactly, whereas an approximation only tries to come very close to the majority of the provided grid points. Here, we restricted ourselves to using the original grid points and an interpolation. An approximation usually smoothes the underlying multidimensional surfaces. However, for our generated Pareto fronts it cannot be assumed that such a smoothing is valid. Thus, an approximation is not applied. Next, we describe the interpolation of a single approximated Pareto front. Note that the practical usage of the interpolation in our work has proven to be computationally expensive as the application to a large Pareto front needs about 3,000 seconds. Thus, we do not apply the presented interpolation in the remainder of this thesis. However, we still present our interpolation approach in order to provide all of our efforts.

8.1.1 Interpolation of a Single Pareto Front

This section provides some details about the used interpolation approach. In general, normal interpolation algorithms in $\mathbb{R}^2$ use a system of equations to generate a polynomial of degree $(n-1)$: $P(x) = \sum_{i=0}^{n-1} a_i x^i$ if the original set includes n grid points. In those cases only the coefficients a_i need to be determined. However, those methods are mainly applied in $\mathbb{R}^2$. Methods for $\mathbb{R}^3$ exists. They often require a triangulation of the provided points. Such triangulations within a multidimensional space are computationally very expensive and not recommended for a larger set of grid points, see Alfred [3]. Also methods like the Shepard-Interpolation, see Shepard [141] or Farwig [49], cannot be used as they are not designed for seven dimensional spaces. Hence, we have chosen the interpolation by *radial basis functions* that have proven to work well for scattered data in multidimensional spaces, see Micchelli [107]. Therefore, we will introduce this concept in some more detail.

Using the interpolation by a radial basis function $\phi(r)$, the influence of a grid point on the function value of an interpolation point depends on the distance to this interpolation point. This motivated the term *radial*.
In detail, we assume that the Pareto front consists of n grid points. First, one of the dimensions needs to be selected to represent the result of the interpolation y. In our case, we have selected the utilization as this objective is restricted to values between 0 % and 100 % and behaves similar for all available workloads. All other dimensions now represent the grid points $\vec{x}_o$, $o \in \{1, \ldots, n\}$, for the interpolation process (AWRT, AWRT1 to AWRT5). Each individual grid point has dimension d (in our case 6) so $\vec{x}_o \in \mathbb{R}^d$. Now, the interpolation can be described by $s_\phi(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$:

$$s_\phi(\vec{x}) = \sum_{o=1}^{n} \alpha_o \phi(\|\vec{x} - \vec{x}_o\|) + \sum_{l=1}^{Q} \beta_l p_l(\vec{x}). \tag{8.1}$$

Equation 8.1 describes the interpolation of the grid points by using the radial basis function $\phi(r)$ with radius

$$r = \|\vec{x} - \vec{x}_o\|. \tag{8.2}$$

Note that by $\|\cdot\|$ we denote the Euclidean distance. In our case, $s_\phi(\vec{x})$ is the utilization of a given point $\vec{x}$ that represents the AWRT and AWRT1 to AWRT5.
Micchelli [107] describes the interrelationship between the conditionally positive definite basis function (definition given by Micchelli [107] or by Schaback [131]) $\phi(\|\vec{x} - \vec{x}_o\|)$ of order m and the minimum degree of the regression function $p_m(\vec{x})$ as the order m is used to determine Q:

$$p_m(\vec{x}) = \sum_{l=1}^{Q} \beta_l p_l(\vec{x}) \text{ with } Q := \binom{m-1+d}{d}. \tag{8.3}$$

As introduced and defined by Schaback [131], $p_1, \ldots, p_Q$ are a basis of $\mathbb{P}_m^d$, that is, a basis of all d-dimensional polynomials with order up to m.
The values α_o ($o \in \{1, \ldots, n\}$) and β_l ($l \in \{1, \ldots, Q\}$) of Equation 8.1 are determined by solving a system of equations, see Equation 8.4. Equation 8.5 is only needed if the radial basis function has a degree $m > 0$, see Amidor [4]. In this case ($p_m(\vec{x}) \neq 0$), the system of equations is over-determined. Hence, the additional conditions as described in Equation 8.5 are necessary, see Micchelli [107] or Amidor [4].

$$\sum_{o=1}^{n} \alpha_o \phi(\|\vec{x}_k - \vec{x}_o\|) + \sum_{l=1}^{Q} \beta_l p_l(\vec{x}_k) = y_k, \ \forall\ k \in \{1, \ldots, n\} \tag{8.4}$$

$$\sum_{o=1}^{n} \alpha_o p_l(\vec{x}_o) = 0, \ \forall\ l \in \{1, \ldots, Q\} \tag{8.5}$$

The resulting system of equations is solved using the iterative conjugated gradient method, see, for example, Bai and Li [10]. This is necessary as a standard method like the Gaussian elimination process requires too much computation time. Note that some of our fronts consist of more than 13,000 grid points, see Table 7.1. This results in more than 13,000 equations. Additionally, the probability of smaller errors due to the imprecise computer calculations using the Gaussian elimination process is relatively high. Thus, we do not use the Gaussian elimination process or similar approaches. The conjugated gradient method is numerically more stable in such scenarios, see Bai and Li [10]. Furthermore, the precision of the results using this method can be influenced by the number of used iteration. For details, see Bai and Li [10].

After presenting the concept of the interpolation by radial basis functions, the used basis functions need to be defined. As already mentioned, Micchelli [107] has shown, that all conditionally positive definite functions can be used. Such functions have an order of $m = 0$. Hence, the regression term does not need to be considered and the interpolation is established by solving the system of equations, see Equation 8.4. In the following, we list the three used radial basis functions:

$$\phi(r) = e^{-\gamma r^2},\ \gamma \geq 0,\ m = 0 \tag{8.6}$$

$$\phi(r) = \sqrt{\gamma^2 + r^2},\ m = 1 \tag{8.7}$$

$$\phi(r) = \frac{1}{\sqrt{\gamma^2 + r^2}},\ m = 0 \tag{8.8}$$

Within the Gaussians function, see Schaback [131], in Equation 8.6 the term γ describes the influence of the individual grid points on the interpolation value. The further a grid point the lower the influence. Equation 8.7 presents the basis function *Multiquadrics* described by Alfeld [3] and Hardy [75]. However, in this case, the radial basis function has a highest order of $m = 1$, see Alfeld [3]. Hence, a regression term is needed if this function is applied. Alfeld claims that this function is often successfully used. Nevertheless, the increasing influence of a grid point by an increasing radius contradicts the main idea. This leads to the *inverse Multiquadrics* definition with order $m = 0$, as given in Equation 8.8, see Hardy [75].

The application of the above introduced interpolation by radial basis functions is still computationally expensive, as the calculation of a single interpolation for a larger Pareto front needs about 3,000 seconds. This is not acceptable, as during the later transformation of Pareto fronts the interpolation is very frequently required in order to determine a mathematical representation that can be used to calculate distances between Pareto fronts. Consequently, we did not use interpolated Pareto fronts but operated directly on the available grid point information.

8.1.2 Distance between Two Pareto Fronts

During the transformation of Pareto fronts we need to measure the distance between two Pareto fronts in order to develop the final representative Pareto front by minimizing the overall distances between the different Pareto fronts. In the following, by PF_1, PF_2 we will denote two such Pareto fronts. Further, a single point within such a Pareto front is denoted by $\vec{p} \in \mathbb{R}^7$ and specifies all seven objectives. The distance between two fronts can be calculated with several approaches. Note that as mentioned above, the representation of Pareto fronts is based on the corresponding grid points and not on a mathematical function. Hence, the distances of Pareto fronts need to be derived from the distances between the grid points of the different fronts. To this end, there are several possible measurements of calculating those distances. Here, we have used two approaches.

In the first approach, the distance of a single grid point of the first Pareto front ($\vec{p}_1 \in PF_1$) to the second Pareto front PF_2 is determined by the distance to the nearest grid point of the second front ($\vec{p}_2 \in PF_2$) which is denoted by $D(\vec{p}_1, PF_2)$. Thus, we can formulate the distance as:

$$D(\vec{p}_1, PF_2) = \min_{\vec{p}_2 \in PF_2} \|\vec{p}_1 - \vec{p}_2\|. \tag{8.9}$$

Note that with $\|\cdot\|$ we specify the Euclidean distance. In the second approach, we calculate the distance of a single grid point of the first Pareto front ($\vec{p}_1 \in PF_1$) to the other Pareto front (PF_2), by generating a hyperplane HP that includes the seven nearest grid points of the second Pareto front PF_2 to the point $\vec{p}_1 \in PF_1$ and calculating the Euclidean distance of $\vec{p}_1$ to the hyperplane HP:

$$D(\vec{p}_1, PF_2) = D(\vec{p}_1, HP). \tag{8.10}$$

Based on the distance measurements of single points to Pareto fronts we can define several distance measurements of two Pareto fronts PF_1 and PF_2. The first possibility is to calculate the average unsigned distance of all points of both Pareto fronts in each case from the other front, see Equation 8.11. Note that the distance measurement of a single grid point to a Pareto front can be one of the previously introduced two forms. $|PF_1|$ and $|PF_2|$ specify the number of grid points in the two Pareto fronts PF_1 and PF_2.

$$D(PF_1, PF_2) = \frac{1}{|PF_1| + |PF_2|} \left(\sum_{\vec{p} \in PF_1} \|D(\vec{p}, PF_2)\| + \sum_{\vec{p} \in PF_2} \|D(\vec{p}, PF_1)\| \right) \tag{8.11}$$

The second possibility uses the squared average of the distances of all points in both Pareto fronts in each case to the other front as shown in Equation 8.12.

$$D(PF_1, PF_2) = \frac{1}{|PF_1| + |PF_2|} \left(\sum_{\vec{p} \in PF_1} D(\vec{p}, PF_2)^2 + \sum_{\vec{p} \in PF_2} D(\vec{p}, PF_2)^2 \right) \tag{8.12}$$

In this work, we restricted ourselves to these two presented possibilities to calculate the distance between two Pareto fronts. However, there are of course many more possible distance measurements. Now, we are able to quantitatively determine the quality of transforming Pareto fronts with the goal to achieve the lowest possible distances between them.

8.1.3 Transformation of Pareto Fronts

Based on the presented distance measurements, we describe our three transformation methods that have been used in this work.

8.1.3.1 Maximum Normalization

The first transformation of Pareto fronts in order to achieve a representative front is the *maximum normalization.* This normalization maps the values of every dimension to the interval of $[0, 1]$. For a single dimension the transformation can be described by Equation 8.13 assuming that the values of this dimension of the Pareto front are represented by x.

$$\begin{aligned} x \in [0, x_{max}] &\rightarrow x' \in [0, 1] \\ x' &= \frac{x}{x_{max}} \end{aligned} \tag{8.13}$$

This kind of transformation has the advantage that all dimensions of the Pareto fronts have the same influence on the distance measurement, as all are mapped to the interval of $[0, 1]$. Furthermore, all values are different from the value 0 as long as no original value is equal to 0. However, the disadvantage of this procedure is that the codomains of the various Pareto fronts still differ and therefore cannot be compared easily.

8.1.3.2 Value Range Normalization

This leads to the second approach of using a normalization of the codomains that is described by Equation 8.14.

$$\begin{aligned} x \in [x_{min}, x_{max}] &\rightarrow x' \in [0,1] \\ x' &= \frac{x - x_{min}}{x_{max} - x_{min}} \end{aligned} \tag{8.14}$$

Now, the codomains are equal. However, this approach has the disadvantage of having values of 0 for the former minimum values within each codomain. This destroys the normal view of the machine provider as they are normally not aware of such values (for example a response time of 0 cannot be improved).

8.1.3.3 Transformation by using Scalings and Movements

In opposite to both normalization transformations the third approach uses individual scalings and movements for the different existing dimensions in order to transform one Pareto front to another. We will not use rotations to transform Pareto fronts as this would result in not overlapping axes of coordinates. In order to find the right settings for the transformation process, we have again used Evolution Strategies. Furthermore, the quality of the transformation is calculated by using both measurements introduced in Section 8.1.2. In detail, the moving and scaling are shown in Equation 8.15. Both parameter t and s are determined for each dimension of each Pareto front individually by the usage of an Evolution Strategy.

$$x' = s \cdot x + t \tag{8.15}$$

8.1.3.4 Transformation Method Selection

For the final transformation of Pareto fronts, we have selected the transformation by scaling and moving as only this form ensures the relative ordering of simulation results within one Pareto front by avoiding the problems that occur during the normalization process. Hence, the results of the transformation only correspond to this method. The final process of generating a representative Pareto front first selects one of the available fronts and then tries to transform all other fronts to this selected front. To this end, Evolution Strategies are used to find the fitting parameters t and s for all dimensions of all Pareto fronts individually.

8.1.4 Parameter Optimization using Evolution Strategies

Next, we describe the corresponding settings of the Evolution Strategy that was used to determine the values for the different t and s values. As we have seven objectives, we need to

determine seven t-values and seven s-values corresponding to Equation 8.15 for each transformation.
We used a standard $(\mu, \kappa, \lambda, \rho)$ Evolution Strategy as described in Section 3.1. For each optimization run we used 100 generations. Further, we set $\kappa = 100$, that is, we used a plus strategy $(\mu + \lambda)$. For our transformations we used several settings for μ and λ: $(1 + 10)$, $(7 + 35)$, $(15 + 100)$, $(20 + 140)$, and (25+170). The chosen ratio of $\mu/\lambda = 1/7$ is suggested by Schwefel [136].
The value ranges for the scaling and moving parameters for the different axes have been limited. This limitation is reasonable and helps the Evolution Strategy to find good settings quicker. In detail, we specified a value range of $-2,000,000s \leq t \leq 2,000,000s$ for the t-values for all axes that specify overall or user group dependent average weighted response times. The corresponding s-value is limited to $0.5 \leq s \leq 2$. For the scaling and moving of the utilization axis we defined the following value ranges: $-20\% \leq t \leq 20\%$ and $0.7 \leq s \leq 1,4$.
Within the Evolution Strategy, we used the mutation scheme for real-value coded problems as described in Section 3.1.3.1. Further, we used a mutation strength vector that was adapted as described in Section 3.1.5.2.2. That is, we used a separate mutation strength for each value. The two exogenous learning rates are set to:

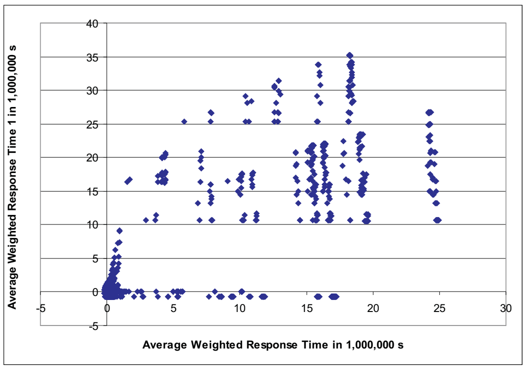
$$\tau_0 = \frac{1}{\sqrt{2 \cdot 14}}, \text{ and} \tag{8.16}$$

$$\tau_1 = \frac{1}{\sqrt{2\sqrt{14}}}. \tag{8.17}$$

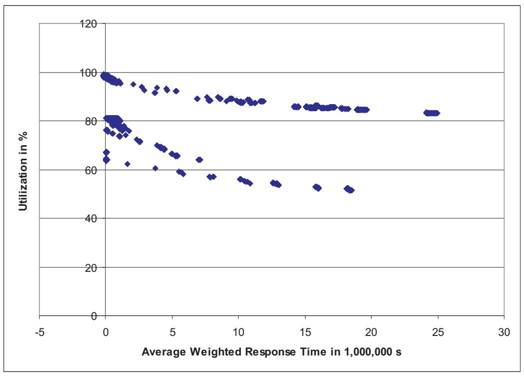
A discrete recombination was applied to the object parameters whereas an intermediate recombination was used for the strategy parameter, see Section 3.1.4. The recombination uses $\rho = \mu$.
For the transformations of the Pareto fronts to a representative Pareto front, we used the CTC workload trace as the reference and tried to scale and move all other fronts in order to minimize the distances to this front. The distance between the fronts was determined by using Equations 8.9 and 8.11.

8.1.5 Transformation Results

The results of our method to generate a representative Pareto front are given within this section. However, we do not present all results in detail but show the resulting Pareto front. In Figure 8.1a the average weighted response times over all users and user group 1 are drawn. The transformation results are not acceptable, as some of the average weighted response times have negative values. Furthermore, the triangular structure of the individual Pareto fronts, see, for example, Figure 7.5b, cannot be observed anymore. Figure 8.1a includes at least two of such triangular structures.
The same holds for all other dimensions. For example, Figure 8.1b displays the relationship between the average weighted response time and the utilization. Again, the resulting Pareto front includes negative values for the average weighted response time. Furthermore, the figure clearly displays two Pareto fronts compared with Figure 7.5a. The complete resulting front after transforming all Pareto fronts is presented in Appendix C on page 187.
The results of our transformation approach cannot be used for further processing as the quality is too low. One of the reasons for the transformation results are the extremely different ranges of values for the different axes and different workload traces. Therefore, the optimization



(a) AWRT versus AWRT1.



(b) AWRT versus UTIL.

Figure 8.1: Resulting Pareto front after transforming all workload traces.

process is highly influenced by outliers. The distance measurement between Pareto fronts pays the same attention to all grid points. Hence, it is beneficial to focus on the scaling of outliers in order to minimize the overall distance. Unfortunately, this behavior leads to inappropriate transformation results for most of the other grid points. Especially, the utilization objective is more or less neglected. Those perceptions lead to an approach to reduce the number of outliers that negatively affect the transformation result.

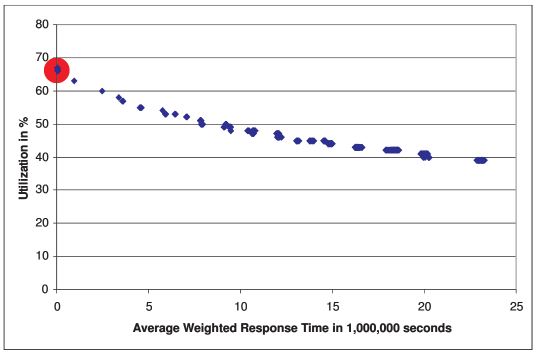
8.1.6 Pareto Front Reduction

Resulting from the previous observations, we analyzed the data of all Pareto fronts individually. For all generated Pareto fronts we found that a selection of all schedules with the highest utilization values does not significantly affect the amount of non-dominated solutions. Furthermore, the selection of those schedules with the highest utilization values results in a very low average weighted response time for all individual user groups and all users together.

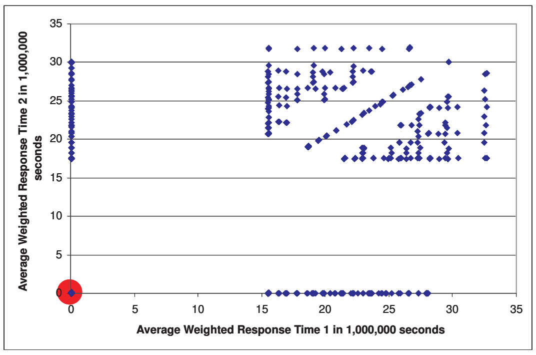
Further analysis shows that all other schedules do not belong to the class of nondelay schedules, see Section 2.2. However, also some schedules with a very high utilization are not nondelay schedules. In those cases, the delays are small.

It can be assumed that all schedules with a low utilization are specialists. They optimize a single objective with a small improvement compared to the other solutions by drastically decreasing all other objectives. Hence, those solutions represent the mentioned outliers. In Figures 8.2a and 8.2b the Pareto front for the CTC workload trace is drawn. The circle includes all non-dominated solutions, which we select for the following processing. All other solutions will be neglected.

In Figures 8.3a, 8.3b, and 8.3c the selected non-dominated solutions are presented in more detail. As for all other workloads, the removal of outliers does not affect the number of non-dominated solutions significantly. Moreover, the coverage of the solution space by the approximated Pareto front is good. All other diagrams for the reduced CTC based Pareto front are presented in Appendix D on page 191. Figure 8.3c presents one of the major outcomes of this removal procedure. As the utilization values of the reduced Pareto front are nearly equal and significantly higher than the utilization values of all other non-dominated solutions, it can be concluded that the utilization objective can be removed from the set of possible objectives.



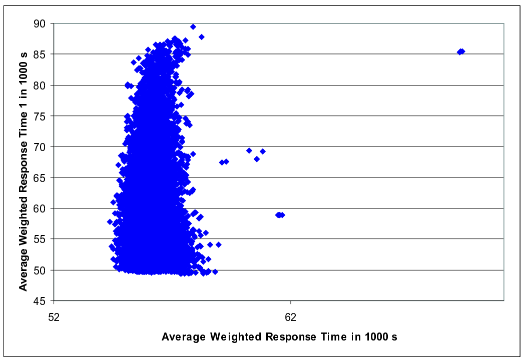
(a) UTIL versus AWRT



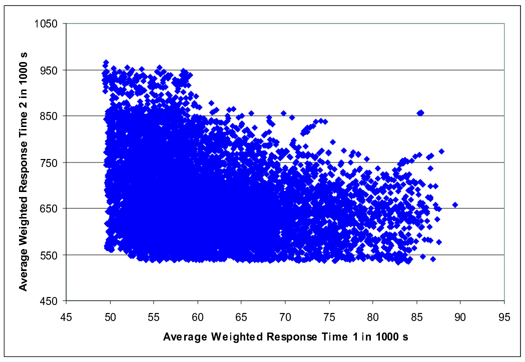
(b) AWRT1 versus AWRT2.

Figure 8.2: Selection of the reduced data set from the set of non-dominated solutions of the CTC workload trace.

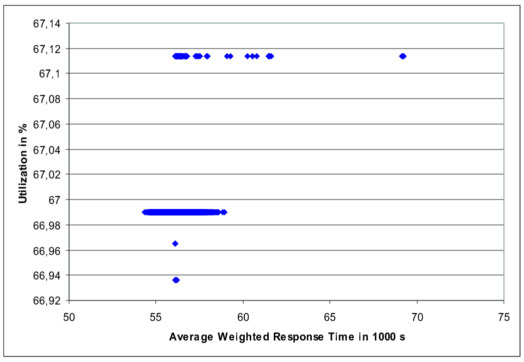
In other words, if the providers minimize the average weighted response times for the existing user groups, without artificially delaying jobs, the utilization does not vary much. This result is interesting and has not been published yet.



(a) AWRT versus AWRT1.



(b) AWRT1 versus AWRT2.



(c) UTIL versus AWRT.

Figure 8.3: Reduced CTC workload trace.

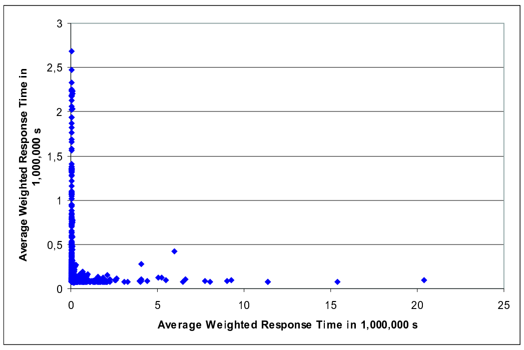
In Table 8.1 the number of non-dominated solutions in the original approximations of the

Pareto fronts and the corresponding reduced sets are presented. The table shows that only for the KTH and the SDSC00 workloads more than 10 % of the non-dominated solutions are removed. In both cases, the number of original solutions is smaller compared to all other workloads and hence the removal of some hundred jobs has a higher effect. The Pareto fronts with the highest numbers of solutions (CTC, SDSC96) are nearly unchanged.

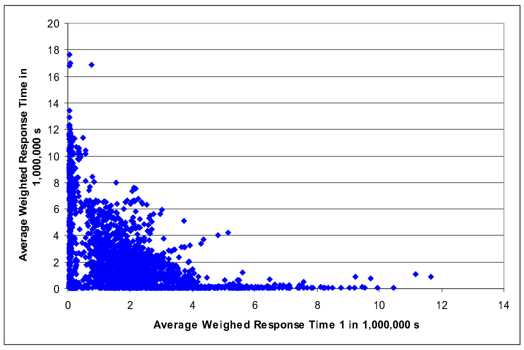
Workload trace	Original number of non-dominated solutions	Number of reduced non-dominated solutions	Ratio of the number of reduced to the original number of non-dominated solutions in %
CTC	13954	13414	96.1
KTH	2536	1995	78.7
LANL	8985	8445	94.0
SDSC00	3123	2584	82.7
SDSC95	8280	7729	93.3
SDSC96	11396	11034	96.8

Table 8.1: Numbers of non-dominated solutions in the original and the reduced Pareto fronts.

Figures 8.4a and 8.4b present the reduced Pareto front for the KTH and SDSC00 workloads. Both figures significantly differ from the corresponding Figure 8.3b of the CTC workload trace. These differences represent the different variation possibilities of the various workloads. In the following, we generate a new representative Pareto front by transforming all reduced Pareto fronts. To this end, we use exactly the same Evolution Strategy and all settings as presented in Section 8.1.4.



(a) KTH workload.



(b) SDSC00 workload

Figure 8.4: AWRT1 and AWRT2 of the reduced workload traces.

8.1.7 Results of the Transformation of the Reduced Pareto Fronts

The quality of the transformations of the reduced Pareto fronts to form a representative Pareto front is also not satisfying. Exemplarily, we present the AWRT1 and AWRT2 in Figure 8.5.

Again, at least two Pareto fronts can clearly be seen. Furthermore, the resulting front still consists of solutions with negative response times. Appendix E on page 195 provides all diagrams for the generated front.

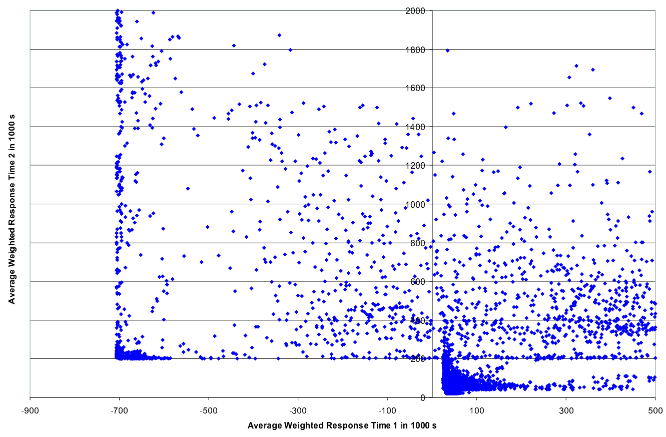


Figure 8.5: AWRT1 and AWRT2 of the resulting combined Pareto front after transforming all reduced workload traces.

The generation of a representative Pareto front based on all Pareto fronts of the available workloads was not successful. Hence, we could not follow the idea of specifying the machine provider's preferences by selecting certain regions within this representative front. Therefore, in the following we only use the Pareto front of a single workload trace to specify the providers preferences and apply the resulting scheduling strategy to all other workloads in order to analyze the robustness of the generated scheduling strategy.

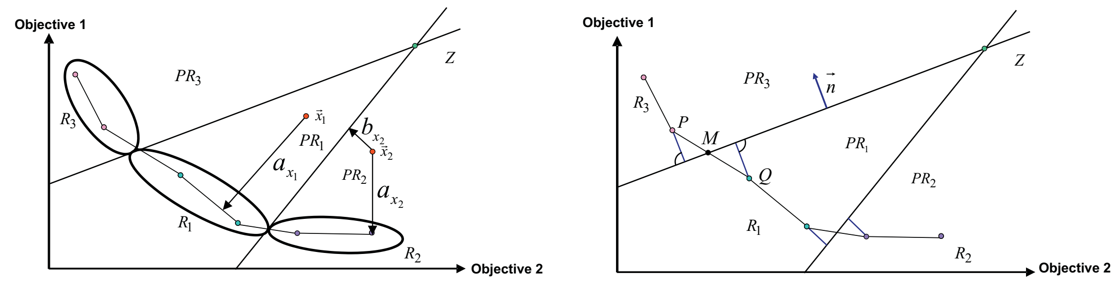
8.2 Generation of a Provider Preferences Function

The first approach to generate a robust scheduling system, as described at the beginning of this chapter, cannot be used due to the missing representative Pareto front.
Therefore, we apply the second approach that was presented at the beginning of Chapter 8. Within this approach, one of the generated Pareto fronts is used to define the provider's preferences. Then, a scheduling strategy is developed in order to optimize the given preferences. At the end, the generated algorithm is applied to other workloads and the results are evaluated in the sense that we test whether the generated strategy produces similar scheduling results.
Thus, we need to define the provider's preferences first. To this end, we use two different approaches. In the first approach the providers select preferred regions within the multidimensional objective space. This is done graphically by selecting the non-dominated solutions that the corresponding scheduling strategy should reach. Several regions can be defined to express the different levels of provider preferences. Based on those regions, we build a fitness function that can be used to determine the quality of each possible schedule during the later scheduling strategy development process. To this end, the distances of the schedule to the Pareto front and to the region with the highest rank are used.
The second approach to express the provider's preferences uses a simple linear function based on the 7 objectives. Note that the Pareto front and the scheduling solutions generated by the

commonly used standard strategies FCFS, CONS, and EASY are still needed. The relationship between the already achievable scheduling solutions and the non-dominated solutions are needed to guide the providers as they are normally not aware of the possible improvements of the different simple objectives. In the following, both attempts to generate a complex objective function to express the provider's preferences are explained.

8.2.1 Region Based Specification

This approach to describe the preferences of the machine providers is based on the selection of regions within the multidimensional solution space. To this end, the reduced Pareto front is visualized with the scatter plot matrix method, see Wong and Bergeron [165]. Within this method, the machine provider is able to see all combinations of two-dimensional plots, in our case 21 of such plots ($\frac{7 \cdot 7 - 7}{2} = 21$). Then, a tool enables the provider to select convex sets within those plots. An arbitrary priority is assigned to each selected convex set. The most preferred region must have the priority equal to 1. The set of all priority regions represents the ordered levels of preference of the machine provider. Furthermore, the system is not restricted to a constant number of such priority regions. The selected regions of non-dominated solutions must represent convex sets as this enables the generation of unique region definitions. This is necessary to easily determine which region a given schedule belongs to. The constraints for the regions are that a region with a lower priority is not placed within a region with a higher priority. Contrary, a region with a higher priority can be included within a region of a lower priority. These constraints do not limit the usability for practical applications. From a practical point of view, schedules with a lower priority are not inside a set of schedules with a higher priority. However, schedules with a higher priority might be surrounded by solutions with a lower priority.



(a) Definition of preferences with regions and the corresponding fitness function definition.

(b) Generation of a hyperplane (bisector).

Figure 8.6: Region concept and definition of bisectors.

Figure 8.6a shows such a possible definition of three priority regions. During the generation of an optimized scheduling strategy, for each possible schedule we need to decide the corresponding region. Hence, we extend the region definition from the Pareto front to the whole objective space. The regions are established by choosing a point Z and then bounding the space that includes the selected non-dominated solutions of the Pareto front with the corresponding priority and this defined point Z. The concept is depicted in Figure 8.6a. In this example, both objectives should be minimized. However, if this is not possible, the minimization of objective 1 is of higher importance than the minimization of objective 2. With this

general concept, it is possible to assign a rank to each schedule within the multidimensional space. Furthermore, the distance to the Pareto front and the distance to all other regions can be calculated. This information is later used to define a fitness function for the scheduling strategy development. In Figure 8.6a the distance of the solution x_2 to the Pareto front is denoted by a_{x_2} and the distance to priority region 1 with b_{x_2}.
Next, we describe the mathematical background for the generation of the regions and also for the concept to determine the distances of a possible schedule to the Pareto front and to the region with rank 1.
After the machine providers have selected convex regions within the space of non-dominated scheduling solutions and assigned a unique priority to each of these regions the above mentioned point Z needs to be defined, see Figure 8.6a. Here, the only condition is that for all coordinates of Z the values are higher than the highest corresponding values of all non-dominated solutions. Then the whole space of solutions is divided into regions by using hyperplanes. Those hyperplanes are constructed to include the point Z. Furthermore, each hyperplane divides schedules of two different regions. So, in order to have a unique division of the whole space, several of such hyperplanes are necessary to distinguish between the existing priority regions. As the whole space consists of seven dimensions, the hyperplanes have six dimensions. Each hyperplane can be represented in a normal form as described in Equation 8.18. Here, the vector $\vec{n}$ is the normal vector of the corresponding hyperplane. $\vec{x}$ describes all points on this hyperplane that satisfy Equation 8.18. The value of d represents the distance of the hyperplane to the point of origin.

$$\vec{n} * \vec{x} = d \tag{8.18}$$

The generation of a hyperplane is depicted in Figure 8.6b. We assume two points P and Q from different priority regions with the corresponding vectors $\vec{p}$ and $\vec{q}$. Furthermore, the point Z with vector $\vec{z}$ is defined. Now, we generate a bisector that represents the hyperplane. First, we define the point M with the corresponding vector $\vec{m}$ that is in the middle between the points Q and P. The construction of the hyperplane is expressed in Equations 8.19 to 8.21. The final hyperplane is described by Equation 8.22.

$$\vec{m} = \frac{\vec{p}+\vec{q}}{2} \tag{8.19}$$

$$\vec{r} = \frac{\vec{p}+\vec{q}-2\cdot\vec{z}}{2} \tag{8.20}$$

$$\vec{n} = \frac{\vec{q}-\vec{z}-(((\vec{q}-\vec{z})*\vec{r})*\vec{r})}{\|\vec{q}-\vec{z}-(((\vec{q}-\vec{z})*\vec{r})*\vec{r})\|} \tag{8.21}$$

$$\vec{n}*\vec{x} = \vec{n}*\vec{m} \tag{8.22}$$

The advantage of the representation of the hyperplanes by a normal form (see Equation 8.22) is the possibility to easily determine on which side of the hyperplane a given point is located. This is later used to determine the priority region of a given schedule. The signed distance D of a point $\vec{x}$ within the multidimensional space can be calculated by: $D = \vec{n} * \vec{x} - \vec{n} * \vec{m}$.
In detail, the preference regions are established by generating all bisectors between schedules within the region and schedules of other regions. Then, we remove all bisectors that divide schedules of the same priority region. This leads to the final set of bisectors that describe the specific preference region.

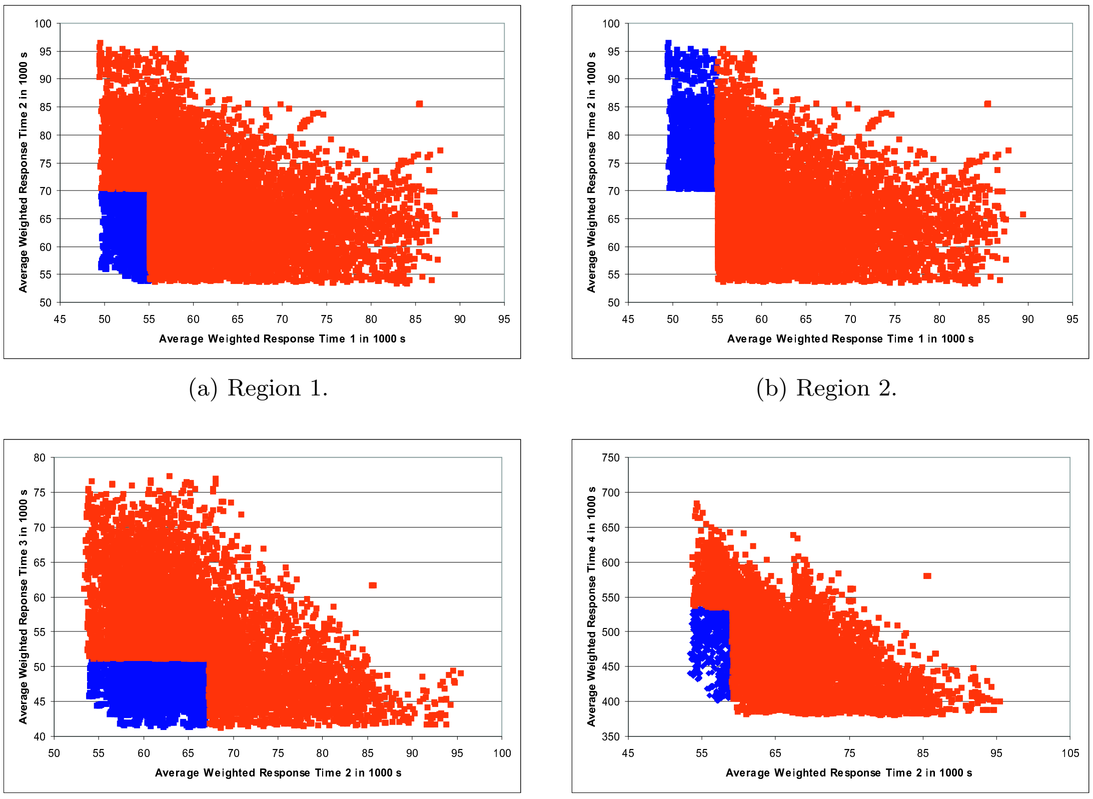
This definition and generation of priority regions ensures that for all possible schedules a unique priority can be assigned. To this end, it is only necessary to test whether the given schedule is within a region by using the distance definition of all corresponding hyperplanes. If the sign of the distance for all hyperplanes is equal to the sign for a schedule of this priority region, the given schedule also belongs to this priority region. Hence, the priority is determined by testing a given schedule within the region with the highest rank, then the next rank and so on, until all conditions are satisfied.
The distance of an arbitrary scheduling solution to a special priority region is determined by the distance to the nearest hyperplane of this selected region. The distance of a scheduling solution to the Pareto front is calculated by using the distance to the hyperplane that includes the seven nearest solutions of the Pareto front.

8.2.1.1 Exemplary Region Selection

After the presentation of the concept to generate priority regions, we present one possible definition of such priority regions. Note that this is only an example and each machine provider is able to define his own preferences individually. In this example, we use the generated Pareto front of the CTC workload trace.
In Figure 8.7a, the priority region 1 is shown. Here, we assume that the power user groups 1 and 2 are the most important groups for a machine provider. Furthermore, user group 1 has a higher priority than user group 2.
Figure 8.7b presents the definition of priority region 2. Here, the priority region 1 is already removed from the two dimensional plot, as the corresponding schedules have already been marked with a priority. Priority region 2 specifies that solutions with a very low average weighted response time for user group 1 are superior to other solutions if region 1 cannot be reached.
The third priority region is defined in Figure 8.7c. Again, all schedules belonging to the higher priority regions are already removed from this two dimensional representation. The definition of priority region 3 specifies that if the average weighted response time of user group 1 cannot be reached (priority regions 1 and 2) the average weighted response times of user groups 2 and 3 should be minimized with a little higher priority on user group 3.
The last priority region that we define is drawn in Figure 8.7d. Within this two dimensional plot, the schedules belonging to the higher priority regions are again removed. If those regions cannot be reached, schedules that optimize the average weighted response times of user groups 2 and 4 are preferred. As we do not specify more than 4 priority regions explicitly, all remaining schedules of the Pareto front belong to the last priority region, in this case priority region 5.

8.2.1.2 Generating an Evaluation Function for Each Possible Schedule

Those priority regions only define the different preferences of the machine provider without providing a mathematical function that directly assigns a scalar objective value. To this end, we use three kinds of information of a possible schedule S: the priority region it belongs to $pr(S)$, the distance to the Pareto front $a(S)$, and the distance to the priority region 1 $b(S)$. As mentioned above, the determination of the priority region can easily be done by using the relative location of given solutions to the hyperplanes that define the various regions. The distance to the priority region 1 can be calculated by the minimum distance to a corresponding



(a) Region 1. (b) Region 2.

(c) Region 3. (d) Region 4.

Figure 8.7: Selection of priority regions from the CTC workload.

hyperplane of region 1. The distance of a schedule to the Pareto front is determined by calculating the distance to the hyperplane that is generated by using the seven nearest points of the Pareto front. Figure 8.6a includes those distances already. In general, every possible function based on the described values could be used instead. The resulting objective function $f(S)$ for schedule S that is exemplarily used in our research work is given in Equation 8.23. Here, the distance to priority region 1 is multiplied by the priority number $pr(S)$ that S belongs to. Moreover, the distance to the Pareto front is added. The defined function $f(S)$ is only an example and can be specified by each machine provider individually.

$$f(S) = pr(S) \cdot b(S) + a(S) \tag{8.23}$$

In Chapter 10, we will demonstrate that online scheduling strategies are able to generate schedules that are very close to the Pareto front. This proves that the selection strategies presented in this section are reasonable. Thus, the resulting evaluation objectives are usable in practical scenarios.

8.2.2 Linear Objective Functions

The definition of sets of convex priority solutions and the following establishment of priority regions is computationally time consuming. Further, the machine provider needs some

experience during the definitions of the priority regions. Moreover, the individual schedule evaluation is time consuming, as the distances to the Pareto front and priority region 1 must be computed.
Hence, we use a second approach by establishing a linear objective function. To this end, the machine provider must be aware of the Pareto front and the scheduling results achieved by using the standard algorithms, like FCFS or EASY. Only this relation between possible schedules and current schedules helps the providers to identify their goals.

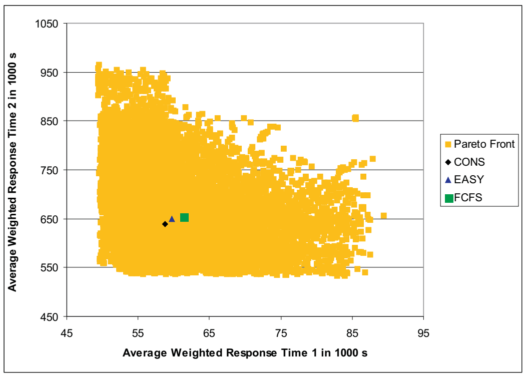
8.2.2.1 Relation between achievable Scheduling Solutions and Solutions generated by Standard Strategies

The above mentioned relative location of the standard algorithms to the non-dominated schedules is depicted in Figures 8.8a to 8.8c for the AWRT1 and AWRT2 of the CTC, KTH, and SDSC96 workload traces. The user groups 1 and 2 were selected as the previously defined priority region 1 includes schedules that minimize the average weighted response times for both user groups. As drawn in Figure 8.8a the schedule generated by using CONS works best in the case of the CTC workload. For this workload and all other workloads the FCFS generated schedules are of lowest quality regarding this combined minimization objective. This behavior is especially true for the KTH workload trace, see Figure 8.8b. In this case as well as for the SDSC96 workload trace, EASY and CONS produce nearly equal schedules regarding this objective.
In the case of the CTC workload, the machine provider might aim to lower both average weighted response times of the user groups 1 and 2 compared with the standard algorithms. However, for the KTH only the average weighted response time of user group 1 can be lowered significantly whereas the response time of user group 2 is already low. The most difficult decision problem arises in the case of the SDSC96 workload trace. In this case, it is only possible to decrease one of both objectives, while the other objective would increase in the same moment.

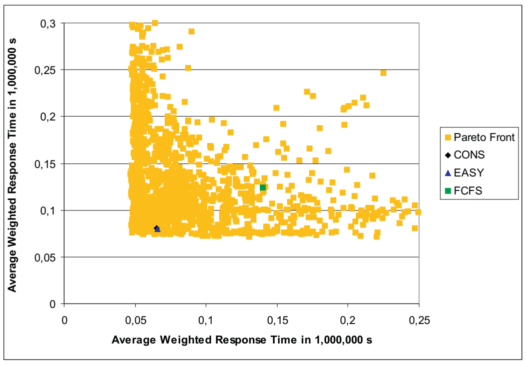
8.2.2.2 Choosing an Exemplarily Linear Function

The objective function that we use for our research should be similar to the objective function that is based on the priority region definition. Hence, we focus on the main goal of minimizing the average weighted response times of user groups 1 and 2. Thus, we choose to use the objective function defined in Equation 8.24. This function aims to minimize the response times of both specified user groups by having a higher preference on user group 1. This includes the previously defined region 2. The previously defined priority regions 3, 4, and 5 are not included, as the weights for these regions are very hard to define.

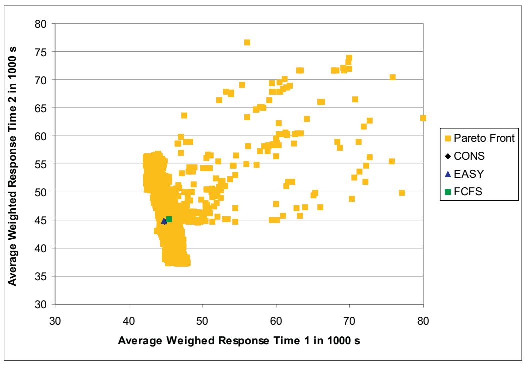
$$f(S) = 10 \cdot \mathrm{AWRT1} + 4 \cdot \mathrm{AWRT2} \tag{8.24}$$



(a) CTC workload trace.



(b) KTH workload trace.



(c) SDSC96 workload trace.

Figure 8.8: AWRT1 and AWRT2 for the three different Pareto fronts with the results of the corresponding standard algorithms.

Chapter 9

Generating Appropriate Scheduling Strategies

This chapter describes the different scheduling strategies that we use to optimize the previously defined complex evaluation function. During this optimization process, we test possible evaluation function definitions as well as the region based definition presented in Section 8.2.1 and the definition of a linear objective function as introduced in Section 8.2.2. Again, these functions are arbitrarily chosen and hence just examples for the feasibility of our approach. The usage of the two different approaches results in two different solution techniques as introduced in Section 2.5.1. The region based objective function represents the problem

$$P_m|\bar{r}_j, \bar{p}_j, m_j|F_{Tp}(\text{AWRT, AWRT1, AWRT2, AWRT3, AWRT4, AWRT5, UTIL}), \quad (9.1)$$

whereas the linear function corresponds to

$$P_m|\bar{r}_j, \bar{p}_j, m_j|F_l(\text{AWRT, AWRT1, AWRT2, AWRT3, AWRT4, AWRT5, UTIL}). \quad (9.2)$$

In general, our goal is to develop a robust scheduling strategy. Unfortunately, it was not possible to generate a representative Pareto front for all available workload traces. Thus, we cannot use the method presented in Section 6.2.1. Nevertheless, in the following we explain two different approaches to derive robust scheduling strategies.

In the first approach, a scheduling strategy is optimized by simulating and evaluating the same strategy for all workload traces. Then, the results are combined in order to represent the overall objective. Therefore, the resulting strategy is expected to be robust as it should work well for most of the workload traces. In the following, we will refer to this approach as *ALLOPT*.

The second approach optimizes a scheduling strategy for a single workload trace. Then, this strategy is applied to all other workload traces. If the outcome is similar in the sense that the 7 simple objectives and the complex objective function value are close to each other for all or at least most of the workload traces, the strategy is also called robust. This approach of using a *$\underline{s}$ingle* workload for the real optimization is referred as *SOPT*.

For the development of our scheduling strategies, we use on the one hand an adaptation of a Greedy scheduling strategy and on the other hand the development of rule based scheduling strategies. Both attempts are described in the following.

9.1 Adaptation of a Greedy Scheduling Strategy

A Greedy scheduling strategy, as described in Section 4.4.3, is adapted to our problem in order to maximize the providers' preferences. In Section 6.3.1, we already introduced the main concept and advantages of this strategy.

As described above, the Greedy strategy is based on a complex sorting criterion that calculates job weights for all jobs within the waiting queue. These weights are used to reorder all jobs within this queue whenever new jobs are submitted or running jobs finish execution. Then, the jobs in the given order are scheduled using FCFS.

The sorting criterion is only based on static job parameters, like the number of requested nodes, the estimated runtime given by the users, and the current time. Furthermore, the users cannot specify a job weight and hence not influence the positions of their jobs within the waiting queue. Further, the parameters of the static sorting criterion can be specified by the machine providers individually. Hence, the resulting Greedy strategy can be adapted to individual preferences. Furthermore, the overall strategy is simple and easy to implement.

In detail, we use the same procedure as presented in Section 7.3. However, this time we do not aim at generating a Pareto front but at optimizing an individual objective. Thus, we apply a single-objective Evolutionary Algorithm. Using this strategy, we again divide all system states into three classes:

1. weekends,
2. weekdays between 8 am and 6 pm, and
3. weekdays between 6 pm and 8 am.

For each of these classes we assign one static sorting criterion. Again, we use the same criteria as in Section 7.3, namely the criteria 7.2 to 7.5 (on page 76).

Our generation of an adapted Greedy strategy uses all combinations of those four sorting criteria to the three system states. Hence, we optimize twelve different systems and finally select the system that performs best.

For each Greedy strategy, only 36 parameters need to be determined in our scenario, as described in Section 7.3.

9.1.1 Parameter Optimization

For the determination of the 36 parameters we use a standard $(\mu, \kappa, \lambda, \rho)$-Evolution Strategy with a real-value coding as described in Section 3.1. For each optimization, 100 generations are used. Further, we set $\kappa = 100$, $\mu = 15$, and $\lambda = 105$ which fits to Schwefel's recommendation of $\mu/\lambda = 1/7$ [136].

The values of the 36 parameters within the criteria definitions 7.2 to 7.5 are limited in order to reduce the search space. As during the generation of the Pareto front, see Section 7.3, we again use:
$w_i \in [0, 1]$, $a \in [0, 1]$, $b \in [0, 1]$, and $K_i \in [0, 5]$.

Our Evolution Strategy uses a mutation strength for each objective parameter. The resulting mutation strength vector is adapted as described in Section 3.1.5.2.2. To this end, the two learning rates are defined as:

$$\tau_0 = \frac{1}{\sqrt{2 \cdot 36}}, \text{ and} \tag{9.3}$$

$$\tau_1 = \frac{1}{\sqrt{2\sqrt{36}}}. \tag{9.4}$$

The object parameters are recombined by using a discrete recombination whereas an intermediate recombination is applied for the strategy parameters, see Section 3.1.4. In both cases, we use $\rho = \mu$.
Within the next section, we describe our second approach to generate appropriate scheduling strategies.

9.2 Rule Based Scheduling Systems

In our second approach to generate scheduling strategies, we use rule based scheduling systems as already introduced in Section 6.3.2.
So far, the use of rule based systems in scheduling environments is rare. Nevertheless, first attempts [19, 48] have shown the feasibility of such an approach. However, those scheduling systems are all based on single objective evaluation functions. Furthermore, the quality of these first attempts is not further optimized regarding the given objective. Those works only prove the applicability of rule based scheduling systems.
A rule based scheduling strategy is divided into two steps. In the first step, a sorting criterion is used to determine the sequence of jobs within the waiting queue. In the second step, a scheduling algorithm assigns idle machines to the jobs within the waiting queue in the given job order. In the following, we use the term *strategy* to describe the whole scheduling process that consists of both steps. An *algorithm* only describes the procedure of the second step that uses the already sorted waiting queue.
As introduced in Section 6.3.2, the system needs to classify all possible scheduling states in order to assign the corresponding sorting criterion and the scheduling algorithm to each state. Each of those states is described by the actual schedule, the current waiting queue, and the results achieved so far. Then, the rule based system can be established. In this case, each rule describes in its *conditional* part the system state in which the rule is *active* and within the *consequence* part the recommendations for the sorting criterion and the scheduling algorithm. Note that the rule base consists of several rules that might influence the outcome of the whole rule base. Hence, we call the outcome of a single rule a *recommendation* for the overall output which is equal to the consequence part of this single rule. The output of the whole rule base is generated by combining all recommendation of the participating rules.
As stated in Section 6.3.2, local scheduling decisions influence the allocation of future jobs so that the effect of a single decision cannot be determined individually. Therefore, the whole rule base is only evaluated after the completed scheduling of all jobs belonging to a workload trace. This has a significant influence on the learning method to generate this rule base as the evaluation prevents the application of a supervised learning algorithm. Instead, the reward of a decision is delayed and determined by a critic. Furthermore, the generation of an appropriate scheduling system state classification is not known in advance and has to be generated implicitly during the generation of the rule base scheduling system.
Before the concept of rule based scheduling is introduced in more detail, we specify the features that are used.

9.2.1 Scheduling Features

Here, we introduce several features to classify possible scheduling states within our rule based scheduling system. In this work, we restrict ourselves to use seven features. On the one hand, this enables us to describe the current schedule, the actual waiting queue, and the already achieved results. On the other hand, the classification of system states by seven features can still be developed in a reasonable time. Our feature selection only serves the purpose to illustrate our methodology. Note that objectives evaluate the whole scheduling process at the end of a simulation while features only describe the current state of the system.
In the remainder of this chapter, we use the following notations. Within our job system τ at time t we refer to the set of already finished jobs as $\xi(t)$, to the set of running jobs as $\pi(t)$, and to the set of jobs within the actual waiting queue as $\nu(t)$. Further, we use $\varrho_i(j)$ as defined in Equation 6.3 to specify that job j belongs to user group i. Additionally, we use the resource consumption RC_j of job j as defined in Equation 5.1.
The first feature that we use in our work is the *Average Weighted Slowdown* (SD) over all jobs that have completed their processing. The *Slowdown* (SD_j) of a single job j within schedule S is defined as:

$$SD_j = \frac{C_j(S) - r_j}{p_j}. \tag{9.5}$$

SD_j reaches its minimum value of 1 if job j does not wait before it starts execution. Then, the release date is identical with the job's start time. Normally, the range of this feature can be delimited to the interval of [1,100] as values greater than 10 occur very rarely in practice. The feature *Average Weighted Slowdown* (SD) for all already processed jobs $j \in \xi(t)$ uses the same weighting as defined for the AWRT, see Equation 4.2, for the same reason.

$$SD = \frac{\sum\limits_{j\in\xi(t)} RC_j \cdot (C_j(S) - r_j)}{\sum\limits_{j\in\xi(t)} RC_j \cdot p_j} \tag{9.6}$$

This feature represents the scheduling decisions in the past as only jobs that are already finished are used to calculate this feature. Here, we have not limited the window for the slowdown. In practical cases, a limitation to, for instance, the last month or the last week may be appropriate.
The *Momentary Utilization* (U_m) of the whole parallel computer at time t represents the second feature that is used within this work. To this end, we determine the number of machines that are allocated to all currently running jobs $j \in \pi(t)$.

$$U_m = \frac{\sum\limits_{j\in\pi(t)} m_j}{m} \tag{9.7}$$

In addition, we use five features that refer to the waiting jobs $j \in \nu(t)$ for each user group separately. These features - called *Proportional Resource Consumption of Waiting Queue for User Group i* ($PRCWQ_i$) - represent the percentage of the estimated resource consumption of the waiting jobs of the specified user group in relation to the estimated resource consumption of all waiting jobs.

$$PRCWQ_i = \frac{\sum\limits_{j \in \nu(t)} \bar{p}_j \cdot m_j \cdot \varrho_i(j)}{\sum\limits_{j \in \nu(t)} \bar{p}_j \cdot m_j} \tag{9.8}$$

The values ($PRCWQ_i$) for the different user groups enable the rule based scheduling systems to change their behavior according to the foreseeable resource demand of specific user groups. Note that we use the estimated processing time $\bar{p}_j$ for weighting as the real processing time p_j is unknown before the jobs are completed.

9.2.2 Detailed Rule Based Scheduling Concept

A rule based scheduling approach must assign a scheduling strategy to every possible system state. A complete rule base RB consists of a set of rules R_i. Each rule R_i contains a conditional and a consequence part Ω_i. The conditional part describes the conditions for the activation of a rule using the defined features. The consequence part represents the corresponding scheduling strategy recommendation.
In order to specify all scheduling states in an appropriate fashion each rule defines certain partitions of feature space within the conditional part. The rule base system must contain at least one activated rule for each possible system state.
As mentioned above, the scheduling strategy specifies

1. a *sorting criterion* for the waiting queue $\nu(t)$ and
2. a *scheduling algorithm* that uses the order of $\nu(t)$ to schedule one or more jobs.

First, the chosen sorting criterion determines the sequence of jobs within the waiting queue. Second, the selected scheduling algorithm finds a processor allocation for at least one job of the sorted waiting queue. We have chosen four different sorting criteria:

- *Increasing Number of Requested Processors:* Preference of jobs with little parallelism and therefore higher utilization. This sorting provides the potential gain of being able to insert many jobs into the current schedule as jobs with a smaller amount of requested processors are often easier to schedule.
- *Increasing Estimated Run Time:* Preference of short jobs and therefore higher job throughput. This sorting criterion potentially leads to a higher job throughput.
- *Decreasing Waiting Time:* Preference of long waiting jobs and therefore more fairness. This sorting criterion provides a higher fairness as the jobs are processed according to their submission. Jobs with a higher waiting time are selected first.
- *Decreasing User Group Priority:* Preference of jobs from users with a higher priority. The sorting by user groups provides a higher ranking for all jobs of users with a higher priority according to their user group assignment. This criterion reflects our objective function.

The selected scheduling algorithm is one of the four methods presented in Section 4.4. Note that Greedy is already a complete scheduling strategy while the other described scheduling algorithms (FCFS, EASY, CONS) need a sorting criterion of $\nu(t)$ in addition. Again, the set of scheduling algorithms can be extended for other rule base systems.

The general concept of the rule based scheduling approach is depicted in Figure 9.1. As 4 different sorting criteria with 3 possible scheduling algorithms and the combined Greedy strategy are available, we have to choose one of 13 strategies for each possible system state. However, it is not practicable to test all possible assignments in all possible states. For example, let us assume a very coarse division of each feature into only 2 partitions. Then 13 possible strategies and 7 features result in $13^{2^7} \approx 3.84 \cdot 10^{142}$ simulations if all combinations in all possible system states are evaluated. Additional problems occur during the generation of a rule based scheduling system as the number and reasonable partitions of features that are required to describe the system states in an appropriate way are generally unknown in advance.

Hence, we introduce two possible approaches to derive a rule based scheduling system using only a limited number of simulations. These approaches are *rigid rule based scheduling systems* and *Genetic Fuzzy systems*. In the following, both possible approaches are explained in detail.

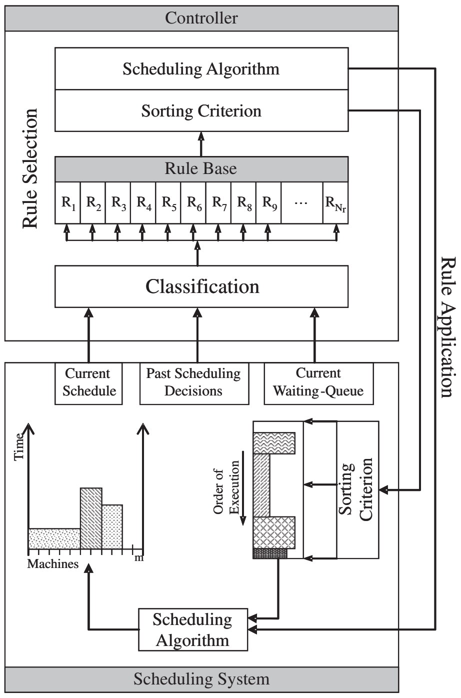


Figure 9.1: General concept of the rule based scheduling approach.

9.2.3 Rigid Rule Based Scheduling Systems

In this work, we use a rigid rule based scheduling system that is developed in two different ways. Such a rigid system uses $N_F = 7$ features with a fixed number of intervals for each feature ω. That is, each feature ω has $N_{p,\omega} - 1$ static bounds, which divide the possible value range of ω into $N_{p,\omega}$ partitions. The static bounds are specified before the assignment of sorting criteria and scheduling algorithms to the various system states is extracted. This concept of such fixed partitions is shown in Figure 9.2.

Generally, a larger number of partitions $N_{p,\omega}$ of a feature ω potentially leads to a more accurate rule set while more system state classes must be optimized. Overall, this results in

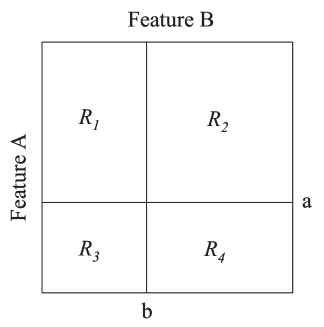


Figure 9.2: Example partitioning of the feature space and the resulting set of rules $R_1 \dots R_4$.

N_r system state classes that must be provided to cover all possible system states with

$$N_r = \prod_{\omega=1}^{N_F} N_{p,\omega}. \tag{9.9}$$

The rigid rule based system described above activates only a single rule in any system state. Hence, the output recommendation of this single activated rule is the output of the whole scheduling system.
The number of reasonable partitions of the different features is not known. Thus, we have evaluated several settings and finally used a single division of the intervals of SD and $PRCWQ_1$ to $PRCWQ_5$ respectively. This leads to two partitions in each case. Further, we found that using three partitions for the U_m feature leads to good rule based scheduling systems. Overall, this produces $2^6 \cdot 3 = 192$ different system state classes that are needed to build a complete rule base.
Furthermore, we have evaluated different division values for the system state class features, see Franke et al. [63]. The partitions which achieved the best results are used for the rigid rule base system development, see Table 9.1.

Feature	Intervals
SD	[1-2],]2-100]
$U_{m[\%]}$	[0-75],]75-85],]85-100]
$PRCWQ_{1[\%]}$	[0-20],]20-100]
$PRCWQ_{2[\%]}$	[0-20],]20-100]
$PRCWQ_{3[\%]}$	[0-25],]25-100]
$PRCWQ_{4[\%]}$	[0-25],]25-100]
$PRCWQ_{5[\%]}$	[0-25],]25-100]

Table 9.1: Feature partitions for the rigid rule based scheduling systems.

A rigid rule based scheduling system has the advantage of a simple implementation and generates rule bases that can easily be interpreted by providers. Future scheduling development may benefit from knowledge gained through this kind of interpretation. The selected scheduling algorithms and sorting criteria for certain system states can directly be extracted from the corresponding rules without further computation.
Unfortunately, the fixed division of the whole feature space has a critical influence on the performance of the scheduling system. At the moment, no mechanism is provided that auto-

matically adjusts the defined partitions. In addition, a reduction of the number of features is not yet considered, which means that all rules must contain and specify all features.
As mentioned above, we use two different methods to establish the assignment of sorting criteria and scheduling algorithms to system state classes. Both will be explained in the following.

9.2.3.1 Iterative Rule Base Development

The first rigid attempt, the iterative rule base generation, is based on a sequential optimization of the output behavior assignment to a pre-defined system state class.
Algorithm 9.1 describes the assignment of sorting criteria and scheduling algorithms to those classes.

Algorithm 9.1 Iterative rule base optimization.

```
Assign a standard strategy to all $N_r$ rules $R_1, \dots, R_{N_r}$ of rule base $RB$;
for ($i = 1$ to $N_r$) do
  $f_{old}(R_i) = \infty$;
  for ($s = 1$ to 13) do
    Use strategy $s$ as the consequence part of rule $R_i$ and run a simulation with the result $f_{new}(R_i)$;
    if ($f_{new}(R_i) < f_{old}(R_i)$) then
      Assign strategy $s$ to rule $R_i$;
      $f_{old}(R_i) = f_{new}(R_i)$;
    end if
  end for
end for
```

Here, the best scheduling strategy is determined for the first system state class by using a standard strategy for all other system state classes. Then, the scheduling strategy for the second system state class is extracted by using the strategy that has been determined for system state class one and the standard strategy for all other classes. Then, the process optimizes all following system state classes with the same procedure.
In our work, we used FCFS together with the sorting by waiting time as the standard scheduling strategy in Algorithm 9.1. This strategy is used at many real installations.
The iterative rule base development needs $192 \cdot 13 = 2496$ simulations for a single workload trace in order to determine the final rule base assuming that we use the introduced simple partitioning of the feature space into 192 system state classes.
This approach optimizes each system state class separately with limited influence on other allocations. Thus, it can be assumed that the resulting rule base can still be improved by considering the cooperation aspects of different rules. Hence, the next section describes our second approach for a rigid rule based scheduling system that includes such a cooperation aspect.

9.2.3.2 Probability Driven Rule Base Development

With the probability driven rule base development, we try to avoid the excessive amount of simulations that need to be performed in order to generate the rule base. Further, this

approach puts the emphasize on the cooperation aspect of the rules within the final rule base as no standard scheduling strategy is used and the evaluation of the assignment of a special scheduling strategy to the consequence part of a rule is based on several simulations with varying strategy assignments for all other rules.
Rule bases are generated by randomly assigning potential scheduling strategies to rules so that each scheduling strategy is assigned to each rule the same number of times. Hence, not all rule bases are generated completely at random. Remember that the conditional part is rigid and does not vary. Thus, a single rule completely describes a single system state class.

Algorithm 9.2 Probability driven rule base generation.

```
for all (rules $R_i \in \{1, \dots, N_r\}$) do
  for all (scheduling strategies $\Omega_i \in \{1, \dots, 13\}$) do
    $pa(R_i, \Omega_i) = p$; {set the number of possible assignments $pa(R_i, \Omega_i)$ of strategy $\Omega_i$ to
    rule $R_i$ to the fixed number $p > 0$.}
    $f_{R_i,\Omega_i} = 0$;
  end for
end for
while ($\exists$ $(R_i, \Omega_i)$ with $pa(R_i, \Omega_i) > 0$) do
  for all (rules $R_i \in \{1, \dots, N_r\}$) do
    randomly select a strategy $\Omega_i$ with $pa(R_i, \Omega_i) > 0$ and assign this $\Omega_i$ to $R_i$;
    $pa(R_i, \Omega_i) = pa(R_i, \Omega_i) - 1$;
  end for
  evaluate the resulting rule base $RB$ by generating a schedule with a given workload trace
  with the resulting objective value $f_{obj}$;
  for all ($R_i, \Omega_i \in RB$) do
    $f_{R_i,\Omega_i} = f_{R_i,\Omega_i} + f_{obj}$;
  end for
end while
for all (rules $R_i \in \{1, \dots, N_r\}$) do
  determine final strategy $\Omega_i = \arg \min_{\Omega_j \in \{1,\dots,13\}} f_{R_i,\Omega_j}$ and assign this strategy to Rule $R_i$;
end for
```

Then, we use those rule bases to produce schedules for the given workload data and evaluate those schedules with the help of the complex scheduling objective. Each schedule results in a scalar objective value. The assignment of a special scheduling strategy to a rule is evaluated by adding the scalar objective values of all schedules that were generated using this assignment. Finally, we build the resulting rule base by assigning the scheduling strategies with the smallest sum of the objective values to the individual rules as we assume a minimization of the objective function. With this approach the number of required simulations can be reduced significantly as we only estimate the optimal assignments. In general, the performance can be increased by generating more rule bases. However, the trade-off between a better performance and more required simulations must be kept in mind.
In Algorithm 9.2, the parameter p describes how often a scheduling strategy is assigned to a single rule. This parameter p influences the number of required simulations that is given by the product of the number of possible scheduling strategies (N_Ω) and the parameter p.

Franke et al. [64] have shown that $p = 50$ is a good compromise between the required number of simulations and the scheduling quality. Hence, we used $p = 50$. This, in our case, results in $13 \cdot 50 = 650$ simulations, which is significantly less than the required number of simulations for the iterative rule base generation approach.

9.2.4 Scheduling Strategies based on Genetic Fuzzy Systems

The previously presented rigid scheduling systems have several drawbacks regarding the generation of an appropriate rule based scheduling system. Mainly, the static number of feature partitions and the static pre-defined bounds for these partitions are not flexible enough and may lead to bad scheduling systems. Furthermore, the whole feature space needs to be divided and appropriate scheduling strategies need to be assigned to each individual partition. Hence, the number of rules cannot be varied.

Consequently, we need a method that automatically adjusts the partitions of the feature space and assigns appropriate scheduling strategies to the resulting system state classes in parallel. *Genetic Fuzzy systems*, see Hoffmann [79], provide the capabilities to solve these problems. We therefore introduce this concept in more detail.

Fuzzy systems are usually designed by modeling implicit knowledge of an expert within a set of linguistic variables and Fuzzy rules, see Hoffmann [79]. Such Fuzzy systems have been applied successfully for many real world problems. However, Fuzzy systems themselves do not provide mechanisms to automatically learn from existing data.

The various design concepts of Fuzzy logic controllers often use Evolutionary Algorithms to adjust the membership function as well as to define the output behavior of individual rules, see, for example, Hoffmann [79]. Especially Genetic Fuzzy systems have proven to deal with such classification and automatic rule base generation problems in a suitable way. All those Genetic Fuzzy systems encode either single rules (*Michigan approach*, Bonarini [14]) or complete rule bases (*Pittsburgh approach*, Smith [145]). Both encoding approaches are used in this work and hence are introduced in more detail.

During the evolutionary optimization the Michigan approach uses a set of individuals each representing only a single rule. This set of individuals is evolving during the optimization process. Hence, all rules are in *competition* in order to form next generations. However, they need to *cooperate* as well to establish rule bases that perform well corresponding to the given objective function. Thus, the difficulty of this approach is the cooperation-competition nature of the problem. Individual rules must be selected during the optimization whereas the evaluation of single rules is not possible. Hence, an effective fitness assignment that incorporates these difficulties is needed during the evolution.

In our work, the determination of a Genetic Fuzzy system by using the Michigan approach is realized by a *Symbiotic Evolutionary Algorithm*, as introduced by Juang et al. [88]. In the following, we will use the terms *Michigan approach* and *Symbiotic Evolution* interchangeably as this concept is based on the Michigan approach. It evaluates the different rules together as several rules are needed to compose a complete rule base. In opposite to Jung et al. [88], Hoffmann [79] restricts the competition of rules to subsets that trigger for similar inputs. This restriction influences the selection of rules and also the recombination between them. The outcoming rules provide a higher diversity and therefore further improve the coverage of the possible input values. This often leads to a quicker generation of the final Fuzzy rules. Considering the time consuming simulations, which have to be performed in order to determine the overall performance of a rule based system with respect to the resulting schedule. The

reduction of the number of necessary simulations is of great importance. Hence, we combine the Symbiotic Evolutionary Algorithm with Hoffmann's approach.
The Michigan approach mainly addresses continuous learning in non-inductive problem areas, see Cordón et al. [24]. Further, the Michigan approach is focused on the idea of representing the knowledge of a single entity that adopts through the interaction with its environment.
However, another major goal during the generation of a rule based scheduling system is the ability to reduce the number of necessary rules to a reasonable minimum. In this scenario, the Pittsburgh approach is more appropriate as the number of necessary rules can easily be encoded within an individual. Contrary, rules within the Michigan approach participate in different Fuzzy systems. Thus, a calculation of the combined number of rules would require an additional overhead. The Pittsburgh approach encodes a whole rule base in each individual. Hence, the encoding and also the evaluation of a reasonable number of rules is much easier. Further, the length of individuals can be integrated within the fitness function that is used to evaluated rule bases. Thus, the preference of rule bases with a smaller number of rules can easily be implemented. The main drawback of the Pittsburgh approach in comparison to the Michigan approach is the increasing size of the individuals. This drastically increases the size of the search domain of the evolutionary optimization and may reduce the efficiency of various genetic operators.
During the generation of our Genetic Fuzzy systems we use Evolution Strategies to perform the adaptation of the model parameter. The application of Evolution Strategies is not selected by the majority of research projects in the area of Fuzzy systems. However, we have chosen these strategies as they have proven to outperform Genetic Algorithms when applied to real-value coded problems, see Bäck and Schwefel [7]. This fits to our problem as the underlying scheduling system is real-value coded. Furthermore, Evolution Strategies are more often applied to real-value coded problems whereas Genetic Algorithms usually process a string of discrete, often binary symbols. In addition, Evolution Strategies support self adaptation which makes them often more flexible to explore the structure of the fitness landscape.
Within this work, we restrict ourselves to Genetic Fuzzy systems. However, other approaches, like Neuro-fuzzy systems [79], might also be able to solve the problem at hand. A comprehensive introduction to Genetic Fuzzy systems and all corresponding learning methods is provided by Cordón et al. [24].
Before the different rule base encoding schemes are explained in detail, we introduce the encoding of individual rules and detail the computation of the final Fuzzy controller output.

9.2.4.1 Coding of Fuzzy Rules

In the following, the coding of Fuzzy rules is introduced. Since our approach is derived from the traditional Takagi-Sugeno-Kang (TSK) model, see Takagi et al. [153], we provide an overview of this concept first.

9.2.4.1.1 Takagi-Sugeno-Kang (TSK) Fuzzy Controller The TSK-Fuzzy controllers are well studied and applied in many research projects, see Cordón at al. [24] and Jin et al. [86]. In opposite to conventional controllers that use a single model to describe the whole system, TSK-Fuzzy controllers are built by using several simple submodels that are combined to represent the global behavior of the system.
TSK models typically consist of N_r IF-THEN rules R_i of the form (see Cordón at al. [24, p. 20]):

$$\begin{aligned} R_i: \quad & \text{IF} \quad x_1 \text{ is } g_i^{(1)} \text{ and } x_2 \text{ is } g_i^{(2)} \text{ and ... } x_{N_F} \text{ is } g_i^{(N_F)} \\ & \text{THEN} \quad y_i = b_{i0} + b_{i1}x_1 + \ldots + b_{iN_F}x_{N_F}. \end{aligned}$$

The vector $\vec{x}$ with N_F elements describes the current system state and hence represents the input of the Fuzzy system. The y_i is the output of rule R_i. In general, $g_i^{(\omega)}$ are linguistic labels associated with fuzzy sets that define their meaning, see Cordón at al. [24, p. 4]. Here, $g_i^{(\omega)}$ is the ω-th input Fuzzy set that describes the membership for a feature ω. To this end, a membership function must be defined. The coefficients b_{i0} to b_{iN_F} are constant. The overall output of a TSK-Fuzzy system can then be computed by:

$$y = \frac{\sum_{i=1}^{N_r} \phi_i y_i}{\sum_{i=1}^{N_r} \phi_i} = \frac{\sum_{i=1}^{N_r} \phi_i (b_{i0} + b_{i1}x_1 + \ldots + b_{iN_F}x_{N_F})}{\sum_{i=1}^{N_r} \phi_i}. \tag{9.10}$$

Here, ϕ_i is the degree of membership that is often called *firing strength* of rule R_i and is defined as the product of all membership functions that are associated with the participating Fuzzy sets:

$$\phi_i = g_i^{(1)}(x_1) \wedge g_i^{(2)}(x_2) \wedge \ldots \wedge g_i^{(N_F)}(x_{N_F}) = \prod_{\omega=1}^{N_F} g_i^{(\omega)}(x_\omega). \tag{9.11}$$

As shown, the output y_i of a TSK-Fuzzy controller is a linear combination of the input variables plus a constant value. However, in the context of rule based scheduling systems, only a single scheduling strategy has to be generated as the recommendation of a single rule. Hence, we follow the approach of Juang et al. [88] by modifying the process of generating the recommendation of single rules and also the output of the whole rule base system. The details of our modifications are presented in the remainder of this section. In our system, the recommendation of a single rule does *not* depend on the input parameters but has to be assigned to a rule. Evolutionary Algorithms are used to find suitable assignments of recommendations to single rules.
In the following, we detail our Genetic Fuzzy system. The coding schemes and learning techniques are adapted and modified using the work of Juang et al. [88] and Jin et al. [86].

9.2.4.1.2 Coding of Rules For a single rule R_i, every feature ω of all N_F features is modeled by a Gaussian membership function $(\mu_i^{(\omega)}, \sigma_i^{(\omega)})$-GMF,

$$g_i^{(\omega)}(x) = \frac{1}{\sigma_i^{(\omega)}\sqrt{2\pi}} \exp\left\{ \frac{-(x-\mu_i^{(\omega)})^2}{2\sigma_i^{(\omega)2}} \right\}. \tag{9.12}$$

In Figure 9.3 a sample (5,0.75)-GMF is depicted.
The $\mu_i^{(\omega)}$ value is the center of the feature ω that is covered by the rule R_i. Therefore, this value defines an area of system state in the feature space where the influence of the rule is very high. Note that using a GMF as feature description the condition

$$\int_{-\infty}^{\infty} g_i^{(\omega)}(z)dz = 1 \; \forall \; i \in \{1, \ldots N_r\} \wedge \; \omega \in \{1, \ldots N_F\} \tag{9.13}$$

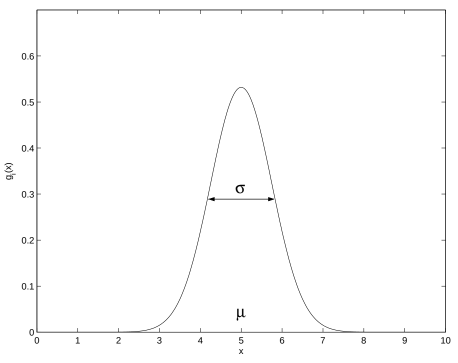


Figure 9.3: Gaussian membership function with $\mu = 5$ and $\sigma = 0.75$.

always holds. In other words, for increasing $\sigma_i^{(\omega)}$ values, the peak value of the GMF decreases because the integral remains constant. Using this property of a GMF we are able to reduce the influence of a rule for a feature ω completely by setting $\sigma_i^{(\omega)}$ to a very high value. For $\sigma_i^{(\omega)} \rightarrow \infty$, a rule has no influence for feature ω anymore. With this approach, it is also possible to establish a kind of default value that is used if no other rules are defined in a feature domain. Based on this feature description, a single rule can be described by

$$R_i = \left\{ g_i^{(1)}(x),\ g_i^{(2)}(x),\ \ldots,\ g_i^{(N_F)}(x), \Omega_i \right\}. \tag{9.14}$$

The consequence of rule R_i is an integer number Ω_i that represents a scheduling strategy. In general, the consequence does not need to be restricted to a single number. For example, it is possible to specify a scheduling strategy which is particularly favorable for a certain system state and another one which is clearly unfavorable. The final output decision is then computed by the superposition of the consequence parts of the involved rules.

The main advantage of using several GMFs for describing a single rule is the automatic coverage of the whole feature space. In contrast to the iterative approach, a rule recommends a scheduling strategy for all possible system states. Hence, it is the focus of this approach to find a meaningful set of N_r rules that generates a good rule base system RB:

$$RB = \{R_1,\ R_2,\ \ldots,\ R_{N_r}\}. \tag{9.15}$$

9.2.4.1.3 Multiple Consequence Parts In order to further support the Fuzzy concept, we decided to use multiple consequence parts. Thus, the consequence part of every rule R_i, $i \in \{1, \ldots, N_r\}$, includes a weighted recommendation for all N_Ω possible outputs Ω_h, with $h \in \{1, \ldots, N_\Omega\}$. The consequence part of rule R_i is therefore described by a vector

$$\vec{\Omega}(R_i) = \left(\begin{array}{c} w_{i1}\ w_{i2} \ldots w_{iN_\Omega} \end{array} \right)^T. \tag{9.16}$$

We restrict the possible weight values w_{ih} of rule R_i to the elements of the set $\{-5,\ -1,\ 0,\ 1,\ 5\}$. The value -5 represents a *particularly unfavorable* scheduling strategy while 5 is a *particularly favorable* one. The other possible weights can be interpreted accordingly. We use a non-linear weight scaling in order to force distinct recommendations. When considering the superposition of those weights similar weights may lead to almost indistinguishable recommendations. Furthermore, we also include 0 as possible weight to express that a rule behaves completely neutral with respect to the recommendation of a scheduling strategy for a given system state. This may also reduce the number of overall rules. A complete single rule R_i with conditional part and consequence part is denoted by

$$\begin{aligned} R_i &= \left\{ g_i^{(1)}(x),\ g_i^{(2)}(x),\ \dots\ g_i^{(N_F)}(x),\ \Omega_1,\ \Omega_2,\ \dots\ \Omega_{N_\Omega} \right\} \\ &= \left\{ g_i^{(1)}(x),\ g_i^{(2)}(x),\ \dots\ g_i^{(N_F)}(x),\ \vec{\Omega}(R_i) \right\}. \end{aligned} \tag{9.17}$$

9.2.4.2 Computation of the Controller Decision

For a given system state, we compute the superposition of the weighted output consequence parts of all rules. The system state is represented by the actual feature vector

$$\vec{x} = \begin{pmatrix} x_1 & x_2 & \dots & x_{N_F} \end{pmatrix}^T \tag{9.18}$$

of N_F feature values. Then we compute the degree of membership $\phi_i(x_\omega) = g_i^{(\omega)}(x_\omega)$ of the ω-th feature of rule R_i for all N_r rules and all N_F features. The multiplicative superposition of all these values as "AND"-operation leads to the total degree of membership

$$\phi_i(\vec{x}) = \bigwedge_{\omega=1}^{N_F} g_i^{(\omega)}(x_\omega) = \prod_{\omega=1}^{N_F} \frac{1}{\sigma_i^{(\omega)}\sqrt{2\pi}} \exp\left\{ \frac{-(x_\omega - \mu_i^{(\omega)})^2}{2\sigma_i^{(\omega)2}} \right\} \tag{9.19}$$

for rule R_i. For all N_r rules together, the corresponding values $\phi_i(\vec{x})$ are collected in a membership vector

$$\vec{\phi}(\vec{x}) = \begin{pmatrix} \phi_1(\vec{x}) & \phi_2(\vec{x}) & \dots & \phi_{N_r}(\vec{x}) \end{pmatrix}. \tag{9.20}$$

Next, we construct a matrix $\underset{\sim}{C}^{N_F \times N_r}$ of the weighted consequences $\vec{\Omega}(R_i)$, $i \in \{1, \dots, N_r\}$, of all rules by using the weighted consequence vectors for all individual rules R_i. This yields

$$\underset{\sim}{C}^{N_F \times N_r} = \begin{bmatrix} \vec{\Omega}(R_1) & \vec{\Omega}(R_2) & \dots & \vec{\Omega}(R_{N_r}) \end{bmatrix}. \tag{9.21}$$

Now, we can compute the weight vector $\vec{\Psi}$ by multiplying the membership vector $\vec{\phi}(x)$ by the transposed matrix $\underset{\sim}{C}^T$:

$$\vec{\Psi} = \vec{\phi}(\vec{x}) \cdot \underset{\sim}{C}^T = \begin{pmatrix} \Psi_1 & \Psi_2 & \dots & \Psi_{N_\Omega} \end{pmatrix}. \tag{9.22}$$

The vector $\vec{\Psi}$ contains the superpositioned weight values for all N_Ω possible scheduling strategy recommendations, that is, $\vec{\Psi} \in \mathbb{R}^{N_\Omega}$.
Finally, we choose the scheduling strategy with the highest overall value as the output of the rule base system by

$$\arg \max_{1 \le h \le N_\Omega} \left\{ \vec{\Psi}_h \right\}. \tag{9.23}$$

9.2.4.3 Michigan Approach

Within the Michigan approach each individual during the evolutionary optimization represents a single rule. The coding of the object parameters of such a rule R_i are shown in Equation 9.24. Each object vector $\vec{o}_k$ of individual a_k that represents the i-th rule consists of N_F pairs $(\mu_i^{(\omega)}, \sigma_i^{(\omega)})$ each specifying the GMF of the ω-th feature. Further, each individual consists of a vector $\vec{\Omega}$ that specifies the weights of the output recommendation for the N_Ω possible scheduling strategies.

$$\vec{o}_k = \{\overbrace{\underbrace{\mu_i^{(1)}, \sigma_i^{(1)}}_{\text{GMF}},\ \mu_i^{(2)}, \sigma_i^{(2)}, \ldots \mu_i^{(N_F)}, \sigma_i^{(N_F)},\ \underbrace{\vec{\Omega}(R_i)}_{\Omega_1\ \ldots\ \Omega_{N_\Omega}}}^{R_i}\ \} \tag{9.24}$$

As mentioned above, we use a Symbiotic Evolution, as introduced by Juang et al. [88]. This represents the Michigan approach and evaluates the different rules together as several rules are needed to compose a complete rule base. However, all single rules are also competing with each other during the evolution. Within this approach several problems need to be solved. First, we must specify the procedure to build different Fuzzy systems. To this end, the similarity of the various rules will be used. Second, the fitness assignment of individual rules must be introduced. Both major problems are addressed next.

9.2.4.3.1 Similarity Measure for Rules An acceptable Fuzzy rule base is expected to possess a good coverage, that is, the cross correlation between GMFs should be small. Otherwise if only rules with a high cross correlation are combined the system may not be able to cover most of the possible system states in an appropriate way. Therefore, a measure of similarity is needed for the selection of rules during the evolutionary optimization process.

For the Symbiotic approach, a rule base RB is composed by N_r rules. In many cases, several rules exist in a population that are centered on very similar configurations for the conditional part. In order to combine only rules which represent different states within the system RB, a similarity measure is needed. This similarity measure is used to build rule bases by incrementally adding rules with the smallest similarity to the last added rule. Note that the first rule is picked randomly.

Specifically, we define the similarity of two rules by the cross correlation of the corresponding GMFs. This cross correlation $\varphi_{g_1 g_2}$ of two GMFs $g_1(z)$ and $g_2(z)$ for one feature can be computed by Equation 9.25.

$$\varphi_{g_1 g_2} = \int_{-\infty}^{\infty} g_1(z) \cdot g_2(z)\ dz = \frac{1}{\sqrt{2\pi}\sqrt{\sigma_1^2 + \sigma_2^2}} \exp\left\{ \frac{-(\mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)} \right\} \tag{9.25}$$

For two rules R_i and R_j with N_F features we compute the cross correlation by averaging the cross correlation over all features as shown in Equation 9.26.

$$\begin{aligned}\varphi_{R_i,R_j} &= \frac{1}{N_F}\sum_{\omega=1}^{N_F}\int_{-\infty}^{\infty} g_i^{(\omega)}(z)\cdot g_j^{(\omega)}(z)\;dz \\ &= \frac{1}{N_F\cdot\sqrt{2\pi}}\sum_{\omega=1}^{N_F}\frac{\exp\left\{\dfrac{-(\mu_i^{(\omega)}-\mu_j^{(\omega)})^2}{2(\sigma_i^{(\omega)2}+\sigma_j^{(\omega)2})}\right\}}{\sqrt{\sigma_i^{(\omega)2}+\sigma_j^{(\omega)2}}} \qquad (9.26)\end{aligned}$$

Remember that for our Symbiotic Evolutionary Algorithm the number of features N_F remains constant during the whole optimization process.
Based on φ_{R_i,R_j}, we compose the rule bases as shown in Figure 9.4. Intuitively, only those rules are combined to a rule base which have a small similarity. Thus, we try to ensure a high degree of diversity between rules within the rule bases and a high coverage of the feature space.

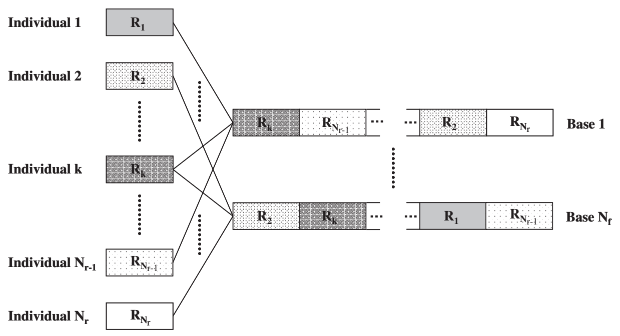


Figure 9.4: Composition of rules to rule systems by Juang et al. [88].

9.2.4.3.2 Fitness Assignment The fitness assignment for all individuals within a generation is obtained by Algorithm 9.3 where N_f and N_r are the number of bases and the number of rules per base respectively. Note that we assume that the child population $Q_{t,\lambda}$ with λ individuals (rules) must be evaluated.

The algorithm generates N_f rule bases RB_n, $n \in \{1,\dots,N_f\}$, by selecting in each case N_r rules from the current child population. During this selection, the total cross correlation between rules is used. The fitness of a rule R_i is calculated by summing the N_r-th part of the fitness of the $N_s^{(i)}$ rule bases where R_i participated divided by the number of participation of R_i.
We subtract the resulting fitness $f(R_i)$ of a single rule R_i from a pre-defined maximum value $f_{\max}$ to achieve a minimization problem. Then, we are consistent with the scheduling objective functions which also must be minimized.
Formally, the fitness assignment $f(R_i)$ of rule R_i is given by

$$f(R_i) = f_{\max} - \frac{1}{N_s^{(i)}}\sum_{n=1}^{N_s^{(i)}}\frac{f_n(R_i)}{N_r}. \qquad (9.27)$$

Algorithm 9.3 Fitness assignment for the Symbiotic Evolutionary Algorithm.

```
$f(R_i) = 0 \; \forall i \in \{1, \dots \lambda\}$;
$N_s^{(i)} = 0 \; \forall i \in \{1, \dots \lambda\}$;
for ($n = 1$ to $N_f$) do
  $S = Q_{t,\lambda}$;
  $RB_n = \emptyset$;
  $R_i$ is selected randomly from $S$;
  $S = S \backslash R_i$;
  $RB_n = RB_n \cup R_i$;
  for ($a = 1$ to $(N_r - 1)$) do
    Select $R_j \in S$ with $R_j = \arg \min_{R_k \in S} \{\varphi_{R_i,R_k}\}$;
    $RB_n = RB_n \cup R_j$;
    $S = S \backslash R_j$;
    $R_i = R_j$;
  end for
  $f_n \leftarrow$ evaluate($RB_n$);
  $f(R_i) = f(R_i) + f_n/N_r \; \forall R_i \in RB_n$;
  for ($i = 1$ to $N_r$) do
    if ($R_i \in RB_n$) then
      $N_s^{(i)} = N_s^{(i)} + 1$;
    end if
  end for
end for
$f(R_i) = f(R_i)/N_s^{(i)}, \; i \in \{1, \dots, \lambda\}$;
```

In Equation 9.27, $f_n(R_i)$ denotes the temporal fitness assignment that is determined by the overall performance of the n-th Fuzzy System of all $N_s^{(i)}$ bases in which R_i participated during the population evaluation process. Usually, it can be assumed that a fair fitness evaluation of the whole population is only guaranteed if the number of participations within the rule bases is constant for all rules within the population. To ensure that this condition is obeyed, we also allow the composition of rules to rule based systems which may have a high degree of similarity. This is only applied, if a rule is very similar to another rule, but has not participated sufficiently enough in different rule bases.

Before we detail the used Evolution Strategy we focus on two major subproblems during the application of the Symbiotic Evolution of Fuzzy systems. First, we will introduce the concept of a default output decision and second the selection of rules from the current population, by using a cluster mechanism, in order to ensure a good coverage of the feature space in the next generation.

9.2.4.3.3 Default Output Decision Unfortunately, Fuzzy control systems may lead to controller scattering. In some cases, the system may initiate a change of output even for a minimal input parameter (feature) variation. To avoid frequent switching between two output decisions, we introduce a threshold for the activation of a controller output. Then, the output of the system is only determined by the output corresponding to the superpositioned membership value if the total degree of membership value exceeds a pre-defined constant

threshold level. Otherwise, a standard output is used.
Using a normalized GMF the feature encoding enables the establishment of a default value by selecting a large $\sigma_i^{(\omega)}$ for the corresponding feature ω. This modeling of a default value is now combined with the above mentioned thresholds.
The threshold value and the default output are part of an evolutionary optimization process. They are encoded in a single chromosome of a second population of default value individuals that is optimized in parallel. The evaluation of the two populations is performed in combination, see Section 9.2.4.3.5.

9.2.4.3.4 Selection by Clusters Selection is an important operator for Evolutionary Algorithms because it controls the diversity of the whole population. The selection operator is able to conserve good solutions for multiple generations. In our case, a rule base should cover most of the possible system states. Furthermore, the population of an Evolutionary Algorithm should also keep and improve those rules that describe niches within the feature space. Therefore, a special selection mechanism has been developed that does not only refer to the individuals' fitness but also to their degree of coverage within the feature space, see Algorithm 9.4.
As mentioned above, a rule's position in the N_F-dimensional feature space is determined by its GMF-$\mu_i^{(\omega)}$-value. To ensure a good coverage of most of the possible system states it is sufficient to consider only the clusters of the GMF centers. Therefore, we restrict our analysis to the identification and clustering of similar GMF-$\mu_i^{(\omega)}$-values of the population. For the determination of clusters we apply the computationally efficient k-Means algorithm with an exchange method as proposed by Späth [149].
Note that the fitness of a cluster is determined by the mean of the C best individuals or all individuals in case that a single cluster consists of less than C individuals. Note that we address the minimization of the fitness objective. This fitness calculation allows the selection of individuals with a good fitness value even if the majority of the cluster members have a very bad fitness. Algorithm 9.4 by μ_s describes the number of rules that must be selected over all clusters. μ_i describes the number of rules that are selected from the i-th cluster. Of course this must be an integer value and hence we round to the next integer value. For our evaluation we have chosen $k = 10$ and $C = 5$.

9.2.4.3.5 Configuration of the Symbiotic Evolutionary Algorithm Different modifications of Evolution Strategies have been developed for parameter optimization. Here, we have chosen a $(\mu + \lambda)$-Evolution Strategy. The specification of the μ and λ values is closely connected to the parameter values for the Symbiotic Approach.

Population Size Note that the variables μ and λ correspond to an Evolution Strategy in this section. The number N_r of rules that form a Fuzzy System is static for the Symbiotic Approach and must be chosen at the beginning of the optimization. After applying several experiments with different values of N_r we have selected $N_r = 50$. The resulting systems have shown a higher performance than systems with a different number of participating rules. As no rule exists twice in a Fuzzy System, each population must consist of at least $\mu = 50$ potential rules. As recommended by Schwefel [136], the ratio of $\mu/\lambda = 1/7$ should be used for the Evolution Strategy. This results in $\lambda = 350$ child individuals in the evolutionary process. Hence, 350 individuals must be evaluated within each generation.

Algorithm 9.4 Selection Strategy.

```
Determine $k$ clusters $\{c_1, \dots, c_k\}$ of all rules $R_1, \dots, R_{N_r}$ in the population $Q_{t,\lambda}$ based on
their GMF-$\mu^{(\omega)}$-values.
for ($i = 1$ to $k$) do
  Compute the mean fitness $\bar{f}(c_i)$ for cluster $c_i$ by selecting the $b = \min\{|c_i|, C\}$ individuals
  $\{R_1, \dots, R_b\} \in c_i$ with the best fitness according to
      $$\bar{f}(c_i) = \frac{1}{b}\sum_{l=1}^{b} f(R_l)$$
  {Here, $C$ represents the number of individuals that are used to determine the mean of
  the fitness of individuals in all clusters.}
end for
for ($i = 1$ to $k$) do
  Add $\mu_i$ individuals from $c_i$ to $P_{t+1,\mu}$ with
      $$\mu_i = \left\lfloor \frac{\mu_s}{\bar{f}(c_i)\sum_{l=1}^{k} 1/\bar{f}(c_l)} + 0.5 \right\rfloor$$
end for
while ($|P_{t+1,\mu}| \leq \mu_s$) do
  Select best $R_i$ from the remaining individuals according to the best cluster. If several
  individuals exist in the best cluster, select the individual with the best fitness;
  $P_{t+1,\mu} = P_{t+1,\mu} \cup R_i$;
end while
```

As proposed by Juang et al. [88], we must ensure that every rule within the population participates sufficiently enough in Fuzzy Systems. In our work, we assume that it is possible to determine the fitness of a rule if it is part of exactly $N_s = 10$ Fuzzy systems. Consequently, we must evaluate $N_f = \frac{350 \cdot 10}{50} = 70$ rule bases in each generation. This results in 70 simulations for each generation. Within this work, we used 40 generations for all Genetic Fuzzy approaches in order to develop the Fuzzy systems within a reasonable amount of computational time. The analysis of the achieved results and the improvements of the last generations regarding the given objective function have shown that 40 generations are appropriate to our problem.

Recombination Operator The recombination operator and also the mutation operator of the next section modify the individual rules during the evolutionary process. To this end, the encoding scheme of Section 9.2.4.1.2 is used.

We haven chosen discrete multi-recombination as recombination operator, see Section 3.1.4. Here, we recombine $\rho = 12$ parents to *one* new offspring. However, those 12 parents are not all randomly selected from the current population. We select $\rho/2 = 6$ parents with a very high degree of similarity and the remaining $\rho/2 = 6$ parents randomly. This ensures that the recombination operator leads to new rules in other areas of the feature space while still covering the already explored areas of the feature space. Hence, this procedure reflects the compromise between exploration and coverage of the feature space.

Note that the recombination does not affect any parameter of the mutation, which is described next.

Mutation Operator For the mutation operator, we have selected the commonly accepted Mutation Success Rule as proposed by Rechenberg [126], see Section 3.1.5.1. Note that in our work a mutation is called successful if a recombined and mutated offspring is better than a randomly selected parent from the ρ parents. During the application of the Mutation Success Rule the value of $\alpha = 0.817$ is used as recommended by Rechenberg. In this work, we use a relatively large population for the Evolutionary Algorithm. The original Mutation Success Rule was developed for the (1+1)-Evolution Strategy [126]. It is considered, for instance, by Deb [27] that the usually optimal ratio 1/5 is too optimistic in case of such a configuration. In order to avoid a continuous reduction of the mutation step size, we have decreased the optimal success rate to 1/8.

Population for Default Values As described above, we use default values to optimize the behavior of the Fuzzy system, see Section 9.2.4.3.3. To this end, we generate a second population of default value individuals that is evolutionary optimized in parallel to the optimization of the Fuzzy rules. Therefore, we introduce a new species of real-value coded individuals that describe a certain threshold and a recommended output decision which is used as default value. This recommended scheduling strategy is used to generate the next schedule if the superpositioning of all rules within the Fuzzy System generates a value that is smaller than the defined threshold. Due to this additional population the whole procedure can be considered as a Cooperative Coevolutionary Algorithm [121] approach, as a default value individual will participate in each generated Fuzzy System. Both populations exist genetically isolated within the ecosystem, that is, individuals will only mate with other members of their own species. By evolving both species in different populations these mating restrictions are always obeyed.
The default value population consists of N_f individuals for the offspring generation. This is reasonable, as the evaluation of the Fuzzy rules require N_f temporarily Fuzzy systems. Each of these Fuzzy systems includes one default value. Hence, the corresponding population should also consists of $N_f = 70$ individuals. By applying the recommendation of Schwefel [136] ($\mu/\lambda = 1/7$) this leads to a (10+70)-Evolution Strategy. So, each individual participates in one Fuzzy system which is generated for the Symbiotic fitness evaluation.
For the actual fitness evaluation, the two populations must cooperate. Once a Fuzzy system has been generated by composing it from rule individuals, a single default value is added to this system, and the overall performance is evaluated by simulation. The assigned fitness value for the default value individual is equal to the performance of the Fuzzy system in which this individual has participated. Note that the fitness value of the rule population is influenced by the default value individuals. So, this cooperation may lead to better rule individuals and default value individuals as well. For the default value population, we apply the "two-point rule" of Rechenberg [12] with $\alpha = 0.8$ and do not use recombination.

9.2.4.4 Pittsburgh Approach

Within the Pittsburgh approach, each individual represents a complete rule base, and consequently interaction among individuals does not occur. In opposite to the Michigan approach, the rule reduction within the different rule bases is easier with the Pittsburgh approach. Here,

the number of rules can be encoded using an endogenous strategy parameter during the evolution. Further, using individuals that represent complete rule bases with differing numbers of rules, the decoding of individuals must be changed as a positional representation cannot be applied anymore.
For each individual we specify the number of rules N_r at the beginning. However, this rule number N_r will not evolve in all of our evolutionary optimizations. Thus, a rule based scheduling system with a constant number of rules can also be modeled using the later presented encoding. Furthermore, we construct a complete rule base RB with N_r rules as given in Equation 9.15. A single rule consists of $2 \cdot N_F$ elements per rule within the conditional part. Additionally, we include the vector $\vec{\Omega}(R_i)$ for the consequence part, which consists of $N_\Omega = 13$ elements. Therefore,

$$\vec{o}_k = \{N_r, \overbrace{\underbrace{\mu_1^{(1)}\ \sigma_1^{(1)}}_{\text{GMF}}, \dots, \mu_1^{(N_F)}\ \sigma_1^{(N_F)},\ \underbrace{\vec{\Omega}(R_1)}_{\Omega_1\ \dots\ \Omega_{N_\Omega}}}^{R_1}, \overbrace{\mu_2^{(1)}\ \sigma_2^{(1)}, \dots, \mu_{N_r}^{(N_F)}\ \sigma_{N_r}^{(N_F)}, \vec{\Omega}(R_{N_r})}^{R_2\ \dots\ R_{N_r}}\} \tag{9.28}$$

is the coding scheme of the object parameter vector $\vec{o}_k$ of individual $\vec{a}_k$ which is a complete rule base. Hence, the number of elements u within the object parameter vector $\vec{o}_k$ of the individual $\vec{a}_k$ can be computed by

$$u = 1 + N_r \cdot (2 \cdot N_F + N_\Omega). \tag{9.29}$$

If we used the same number of rules as for the Michigan approach ($N_r = 50$, $N_F = 7$, $N_\Omega = 13$), each individual within the Pittsburgh approach would consist of $1+50\cdot(2\cdot 7+13) = 1351$ object parameters. Such a problem size requires well defined genetic operators as the search space is large. However, the fitness assignment process is relatively easy compared to the Michigan approach.

9.2.4.4.1 Reducing the Number of Rules As mentioned above, the reduction of rules within individual rule bases is relatively easy using the Pittsburgh approach. This procedure potentially leads to a smaller set of rules that is able to cover a similar feature space as rule bases with many more rules. Furthermore, the number of conflicting or neutral rules might decrease. Moreover, this attempt of adapting the number of necessary rules solves the underlying initial problem, as a good number of rules for the rule bases is not known in advance.
In many research projects the reduction of the number of rules within Fuzzy systems has been discussed, see, for example, Jin et al. [88] or Cordón et al. [24]. However, all these approaches require the availability of training data in order to determine the coverage and the quality of the resulting Fuzzy rule bases after removing several rules. Hence, those approaches are not suitable to our problem as we do not have such training data.
Here, we apply the concept of modifying the selection operator during the evolutionary optimization. Within a first approach, we modify the fitness of individuals depending on their number of rules. Then, the selection is not only based on the original performance but also on the size of the rule bases. In a second approach, we modify the selection operator such that it prefers rule bases with fewer rules in all cases where rule bases have the same fitness value. As already introduced in Equation 9.28, the number of rules within a rule base is encoded by N_r which is part of the objective parameter vector of each rule base and hence can be

used directly for the two approaches described above. However, we use the parameter $N_{r_{max}}$ to specify an upper bound for the number of rules. Hence, N_r can only be varied between 1 and $N_{r_{max}}$ ($1 \leq N_r \leq N_{r_{max}}$).

9.2.4.4.2 Modifying the Fitness Function In the following, we assume that the fitness of an individual a_k during the evolutionary optimization is given by $f_{obj}(a_k)$. To this end, the objective functions derived in Chapter 8 are used to calculate this value.
The rule reduction strategy modifies this value to generate a new fitness value $f_{new}(a_k)$ as shown in Equation 9.30.

$$f_{new}(a_k) = f_{obj}(a_k) - \frac{\beta}{N_r^{(a_k)}} \tag{9.30}$$

Note that we assume a minimization problem. Thus, a small number of rules N_r for a given individual a_k leads to a decrease of the fitness of the rule base specified by a_k and hence emphasizes the quality of this rule base.
The parameter β must be chosen carefully. If this value is too large, the objective minimization becomes less important than the rule base reduction. However, if β is too small, the original objective definition is dominating, which potentially leads to rule bases that consist of many rules.
The major drawback of this approach to reduce the number of rules within the rule bases is the definition of the parameter β. Hence, we also applied another approach of modifying the selection strategy itself.

9.2.4.4.3 Modifying the Selection Strategy The normal selection operator within an Evolution Strategy with $\kappa = \infty$ simply chooses the best μ individuals from the set of μ parents and λ offsprings. In Algorithm 9.5 the modification is detailed.
First, a pool $P'_{t,2\mu}$ of 2μ of the best individuals regarding the objective function is generated from all parent and offspring individuals. During this procedure, rules with a smaller number of rules are selected in cases where two rules with the same objective value are going to be selected. Within our optimizations, such cases of equal objective values are observed frequently as many individuals within some generations shared the same objective value. Hence, this selection strategy really changed the resulting rule systems. The next parent generation $P_{(t+1),\mu}$ is generated by choosing the individuals from $P'_{t,2\mu}$ with the smallest number of rules. This selection procedure ensures that individuals with a good objective are selected by emphasizing rule bases that consist of a small number of rules. The described procedure should only be considered as an example to modify the selection operator. Other values as 2μ could be chosen during the first selection.

9.2.4.4.4 Configuration of the Evolution Strategy This section provides the details of the settings for the Evolution Strategies that are applied to the various modifications based on the Pittsburgh approach.
Within the Pittsburgh approach, the individuals have larger object parameter vectors compared with the Michigan approach. Hence, we adapted the genetic operators as well as the number of individuals. To this end, we have chosen a non-isotropic mutation, see Section 3.1.5.2.2, as this allows the individual adaptation of the mutation for the different dimensions. Therefore, each object parameter of the individuals consists of a corresponding strategy

Algorithm 9.5 Modified selection operation during the Evolution Strategy.

```
$P'_{t,2\mu} = \emptyset$;
$F \leftarrow \{P_{t,\mu} \cup Q_{t,\lambda}\}$;
while $(|P'_{t,2\mu}| < 2\mu)$ do
  $a_k \leftarrow \arg\min_{a_i \in F} f_{obj}(a_i)$;
  $a_l \leftarrow \arg\min_{a_i \in \{F \setminus a_k\}} f_{obj}(a_i)$;
  if $(f_{obj}(a_k) = f_{obj}(a_l))$ then
    if $(N_r^{(a_k)} \leq N_r^{(a_l)})$ then
      $P'_{t,2\mu} \leftarrow P'_{t,2\mu} \cup a_k$;
      $F \leftarrow F \setminus a_k$;
    else
      $P'_{t,2\mu} \leftarrow P'_{t,2\mu} \cup a_l$;
      $F \leftarrow F \setminus a_l$;
    end if
  else if $(f_{obj}(a_k) < f_{obj}(a_l))$ then
    $P'_{t,2\mu} \leftarrow P'_{t,2\mu} \cup a_k$;
    $F \leftarrow F \setminus a_k$;
  else
    $P'_{t,2\mu} \leftarrow P'_{t,2\mu} \cup a_l$;
    $F \leftarrow F \setminus a_l$;
  end if
end while
$P_{(t+1),\mu} \leftarrow$ select $\mu$ individuals from $P'_{t,2\mu}$ regarding to their number of rules $N_r$;
```

parameter that specifies its mutation strength. Further, we apply a standard Evolution Strategy with $\mu = 3$ parent and $\lambda = 21$ offspring individuals. The ratio of 1/7 is suggested by Schwefel [135]. The values of μ and λ are much smaller compared with the Michigan approach. However, for the Pittsburgh approach, we do not need several Fuzzy systems in order to evaluate individual rules. Hence, it is possible to reduce the number of Fuzzy systems, that is equivalent to the number of individuals. Furthermore, these smaller μ and λ values lead to less necessary simulations and hence to a quicker development of the rule bases.

During the application of the Evolution Strategies we do not use any recombination. Especially, if the number of rules differs between the individuals within a generation, a reasonable recombination is hard to describe as a positional recombination cannot be applied.

Each optimization is based on 40 generations. The first 3 individuals are randomly initialized. Thus, our (3+21)-Evolution Strategy leads to $3 + (40 \cdot 21) = 843$ evaluations for the development of a single rule base.

Configuration for the Optimization with the unmodified Pittsburgh Approach Within the unmodified case, we use a constant number of rules $N_r = 10$ for each rule base. This rule number is less than the $N_r = 50$ rules used within the Michigan approach. However, the usage of 50 rules would lead to very large individuals, as shown in Section 9.2.4.4 and the achieved results would highly depend on the selected genetic operators. Further, as a whole rule base is evolving over time, it can be expected that fewer rules fit to the underlying problem. In the Michigan approach, rules are flexibly combined to rule bases which might result in rule bases with some conflicting rules.

The constant number of rules leads to a constant number of object and strategy parameters within each individual. Hence, $u = N_r \cdot (2 \cdot N_F + N_\Omega) = 10 \cdot (2 \cdot 7 + 13) = 270$ parameters must be determined. Thus, the two exogenous learning rates for the non-isotropic mutation, see Section 3.1.5.2.2, are defined as:

$$\tau_0 = \frac{1}{\sqrt{2 \cdot u}} = 0.043, \text{ and} \tag{9.31}$$

$$\tau_1 = \frac{1}{\sqrt{2\sqrt{u}}} = 0.174. \tag{9.32}$$

Configuration for the Optimization using a Modified Fitness Function For this approach, the introduced Evolution Strategy is modified. First, the number of rules within each individual is a variable. However, we limit the number of rules as we allow at most $N_{r_{max}} = 30$ rules within a rule base. Thus, the largest rule bases consist of

$$u = N_{r_{max}} \cdot (2 \cdot N_F + N_\Omega) = 30 \cdot (2 \cdot 7 + 13) = 810 \tag{9.33}$$

parameters. Hence, we need to adapt the learning rates for the non-isotropic mutation by:

$$\tau_0 = \frac{1}{\sqrt{2 \cdot u}} = 0.025, \text{ and} \tag{9.34}$$

$$\tau_1 = \frac{1}{\sqrt{2\sqrt{u}}} = 0.133. \tag{9.35}$$

As given in Equation 9.30, we need to define the parameter β that describes the influence of the rule base size to the resulting objective value. The setting of β highly depends on the original objective values. As we do not have deeper knowledge about reasonable values, we applied the following heuristic to determine our β. For all available workloads we used the achieved scheduling results by using the EASY scheduling strategy. Then we calculated the objective function values for these results using Equation 8.24. Finally, we calculated the average of these objectives by ignoring the objective calculated for the SDSC00 workload. This workload leads to scheduling results that drastically differ from the results achieved for all other workload traces. The final β-value is set to the half of the average objective value, that is $\beta = 50000$. Of course, this β-value needs to be modified for other objective functions.

Configuration for the Optimization using a Modified Selection Strategy For the optimizations with the modified selection strategy we used the same Evolution Strategy and parameter settings as in the previous approach (of course without β). During the optimizations we frequently encountered a premature convergence by using a normal plus strategy. Hence, we used $\kappa = 4$, see Section 3.1, to limit the lifetime of individuals. We determined this value of κ by several simulations with different κ settings. Overall, the usage of a limited lifetime of individuals highly improved our results.

9.2.4.5 Coevolutionary Genetic Fuzzy System Development

As presented in Section 9.2.2, for each scheduling state the rule based scheduling system needs to determine a corresponding sorting criterion and a scheduling algorithm. In the previously introduced rule based scheduling systems, a whole scheduling strategy, consisting of both a sorting criterion and a scheduling algorithm, was assigned to the different scheduling states. However, this combined assignment is not necessary. Moreover, the assignment of the same sorting criterion to two scheduling states within the feature space does not always lead to the assignment of the same scheduling algorithm. This motivates the usage of a Coevolutionary Algorithm as the assignment problem can easily be decomposed into two subproblems.

9.2.4.5.1 Concept of Cooperative Coevolutionary Algorithms Coevolutionary Algorithms potentially lead to better solutions compared with standard Evolutionary Algorithms, if the problem can be decomposed into two subproblems, see, for example, Jansen et. al [85]. Furthermore, Potter and De Jong [124] have proven that Coeveolutionary Algorithms achieve better results with fewer generations compared with standard evolutionary optimization techniques.
In this work, we apply the commonly called Cooperative Coevolutionary Algorithm (CCA), see Paredis [121]. This model uses two distinct species. Both species are genetically isolated. Hence, the genetic operations are only applied to individuals of the same species. The two different species are evolved in two different populations in parallel by using standard Evolution Strategies. However, during the fitness evaluation, two individuals of each species must cooperate. In general, this concept allows a larger number of species. Figure 9.5 provides an overview of the Cooperative Coevolutionary Algorithm concept.

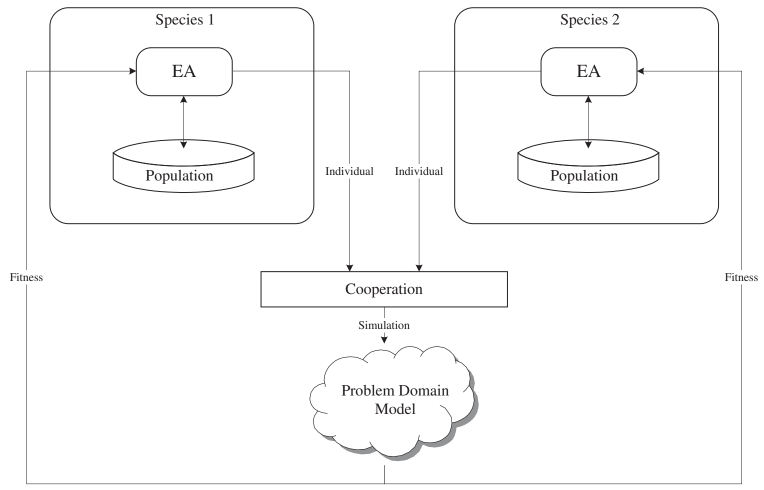


Figure 9.5: General concept of a Cooperative Coevolutionary Algorithm with two species.

Algorithm 9.6 describes the concept in more detail. First, two species with μ individuals each are randomly generated. Then, the individuals of both species are evaluated by randomly combining two individuals, one from each species. Note that other selection schemes are also possible and discussed in the literature, see, for example, Panait et al. [120]. However, those

Algorithm 9.6 CCA based on a $(\mu + \lambda)$-Evolution Strategy.

```
for ($s = 1$ to 2) do
    $P^{(s)}_{0,\mu} \leftarrow$ initialization with $\mu$ individuals, $P^{(s)}_{0,\mu} \in \mathcal{M}_\mu(\mathbb{I})$;
end for
Evaluate $\left(P^{(1)}_{0,\mu}, P^{(2)}_{0,\mu}\right)$ by combining their individuals randomly;
{Each individual is evaluated exactly once.}
$t \leftarrow 0$;
repeat
    for ($s = 1$ to 2) do
        $Q^{(s)}_{t,\lambda} \leftarrow$ generate $\lambda$ new individuals from $P^{(s)}_{t,\mu}$ by performing self-adaptive coordinate
        wise mutation, $Q^{(s)}_{t,\lambda} \in \mathcal{M}_\lambda(\mathbb{I})$;
        {We do not use recombination ($\rho = 1$) and use a plus-strategy ($\kappa = \infty$)}
    end for
    Evaluate $\left(Q^{(1)}_{t,\lambda}, Q^{(2)}_{t,\lambda}\right)$ by combining their individuals randomly;
    {Each individual is evaluated exactly once.}
    for ($s = 1$ to 2) do
        $P^{(s)}_{(t+1),\mu} \leftarrow$ select$(P^{(s)}_{t,\mu} \cup Q^{(s)}_{t,\lambda})$;
    end for
    $t \leftarrow t + 1$;
until termination condition;
```

methods need more evaluations and additional simulations in our case. In order to avoid this effort, we use our simple heuristic. After evaluation, the genetic operators produce λ offsprings for each species separately. Then, the resulting offspring individuals are again evaluated by a randomly chosen cooperation. Finally, normal evolutionary selection determines the next parent generation.

9.2.4.5.2 Rule Based Scheduling Development by applying Coevolutionary Algorithms As mentioned above, our scheduling problem can be decomposed into two separate subproblems. This concept is shown in Figure 9.6. Contrary to the general rule based scheduling, see Figure 9.1, we use two separate rule bases within the same feature space. One determines the sorting criterion depending on the system state and the other calculates the scheduling algorithm. However, the partitioning of this feature space differs between the two species. To this end, the different GMF-$\mu_i^{(\omega)}$ and GMF-$\sigma_i^{(\omega)}$ values are determined separately for the two species. The resulting scheduling system is expected to react very accurately on certain system states.

Such a coevolutionary approach yields several potential advantages for the resulting scheduling system and for the extraction process of appropriate rule bases.

First, each of the two separate rule bases has fewer output recommendations. In detail, for the sorting criterion as well as for the scheduling algorithm, we have only $N_\Omega = 4$ possible output recommendations instead of 13 as in the combined scenario. This reduces the length of the individuals within the populations and enables a better and faster adaptation. However, note that the sorting criterion is redundant if the Greedy scheduling algorithm is selected since Greedy includes its own sorting. Second, as the feature space partition can be optimized for

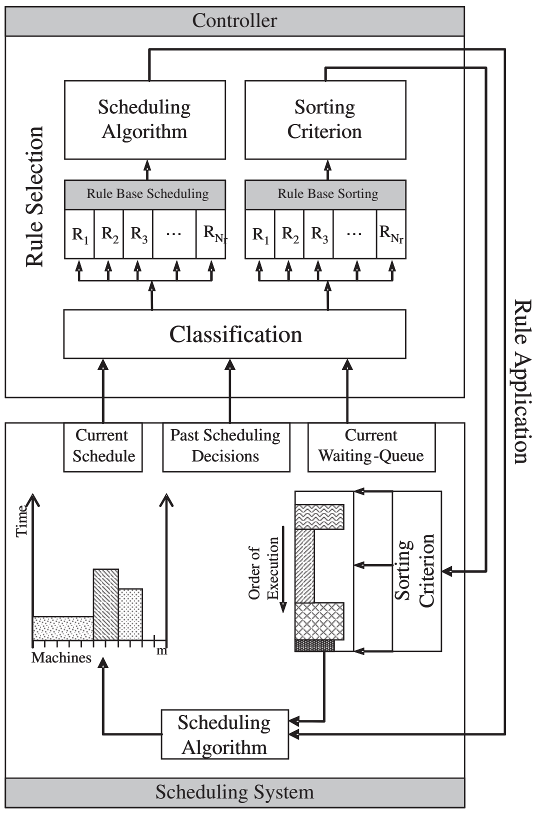


Figure 9.6: General concept of the rule based scheduling approach with dedicated rule bases for scheduling algorithm and sorting criterion.

both species separately, fewer rules might be required for each species.

Configuration of the Evolution Strategies The Evolution Strategies for both populations are identical. We apply the Pittsburgh approach with the same genetic operators and no recombination for both populations. In detail, we use a constant number of rules $N_r = 10$ and a (3+21)-Evolution Strategy for both populations. The optimization is limited to 40 generations. Consequently, each individual within the populations consists of

$$u = N_r \cdot (2 \cdot N_F + N_\Omega) = 10 \cdot (2 \cdot 7 + 4) = 180 \tag{9.36}$$

object parameters. Hence, we adapt the learning rates for the non-isotropic mutation by:

$$\tau_0 = \frac{1}{\sqrt{2 \cdot u}} = 0.053, \text{ and} \tag{9.37}$$

$$\tau_1 = \frac{1}{\sqrt{2\sqrt{u}}} = 0.193. \tag{9.38}$$

Chapter 10

Evaluation of the Generated Scheduling Strategies

In the following, we present the scheduling results that are generated using the different strategies presented in the previous chapter. First, the Greedy scheduling strategy results will be discussed. Only during this discussion, we compare some strategies resulting from the optimizations of the region based and the linear objective functions. After this, only the linear objective function is applied for our comparisons. Second, we present the results corresponding to all rule based scheduling strategies in detail. Finally, we compare all of our developed approaches and discuss the corresponding recommendations for the different usage scenarios.

For all evaluations, we execute discrete event simulations using the scaled workload traces, see Chapter 5. As mentioned above, we apply the two objective functions derived in Chapter 8. Some of the achieved results are presented relative to the EASY standard scheduling strategy introduced in Section 4.4. We compare them to EASY as this strategy performs best regarding the defined objective function for the majority of the workload traces, see Appendix F. The performance of CONS is similar to the performance of EASY for most of the workloads. FCFS always performs worse compared with EASY and CONS.

This relative measurement provides the potential improvements that can be achieved in real environments in comparison to the frequently used strategy $EASY$. For each objective O and a developed strategy str we define the relative value $O_{[\%]}$ as described in Equation 10.1.

$$O_{[\%]} = \frac{O_{EASY} - O_{str}}{O_{EASY}} \cdot 100 \tag{10.1}$$

Note that for all average weighted response times and the objective function values a positive $O_{[\%]}$ describes that the new scheduling strategy str is better than EASY as all of those objectives must be minimized. However, for the utilization (UTIL) this is different, as this objective has to be maximized. Hence, in this case a positive value of $O_{[\%]}$ shows that the EASY is superior.

Furthermore, we present our achieved results relative to the Pareto front of all feasible schedules for the simulated workloads. Noteworthy, the Pareto front is generated off-line and it cannot be taken for granted that this front can be reached by our proposed online scheduling strategies at all. Therefore, we refer to this front as a reference for the best achievable solution. As explained in Chapter 7, our Pareto front is only an approximation as it is de-

rived by heuristics. Although we do not know the real Pareto front, the high diversity of our approximation indicates that the quality of the approximation is very good.
During most of our evaluations, we use the defined linear objective function, see Equation 8.24, to characterize the scheduling results. However, this is problematical in all cases where the scheduling strategies were optimized using the region based objective function. The main problem arises from the fact that the solutions within the preferred region, see Section 8.2.1, are of equal importance to the machine provider whereas the different basic scheduling objectives vary. Hence, we only use the linear scheduling objective in order to demonstrate that scheduling strategies, which were optimized using the region based approach, improve the scheduling outcome in comparison to the EASY.

10.1 Greedy Scheduling Strategy

The absolute results for the adapted Greedy scheduling strategy, as explained in Section 9.1, are given in Table 10.1 for the CTC workload trace. The detailed results of all Greedy adaptations for all workload traces are given in Appendix H. Note that the presented results correspond to the best Greedy strategy using the CTC workload trace. This best strategy was achieved by parameterizing the sorting criterion defined in Equation 7.3. Furthermore, this sorting criterion performed best in all of our different optimization approaches using the Greedy approach.

Type	Method	AWRT	AWRT1	AWRT2	AWRT3	AWRT4	AWRT5	UTIL	f_{obj}
SOPT (CTC)	linear	55236.17	52755.80	61947.65	56275.18	54017.23	35085.84	66.99	775348.55
SOPT (CTC)	region	55444.01	53569.68	63350.92	58753.23	51106.22	37509.74	66.99	789100.50
ALLOPT	linear	55301.26	55104.02	61644.73	56354.42	53310.46	34913.51	66.99	797619.16
EASY		53186.81	59681.28	64976.07	50317.47	46120.02	31855.68	66.99	856717.01

Table 10.1: Results for the CTC workload trace based on the optimization using the CTC trace. All AWRT values are given in seconds, the utilization (UTIL) in %.

Table 10.1 shows that the Greedy strategy outperforms the standard scheduling strategies regarding the defined objective function. As one can see, the objective values for the Greedy strategies trained using the region based and the linear objective functions perform better than the EASY strategy. As discussed above, we cannot conclude that the Greedy strategy optimized using the linear objective function performs better than the strategy that was developed using the region bases objective function as our measure is the linear objective. However, the results clearly indicate that the region based approach leads to a scheduling strategy with a good behavior. The Greedy strategy that was developed using all workload traces together (ALLOPT) performs better for the CTC workload than EASY.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj}\%$
SOPT(CTC), linear objective								
EASY	-3.85	11.60	4.66	-11.84	-17.12	-10.14	0	9.50
SOPT(CTC), region based objective								
EASY	-4.24	10.24	2.50	-16.77	-10.81	-17.75	0	7.89
ALLOPT, linear objective								
EASY	-3.98	7.67	5.13	-12.00	-15.59	-9.60	0	6.90

Table 10.2: Relative results in % for the CTC workload trace.

The same results are presented in Table 10.2 relative to EASY. The Greedy strategy derived with the linear objective function leads to an improvement of more than 9 %. The region based approach results in an improvement of more than 7 % regarding the defined objective. The Greedy strategy that is developed using all workload traces together improves the scheduling result for the CTC workload trace by at least 6 %.

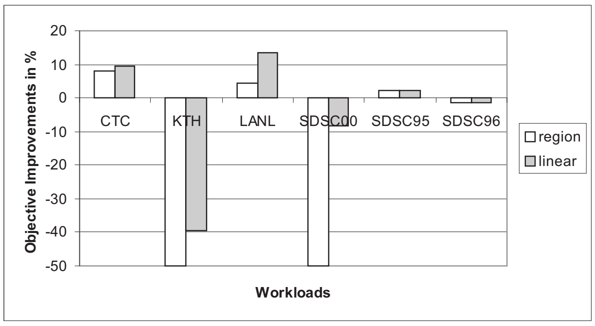
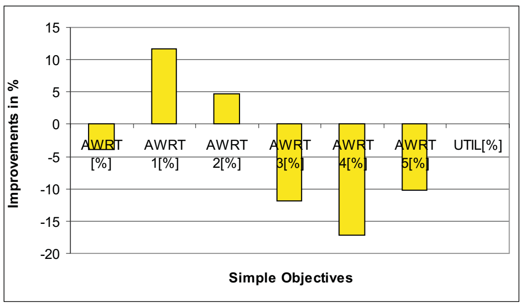


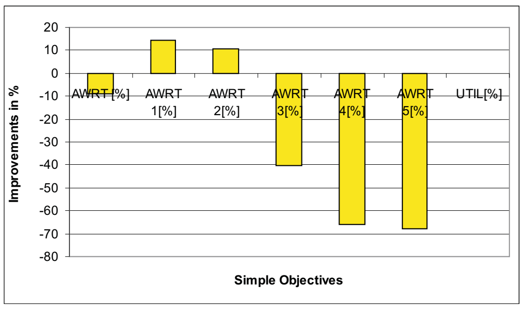
Figure 10.1: Relative results for the Greedy strategy (SOPT(CTC)) applied to all workloads. The real results using the region based optimization approach are for KTH: -260 % and for SDSC00: -400 %.

Figure 10.1 presents the results for all workload traces that were achieved using the Greedy strategy that is optimized using the CTC workload trace. Again, the results for the region based and the linear objective function approaches are given. The results indicate that the two resulting Greedy strategies behave well for the CTC, LANL, and SDSC95 workload traces. Both strategies fail for the KTH, SDSC00, and SDSC96 workload traces.

From these results we conclude that the Greedy strategies developed with both objective functions behave similar. The different results for the SDSC00 and KTH workload traces may result from the already mentioned problem that all solutions within the preferred region are of equal quality in this approach. Thus, a more precise definition with smaller regions might solve this problem. We therefore use the linear objective function in the remainder of this work.



(a) CTC.



(b) LANL.

Figure 10.2: All simple objectives for the CTC and LANL workload traces. The results are related to EASY and given in %. The algorithm is optimized for the CTC workload using the linear objective function.

Table 10.2 presents the results for all simple objectives. It can be observed that the relative AWRT values of user groups 1 and 2 are improved (the absolute values are decreasing). However, this improvement which increases the objective function corresponds with a deterioration of the AWRT values of all other user groups. Figure 10.2a shows this behavior for the CTC workload trace. A very similar outcome can be observed for all other workload traces with the CTC optimized Greedy strategy.

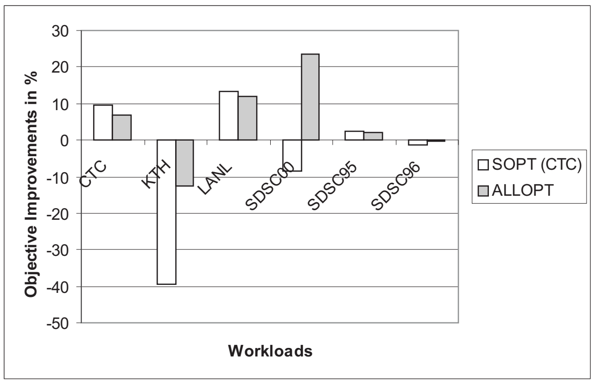


Figure 10.3: Relative results using the defined linear objective function.

Figure 10.2b displays these results for the LANL workload trace. One of the major results from all these observations is the invariance of the utilization, see all results in Appendix H. Figure 10.3 presents the relative results for all workload traces using the two Greedy strategies that were optimized for the CTC workload trace and for all workloads together. Both Greedy strategies perform better compared with EASY in most of the cases. However, the Greedy strategies suffer for the KTH workload trace. The Greedy strategy that was optimized using all workload traces is able to improve the scheduling results for the SDSC00 workload traces. In this case, the CTC optimized Greedy strategy fails. Thus, as expected, the Greedy strategy that was generated using all workload traces is more robust than the Greedy strategy optimized for the CTC workload trace.

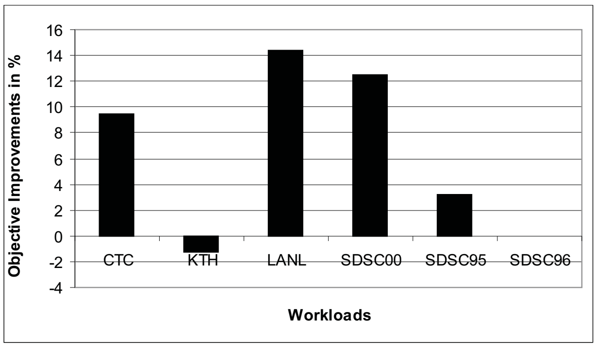
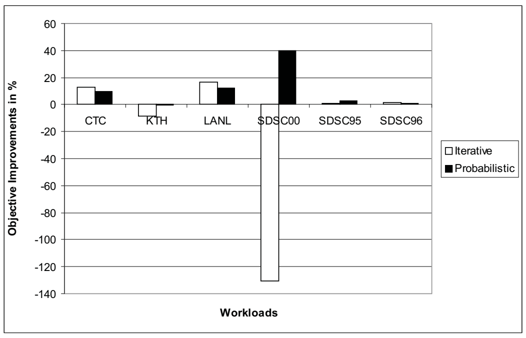


Figure 10.4: Relative results that are achievable for all workloads using one of the Greedy strategies of the last generation that were optimized for the CTC workload trace.

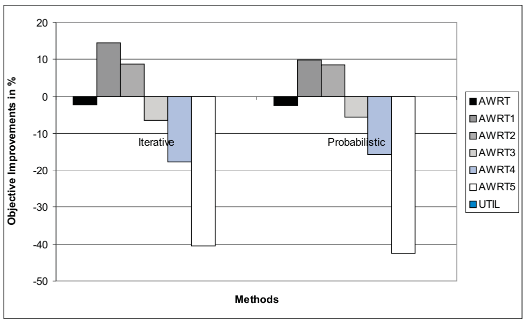
Figure 10.4 presents an interesting observation made for the Greedy optimization using the CTC workload trace only. We applied all Greedy strategies of the last population from the optimization with the CTC workload trace to all workload traces. Figure 10.4 shows the best scheduling results for each workload individually using one of those Greedy strategies. As one can see, the scheduling outcomes are much better compared with the results presented in Figure 10.3. Thus, an evolutionary optimization using a single workload trace leads to a set of strategies that perform very well for other workload traces. For example, the scheduling outcome of the SDSC00 workload traces improves by more than 35 % compared with EASY.

10.2 Rigid Rule Based Scheduling Systems

Next, we show the results of the first two of our rule based scheduling systems. Figure 10.5a presents the objective improvements of the iterative and the probabilistic approaches for the case that the scheduling strategies are optimized for each workload trace individually.



(a) Objective function values.

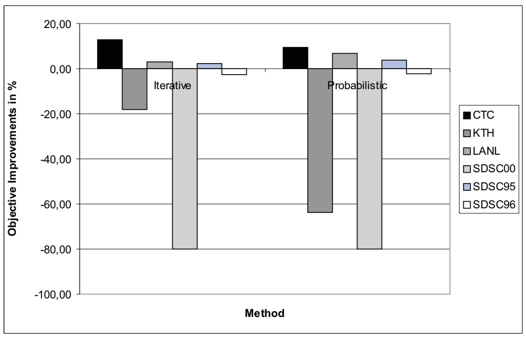


(b) Simple objectives using the CTC workload trace.

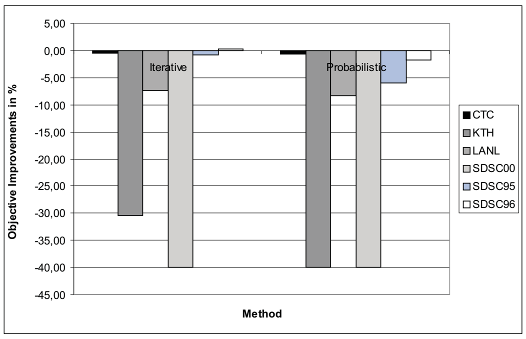
Figure 10.5: Relative results for both rigid rule based scheduling strategies and all workload traces.

Figure 10.5a shows that the probabilistic approach is able to improve the objective value for the CTC, the LANL, and the SDSC00 workload traces. The iterative approach has a better performance than the probabilistic approach for the CTC and the LANL workload. Here, the objective is improved by at least 15 %. However, in opposite to the probabilistic approach, the iterative approach fails for the SDSC00 workload. In this case, the relative objective is reduced by about 130 % whereas the probabilistic approach improves the relative objective by about 40 % in this case. The detailed results are given in the Appendices I.1 and J.1. For the SDSC95 and the SDSC96 workload traces, the objectives do not vary much for both approaches. Furthermore, using the iterative approach the objective for the KTH workload trace decreases by about 9 % whereas the objective is nearly unchanged using the probabilistic approach. Figure 10.5b presents the details of our 7 simple objectives using the CTC workload trace. The results are given relative to the results achieved by using EASY. Here, we can observe the same behavior as for the Greedy scheduling. The average weighted response times for the user groups 1 and 2 decrease (the performance increases at around 10 %). In parallel, the user groups 3 to 5 must accept a worse average weighted response time. Again, as observed for the Greedy scheduling strategy, the utilization is nearly constant.
In Figure 10.6a we show the achieved results by optimizing the iterative approach as well as the probabilistic approach for the CTC workload trace and by applying the two resulting

scheduling strategies to all workload traces. The detailed results are given in the Appendices I.2 and J.2.



(a) SOPT(CTC) applied to all workloads. Note that the results for SDSC00 are cut. The real results are for iterative: -358 % and for probabilistic: -425 %.



(b) ALLOPT applied to all workloads. Note that the results for SDSC00 are cut. The real results are for iterative: -360 % and for probabilistic: -985 %. Furthermore, the KTH results for the probabilistic approach are also cut as the real result is -98 %.

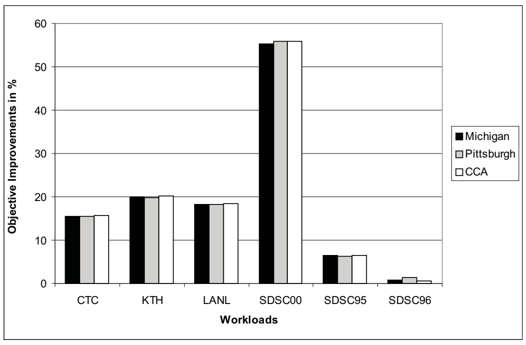
Figure 10.6: The relative results of SOPT(CTC) and ALLOPT for both rigid rule based scheduling strategies applied to all workload traces.

It can be observed that the performance of the scheduling strategy optimized for the CTC workload trace also increases the performance of the scheduling systems using the LANL and SDSC95 workload traces. However, these improvements are less than 3 % for the iterative rule base scheduling strategy and less than 7 % for the probabilistic approach. The application of the scheduling strategies developed using the CTC workload trace fail for the KTH, SDSC00, and SDSC96 workload traces. In the case of the SCSC96 workload trace those changes are less than 2 %. However, for the KTH and the SDSC00 workload traces, the performance is drastically reduced compared with the results achievable by using EASY. Hence, a scheduling strategy that is optimized for a single workload trace, using both rigid rule base approaches, cannot be applied to other workload traces. The resulting scheduling system might perform worse compared with the standard scheduling strategies.

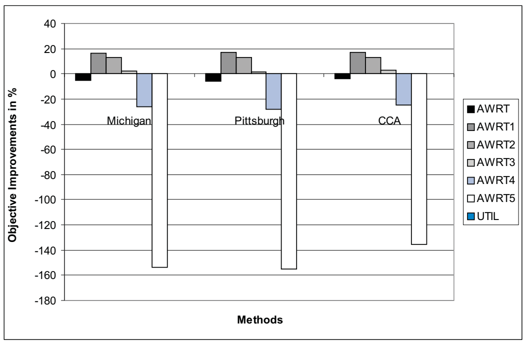
Figure 10.6b presents the results that are generated by applying a scheduling strategy that was developed using all available workload traces together. The ALLOPT scheduling strategy was developed for both rigid approaches individually. The complete results can be found in the Appendices I.3 and J.3. The results presented in Figure 10.6b show that the resulting strategy for the iterative approach only slightly improves the objective for the SDSC96 workload. However, for all other workloads, the objective is decreased. Especially for the KTH and SDSC00 workload traces those objective reductions are larger than 30 %. The probabilistic approach leads to schedules with a very low objective value. For the KTH and SDSC00 workload trace, the probabilistic approach is drastically worse compared to EASY. Thus, the ALLOPT method should not be applied with the rigid rule based scheduling systems.

10.3 Scheduling Strategies based on Genetic Fuzzy Systems

Figure 10.7a shows the objective improvements of the scheduling strategies that are based on Genetic Fuzzy systems and optimized for each individual workload trace. All detailed results can be found in Appendices K.1, L.1, and M.1. As one can see, all three approaches improve the objectives for all available workload traces. The scheduling results for the SDSC96 workload trace are nearly unchanged. The objective values for the SDSC95 workload improve at about 6 %. The scheduling results for all other workload traces improve at least by 15 %. The improvements for the SDSC00 workload are even higher than 50 %. The results in Figure 10.7a demonstrate that the individual optimization of the three approaches that are based on Genetic Fuzzy systems leads to very similar results for each workload trace.



(a) Objective function values.



(b) Simple objectives of the CTC workload trace.

Figure 10.7: Relative results for all strategies based on Genetic Fuzzy systems and all workload traces.

Figure 10.7b displays the 7 simple objectives for all three Genetic Fuzzy system approaches for the CTC workload trace relative to the objectives obtained by using EASY. Similar to the Greedy and rigid rule based scheduling approaches, the user groups 1 and 2 have a shorter average weighted response time and hence these objectives are improved by at least 15 %. Contrary to the Greedy and the two rigid rule based scheduling approaches, the average weighted response time for user group 3 improves as well. The average weighted response times of user groups 4 and 5 are becoming much worse compared with EASY.
Figure 10.8 shows the objective improvements for all workload traces achieved by the Pittsburgh approach and the two corresponding rule reduction attempts, presented in Sections 9.2.4.4.2 and 9.2.4.4.3. The detailed results are given in Appendices L.4 and L.5. Except for the SDSC96 workload, the objective of the scheduling systems with smaller amounts of rules is nearly as good as the results achieved by the original Pittsburgh approach. The different numbers of rules are listed in Tables L.7 and L.9.
The rule reduction with a modified fitness function leads to a single rule for all workloads except of the SDSC00 workload trace. This result is not surprising, as our selected sorting criteria for the waiting queue already focus on the optimization of the user groups 1 and 2. Hence, the single rule that is applied specifies the sorting by user groups and a simple FCFS. Only for the SDSC00 workload trace, 23 rules are extracted.
The results for the attempt by reducing the number of rules using a modified selection operator are similar. In three cases, only a single rule is applied. However, for the CTC, the LANL,

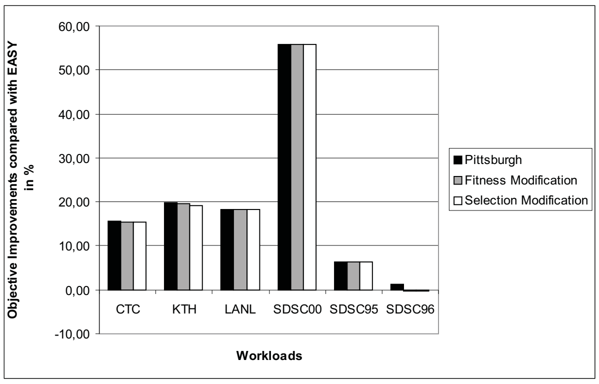
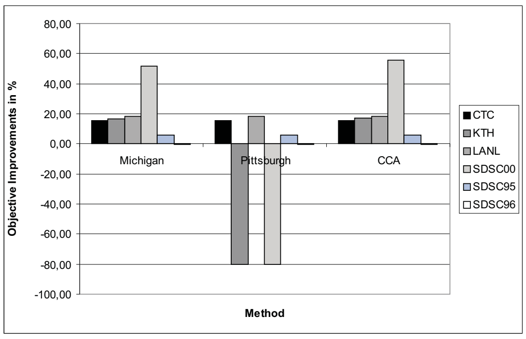
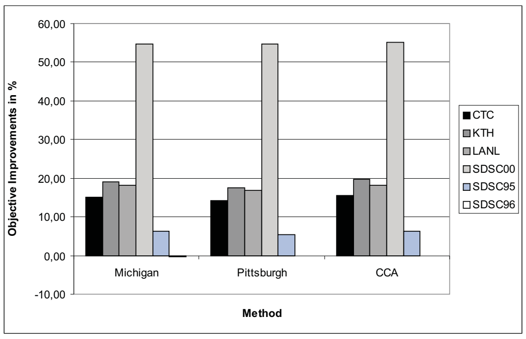


Figure 10.8: Relative objective improvements for the rule reduction attempts for the Pittsburgh approach.

and the SDSC00 workload traces some more rules are applied.



(a) SOPT(CTC) applied to all workloads. Note that the results for KTH and SDSC00 using the Pittsburgh approach are cut. The real results are for KTH: -151 % and for SDSC: -532 %.



(b) ALLOPT applied to all workloads.

Figure 10.9: The relative SOPT(CTC) and ALLOPT results for all scheduling strategies based on Genetic Fuzzy systems applied to all workload traces.

Figure 10.9a shows the achieved objective improvements for all workload traces by applying the scheduling strategy that was optimized for the CTC workload trace (SOPT(CTC)) using all three Genetic Fuzzy system approaches. The improvements for the CTC, the LANL, the SDSC95, and the SDSC96 workload traces are very similar for all three approaches. However, the optimized scheduling strategy using the Pittsburgh and the CTC workload trace fails for the KTH and the SDSC00 workload trace. All detailed results can be found in the Appendices K.2, L.2, and M.2.
Figure 10.9b presents the results that are achieved by using all workload traces during the optimization procedures together (ALLOPT). The detailed results are given in the Appendices K.3, L.3, and M.3. The quality of the best schedules for all workload traces is very similar. All three approaches show a robust behavior. The quality of the achieved schedules for all workloads increases. Using this approach, the scheduling objectives for the SDSC00 workload increase by more than 50 % in all three cases.

10.4 Comparison of all Scheduling Strategies

Figure 10.10 presents the scheduling objective function improvements of all used development methods compared with EASY for the CTC workload trace. All three approaches that are based on Genetic Fuzzy systems lead to scheduling strategies with the highest performance regarding the selected objective function. Furthermore, the scheduling quality of these three approaches is nearly identical. We would recommend the Pittsburgh approach for the optimization of local scheduling strategies as the computational effort to generate the corresponding scheduling strategy is much smaller in terms of necessary simulations compared with the Michigan approach and is easier to implement compared with the Cooperative Coevolutionary approach. Additionally, the generation of scheduling strategies that are optimized for workload traces individually should use one of the two presented approaches to reduce the number of necessary rules. This further stresses the usage of the Pittsburgh approach for individual optimizations as the rule reduction attempts are based on this approach.

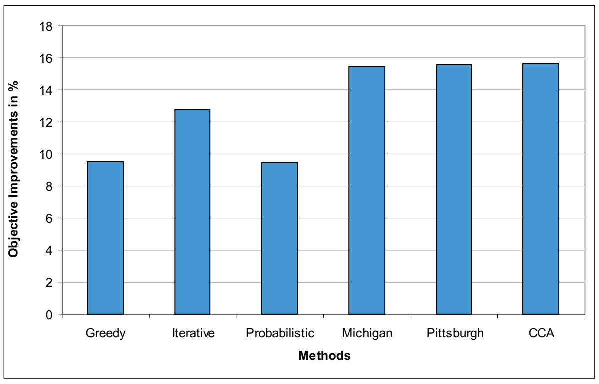


Figure 10.10: Relative objective improvements for all optimization methods optimized and applied for the CTC workload trace.

The iterative rule based system generates schedules that lead to a higher objective value than the Greedy approach and the probabilistic approache. However, the number of necessary simulations to develop a scheduling strategy based on the iterative approach is much higher than for the usage of the probabilistic approach. The Greedy scheduling strategy leads to similar results as achieved by the probabilistic approach. However, the Greedy strategy only needs 36 parameters to specify the whole scheduling strategy. This is the major advantage compared with all other solution approaches. In the case of the CTC workload trace, this simple strategy increases the objective already by about 10 % compared with EASY.

During all of our optimizations, we can observe the invariance of the utilization, see Figures 10.2, 10.5b, and 10.7b. This is quite interesting as the objective function does not include the utilization. This effect demonstrates that we are able to develop scheduling strategies that prioritize different user groups having a nearly constant utilization. This outcome was not expected. Consequently, the objective function definition can be focused on the preferences of user groups.

In Figure 10.11, we display the objective increases of all used approaches compared with EASY by optimizing the different strategies for the CTC workload and applying the resulting scheduling strategies to all workloads (SOPT(CTC)). The results clearly indicate that only the Michigan approach and the CCA approache should be used in such scenarios. All other

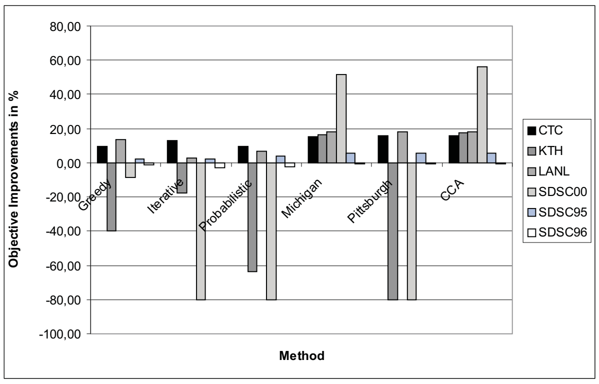


Figure 10.11: Relative objective improvements for all optimization methods and all workloads using the strategies optimized for the CTC workload trace (SOPT(CTC)). Note that some of the results are cut at -80 % as explained above.

approaches fail for at least the KTH and the SDSC00 workload traces. We assume that the Michigan approach with the 50 rules is better able to adapt to the different scheduling scenarios. The Pittsburgh approach only consists of 10 rules. As already shown, those 10 rules are enough to generate very good schedules for certain workloads individually. However, it seems that 10 rules are not enough to generate robust scheduling strategies. The CCA approach uses 10 rules for the sorting criterion and 10 rules for the scheduling algorithm. The achieved results indicate that this case is not too specialized on a certain workload but able to generate good schedules for most of the workloads. The quality of the generated schedules for the Michigan approach and the CCA approach are very similar. Thus, for the SOPT(CTC) scenario we recommend the CCA approach as the number of required simulations is in this case much smaller compared with the Michigan approach.

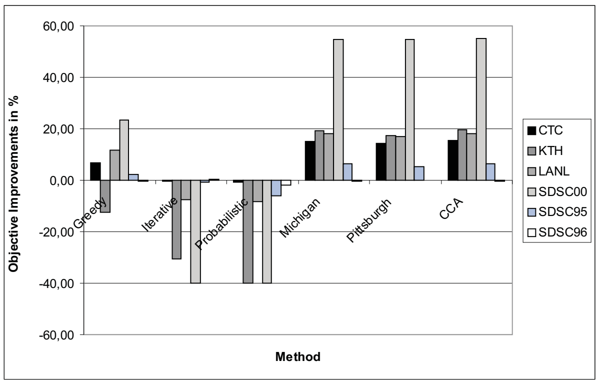


Figure 10.12: Relative objective improvements for all optimization methods and all workloads using the strategies optimized for all workload traces (ALLOPT). Note that some of the results are cut at -40 % as explained above.

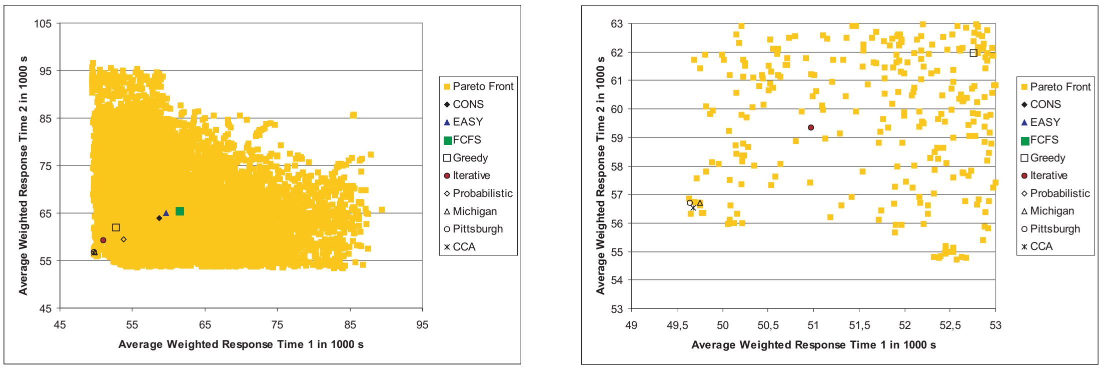
The objective improvements relative to EASY of all approaches for the ALLOPT scenario are given in Figure 10.12. Only the approaches that are based on Genetic Fuzzy systems are able to improve the scheduling outcome for all available workload traces. All other approaches fail for at least one of the workloads. As already discussed, the scheduling quality of all three

Genetic Fuzzy systems is nearly identical. Thus, we would recommend to use the Pittsburgh approach or the CCA approach to generate scheduling strategies by using the scheduling outcomes for all workload traces. The Pittsburgh approach and the CCA approach need significantly fewer simulations compared with the Michigan approach.

Overall, we prefer the CCA approach as this approach generates good scheduling strategies in all of our examined scenarios. The individual optimization based on a single workload trace is possible as well as the generation of a robust scheduling strategy. Such strategies are developed by applying an optimized strategy based on a single workload trace to other workloads or by optimizing a scheduling strategy based on the several workload traces.

The Michigan approach leads to similar results but requires more simulations compared with the CCA approach.

In all of our previous tables and figures, we only presented the relative improvements of the developed scheduling strategies compared to existing standard strategies. In the following, we present the scheduling quality of our generated scheduling strategies compared with the generated Pareto front. By this, we assume that our approximation of the Pareto fronts is close to the real Pareto front. This is reasonable as our approximation covers a wide area within the solution space and has a very good diversity. Thus, by presenting the approximated Pareto front in combination with the achieved scheduling results of the generated strategies we can compare the achieved scheduling results with the best achievable results.

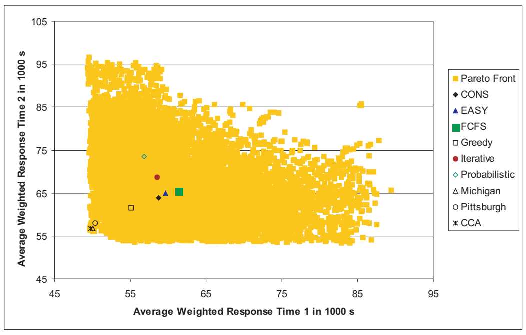


(a) All achieved optimization results. (b) Zoom into the area of best performance.

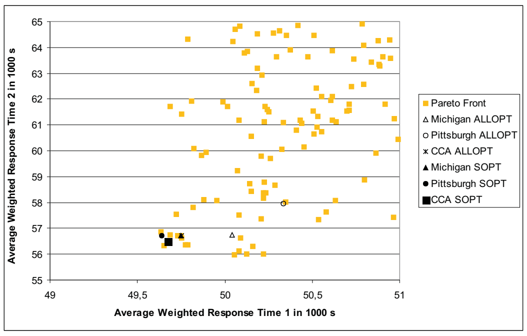
Figure 10.13: The approximated Pareto front of the CTC workload and the best results achieved by all optimization methods using the optimization for the CTC workload (SOPT(CTC)).

Figure 10.13a presents the scheduling results of our six approaches using SOPT(CTC) for the CTC workload trace in combination with the approximated Pareto front and the three standard scheduling strategies. As one can see, all of our approaches lead to scheduling strategies that perform better than the existing standard strategies. Furthermore, the three approaches that are based on Genetic Fuzzy systems are very near to the approximation of the optimal solution.

Figure 10.13b shows the most interesting area of Figure 10.13a. Here, we can see that the optimal solution is not reached but all three scheduling strategies based on Genetic Fuzzy systems are very near to this optimal solution. Furthermore, the Pittsburgh approach and CCA approach are slightly better than the Michigan approach.



(a) The Pareto front of the CTC workload and the best results achieved by all optimization methods using the combined optimization for all workloads (ALLOPT).



(b) Comparing the best solutions using ALLOPT and SOPT and the solutions of the Pareto front for the CTC workload trace.

Figure 10.14: Results for the ALLOPT and SOPT solutions with the CTC workload trace.

The outcome of the generation of scheduling strategies by optimizing the outcome for all workload traces together (ALLOPT) is shown in Figure 10.14a for the CTC workload trace. Of course, the results are not as good as in the SOPT(CTC) case as we do not optimize the strategies for the CTC workload trace only. This is shown in Figure 10.14b. Here, the solution generated by the Michigan approach with SOPT(CTC) is nearly identical with the ALLOPT solution using the CCA approach. Most of the resulting strategies based on ALLOPT achieve a better scheduling quality than the existing standard strategies. Furthermore, the strategies based on Genetic Fuzzy systems are very close to the approximation of the Pareto front. In opposite to the SOPT(CTC) scenario, neither the iterative approach nor the probabilistic approach are able to improve the scheduling results compared with the standard strategies, see Figure 10.14a. In this case, the Greedy strategy is more robust than those two rigid approaches.

Part III

Grid Scheduling

Chapter 11

Grid Scheduling

Grid computing extends the concept of massively parallel processing systems. This is meaningful, as computing power remains a limited resource. On the one hand, this is caused by new emerging complex applications which were enabled by the new technology. On the other hand, many existing applications grow in their demand for computing power. Grids can include many resources of different nature as, for instance, CPU, network, data, or software [101, 62]. Hence, Grids are more and more interesting to offer the user transparent access to those resources. This transparency is the reason to use the term "Grid". This term alludes to the Electrical Power Grid which provides all users with electrical power just on demand without requiring any deeper insight about how and where the power has actually been generated. Similarly, a Computational Grid is supposed to provide computational power on demand to all users without prior knowledge about the locations of the allocated resources. In general, Grids usually denote the sharing of geographically distributed resources that belong to different providers and reside in different administrative domains.

More specific, Grid computing is a method to execute computational jobs requiring a significant amount of computing resources and/or large sets of data. Contrary to single parallel processing environments and large heterogeneous distributed systems, a Computational Grid has many independent resource providers with different access policies. In addition to the size of such a Grid, the diversity of those policies leads to a very complex allocation task that should not be manually handled by the users and providers. This task does not only include the search for suitable machines but also the coordination of the actual job execution on the selected set of machines. Therefore, an efficient and flexible Grid scheduling system is required to manage the job requests of the users. Further, the independent resource providers typically want to maintain control of their Grid resources by use of local management systems. Those local management systems enable local providers to satisfy local users and user groups. To this end, the scheduling systems described in the previous chapters can be used and are of special interest. The different provider objectives increase the complexity of the Grid allocation task as those local management systems usually do not provide all system information due to their architecture or due to policy restrictions. Similarly, it is difficult to consider the requirements of individual jobs with complex workflows or complex user objectives. These scheduling objectives are a consequence of the differences in cost and quality of service among the machines. The workload of a Grid system is generated by independent users who submit their jobs over time. It is the task of the scheduling system to decide when and where a job is executed and to allocate resources to this job. As mentioned above, this process is subject to the job requirements, the users' objectives and the resource providers' policies to grant access to the resources. The various steps of performing the scheduling task have been outlined by Schopf [133]. These steps include the finding of suitable resources for such a request, the decision which of the actually available resources to choose, and the determination of the start of a particular application. As, at least temporarily, more requests are submitted than machines are available, this leads to machine conflicts which must be settled by the scheduling algorithm. Grid scheduling significantly differs from conventional job scheduling on parallel computer systems which has been addressed in the previous chapters. Although several core Grid services are already available, those higher-level services for coordinating the resource access are still missing. Even existing complete Grid systems, like, for instance, Condor, Legion, or Nimrod/G, deliver a full set of services only for specific configurations.

In this chapter, we analyze the requirements for scheduling methods in a Grid environment. This leads to implications for the architecture of a Grid scheduling system. Particularly, we discuss several scheduling algorithms with respect to their suitability for Grids. First, we describe how conventional scheduling strategies for parallel systems can be extended to include the Grid paradigm. As machine and job management problems in a Grid can be modelled as a utility market, it seems appropriate to also consider economic and market oriented methods. Therefore, we discuss those methods and present a market oriented scheduling strategy and its exemplary implementation. The different strategies are evaluated by use of simulations with trace based workloads for two scenarios. As a result, it is concluded that conventional and economic strategies are both suitable in general for Grid scheduling. Depending on the actual environment constraints in a real Grid system, both strategies have their advantages and their drawbacks, which are described in detail. Due to its higher flexibility, the economic approach seems to be the solution of choice for Grid scheduling in the long term.

The presented chapter represents the summary of the various Grid scheduling related publications by Ernemann et al. [42, 41, 43, 40, 48, 45, 44] or Franke et al. [83, 65] and serves as a good overview for related problem areas.

11.1 Introduction to Computational Grids

While some Computational Grids are based on machines within a single administrative domain, like a large company, most Computational Grids consist of resources with different providers. In the latter case, most providers are not willing to exclusively assign their resources to the Grid. For instance, a computer may temporarily be removed from a Grid to work solely on a local problem if there is a need for it. In order to react immediately in those situations, providers typically insist on local control over their resources, which is achieved by use of a local management system.

On the other hand, it would be a cumbersome and tedious task for a potential Grid user to manually select all resources needed to run his Grid application. To prevent those tasks from significantly slowing down the proliferation of Computational Grids, a specific Grid management system is needed. Ideally, such a Grid management system includes a separate scheduling layer that collects the Grid resources specified in a job request, considers all requirements, and interacts with the local scheduling systems of the individual Grid resources. Hence, the scheduling paradigm of a Grid management system will significantly deviate from that of local or centralized schedulers for large computer systems which typically have immediate access to all system information. Although it seems obvious that a Grid scheduler will consist of more than a single scheduling layer, the details of an appropriate scheduling architecture have not yet been established [140]. Nevertheless, it is clear that some layers are closer to the user (**higher-level scheduling instance**) while others are directly affiliated with the resource (**lower-level scheduling instance**). Of course, those different layers must exchange information among each other.

In general, Grids may not be restricted to two levels of scheduling layers but may be built hierarchically. For instance, a large Computational Grid may consist of several Sub-Grids. Each of those Sub-Grids has a separate Grid management system. The Grid scheduler of one Sub-Grid may decide to forward the request of a local user to another Grid scheduler from a second Sub-Grid. Then, this second Grid scheduler interacts with the local scheduling systems of the resources in its Sub-Grid. Clearly, three scheduling instances are involved in this situation. The Grid scheduler of the second Sub-Grid is a lower-level scheduling instance with respect to the Grid scheduler of the first Sub-Grid, while it is a higher-level instance in the communication process with the local scheduling systems of the second Sub-Grid. Of course, the scheduling instance in the lowest layer is always a local scheduling system.

Presently, a variety of different local scheduling systems, like PBS, LoadLeveler, LSF, or Condor [100], are installed in large computer systems. A Grid scheduling layer must exploit the capabilities of those local scheduling systems to make efficient use of the corresponding Grid resources. However, those capabilities are not the same for all local scheduling systems due to system heterogeneity.

All this leads to several general aspects that must be considered in designing a generic Grid scheduling environment, see also Czajkowski et al. [26]:

Site Autonomy All resources of a Grid are typically not owned and maintained by the same administrative instance.

Independent Schedulers A Grid scheduler has no exclusive control over all resources in a Grid. Therefore, the Grid scheduler must cooperate with the existing independent local schedulers.

Heterogeneous Substrate The resources usually have their own local management software with different features depending on the local policies and functionalities of the management software. Hence, the Grid scheduler has to cope with the limitations of the local management.

Online Problem The Grid scheduling takes place in an online environment where jobs are submitted at any time. Additionally, information on the current state of resources may be difficult to obtain and to keep up to date.

Scalability and Reliability As the Grid is intended to span over a very large number of systems, a Grid management system requires a high degree of scalability. The general Grid management must remain operable even if some components are impacted by a resource failure.

Variable Scheduling Objectives The scheduling objectives may vary for each available resource according to the provider's policy. In addition to the objectives of the provider the needs of the users must also be taken into account.

Co-allocation Some applications need several resources from different providers at the same time. The Grid scheduling system must be capable of coordinating these resources.

Resource Reservation Several complex applications require the reservation of resources in advance. Further, it is advantageous for the scheduler to consider system downtime or restricted access that is known beforehand. Finally, advance reservations are required for co-allocated resources or multi-site applications.

For the design and evaluation of Grid scheduling strategies, we distinguish two Grid computing scenarios.

1. **HPC Grids:** Cooperation between a limited or moderate number of computing sites with high performance computer systems and a dedicated user community.
2. **Global Grids:** Large-scale Grids with a diverse number of resources and independent users.

The first scenario is the typical use case of a Grid. The relatively small user community is part of a scientific community. Furthermore, those grids represent virtual organizations. This scenario is also applicable for large commercial companies in which existing computing resources, for instance, at different locations, are used efficiently.
The second scenario represents the idea of a global large-scale Grid infrastructure in which individual users have access to a large number of Grid resources. These resources belong to independent and typically unknown providers. This scenario is the generally discussed and anticipated long-term vision for Grid technology.
In the following, we analyze these two Grid scenarios. Both have different requirements which must be taken into account when designing a Grid scheduling system. For instance, the scalability issue is less important in the first scenario. Here, it may not be necessary to pay much attention to complex scheduling objectives including cost management and commercial Grid business models. Therefore, we may use a different approach on Grid scheduling in this scenario by extending existing scheduling methods for high performance computer systems. For evaluation purposes, however, we use the same generic Grid model for both scenarios which allows us to compare the presented methods later. But note that especially the second scenario covers a larger diversity of Grids.
Our Computational Grids consist of several independent computing resources which are linked by interconnection networks. The term *independent* emphasizes the fact that the participating resources are not controlled by a single entity and may be geographically distributed. While this definition for a Grid does not specify details of the individual resources in the network, in practice, these resources are often high performance computing components, like massive parallel processors, networks with high speed and high bandwidth, or large databases. Only the component at the user access point may not necessarily belong to this category. However, the user of such a Grid system may not be aware of the internal system structure but has the illusion of a single virtual machine. To support this concept, the Grid management system may use several components of the system concurrently to solve a large problem. This is called multi-site execution.
In our studies, we assume that each participating site has a single massively parallel machine that consists of several nodes. A parallel job can be allocated to any subset of nodes of a machine. This model comes reasonably close to a Grid environment consisting of real systems like an IBM RS/6000 scalable parallel computer, a Sun Enterprise 12000, or an HPC cluster. For simplicity, all machines and all nodes in this study are identical. The machines at the different sites only differ in the number of nodes. The existence of different resource types

would additionally limit the number of suitable machines for a job. In a real implementation, a preselection step is part of the Grid scheduling process and is normally executed before the actual scheduling takes place. After this preselection phase, the scheduler ideally chooses from several resources that are all suitable for the job request. In this study, we neglect this preselection step and focus on the remaining scheduling task. Therefore, in the following, it is assumed that all job parts could be executed on any node.

Like in the single massively parallel computer scenario of Chapter 4, the jobs are not preempted nor time-sharing is used. Thus, once started, a job runs until completion. Furthermore, we do not consider the case that a job exceeds its allocated time. After its submission, a job requests a fixed number of resources that are necessary for starting the job. This number is not changed during the execution of the job, that is, jobs are neither moldable nor malleable [58].

As Grid workloads are not yet available, we model a Grid scenario with the help of workloads for single parallel machines, see Section 4.5. Here, jobs are submitted by independent users at the local sites. This produces an incoming stream of jobs over time. Thus, we deal with an online scheduling problem without any knowledge on future job submissions. The scheduling system allocates resources for the jobs and determines their starting time. Then, the jobs are executed without any further user interaction.

In a real implementation, the job data must be transferred to the remote site before the execution. This transport of data requires additional time. This effect can often be hidden by Grid data management services which pre-fetch data before the execution and return output data afterwards. In this case the resulting overhead is not necessarily part of the scheduling process. For now, we neglect this data transport for the scheduling and discuss this decision in the evaluation following later.

11.2 Problem Classification

The general scheduling problem of Grid Computing is, regarding the general framework and notation introduced in Section 2.1, difficult to describe. The notation of scheduling systems is based on the assumption that a complex system and all available information can be explored by a single administrator. However, in the context of Grid Computing this is not valid, as none of the providers is able to access all information of all available machines. Thus, we need to extend the notation from Section 2.1. To this end, we introduce a new element within the β-field: *MP*. This reflects the fact that **M**ultiple **P**roviders (MP) are involved. Consequently, the scheduling objective cannot be optimized by a single provider but only by the cooperative behavior of all participating computing sites. Furthermore, the meaning of the β-field entry M_j must be specified. In the original work by Graham, M_j specifies that job j can only be executed on a subset of all m machines. In the Grid scenario these subsets M_j are normally given by the administrative domains as jobs can only be executed within one domain with the exception of a multi-site scheduling, see Section 11.3.

Consequently, the problem classification of Section 4.3 where we described the scheduling problem of a single massively parallel processing system by $P_m|\bar{r}_j, \bar{p}_j, m_j|C_{max}$ has to be extended to $P_m|\bar{r}_j, \bar{p}_j, m_j, M_j, MP|C_{max}$ for Grid Computing. We still assume that the makespan (C_{max}) is minimized. However, if the participating k providers have different objective functions $(f_1, f_2, \ldots, f_k)$, the scenario can be described by $P_m|\bar{r}_j, \bar{p}_j, m_j, M_j, MP|F_s(f_1, f_2, \ldots, f_k)$ regarding the notation for multi-objective scheduling problems, see Section 2.5.1.

11.3 Evaluation of Load-Sharing in Grids

Within this section, we address Grid scheduling strategies for the above mentioned Scenario 1. As already pointed out the different scheduling strategies are derived from conventional scheduling on parallel computers. In this first study, we consider different levels of cooperation methods between the computing sites and analyze the impact on the scheduling quality in terms of minimization of response times. The scheduling algorithms are mainly based on the well known First Come First Serve (FCFS) strategy and its derivatives, see Section 4.4.
We start by considering the following three different cases.

1. **Local Job Processing**

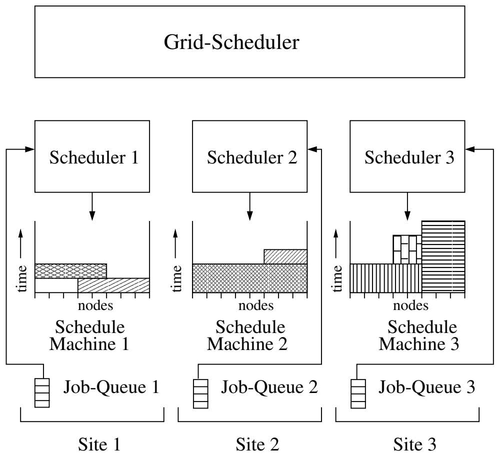


Figure 11.1: Sites Executing All Jobs Locally.

Here, the computing resources at a site are dedicated only to their local users (see Figure 11.1). Hence, each site has its own workload that is not shared with other sites. For our simulations, we use the EASY backfilling algorithm of Section 4.4.2.1 that has been implemented in several installations of high performance computers [144].

2. **Job Sharing**

In this case, all jobs are forwarded to a central Grid scheduler as seen in Figure 11.2. Note that this scheduler may be implemented in a distributed fashion. Such an implementation requires an additional management overhead for exchanging information about the current state of the queues and schedules in the system. Nevertheless, it knows about all submitted jobs. This is only possible in the HPC Grid Scenario 1 where the number of different sites is limited. However, in practice a local system may decide to forward only a subset of its local job to a Grid scheduler. The scheduling algorithms consist of two steps representing two layers of a Grid scheduler. In the first step, the machine is selected, while in the second step, the actual scheduling on the target machine is executed.

Step 1 - Machine Selection: There are several methods possible for selecting the target machine. For instance, other simulations [74] showed good results for a

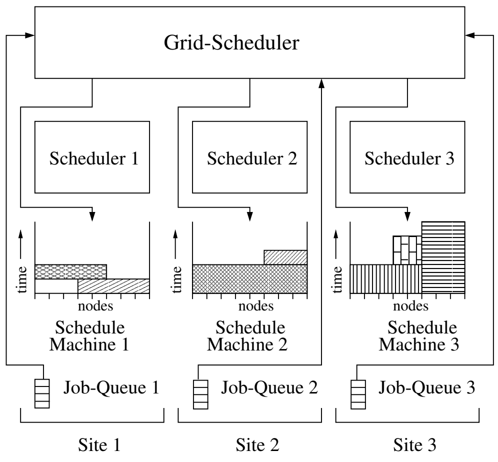


Figure 11.2: Sites Sharing Jobs and Resources.

selection strategy called *BestFit*. Here, the machine is selected on which the job would leave the least number of idle resources if it is started there as soon as possible.

Step 2 - Scheduling Algorithm: Again, EASY backfilling is used.

3. **Multi-Site Computing**

This case allows the concurrent execution of a job on more than one machine. It comes closest to the concept of a single large virtual machine while the usage of multi-site applications has been theoretically discussed for quite some time [15]. There are only few real multi-site applications in practice. Therefore, there are no data to determine the effect of multi-site execution on the execution time of a job.

In our model, the local schedulers forward all jobs to a Grid scheduler. This time, the Grid scheduler does not simply select a single site for job execution but may also split jobs to be executed across site boundaries (see Figure 11.3). Note that the job parts still run in parallel on the different sites.

Again, there are several strategies possible for multi-site scheduling. Here, we use a scheduler that first tries to find a site with sufficient currently idle resources for immediately starting the job. If such a machine is not available, the scheduler tries to allocate resources from different sites for the job. To this end, the sites are sorted in the descending order of idle resources which are then allocated in this order to the job. This way, the number of execution sites is minimized. If there are not enough free resources available for a job, it is queued and backfilling is later applied.

Spreading job parts over different sites usually produces an additional overhead. This overhead is a result of the process communication during run time which must be partially executed over the wide area network (WAN). However, as WAN networks become faster, this overhead may decrease over time. Further, it depends on the particular application. For those jobs with limited communication demand there is only a small impact. The overhead in a real Grid scenario highly depends on the job communication pattern and the network configuration between the sites. Due to the lack of available data, we

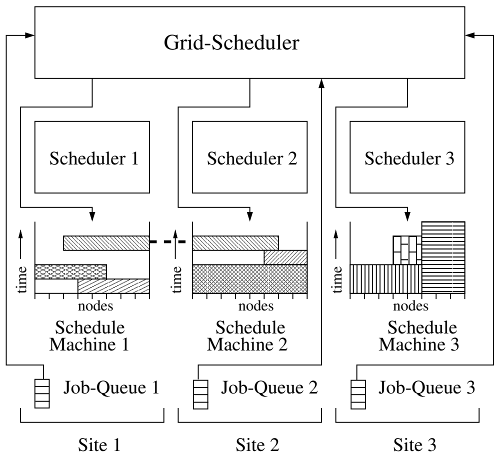


Figure 11.3: Support for Multi-Site Execution of Jobs.

assume a proportional increase for the execution time of all multi-site jobs. To this end, we model the influence of the overhead by extending the required execution time p_i to p_i^* for a job i that runs on multiple sites by a constant factor p:
$p_i^* = (1+p) \cdot p_i$ with $p = 0$ .. 300 % in steps of 5 %.
Note that without the introduction of any penalty for multi-site execution, the Grid would behave like a single large computer. Hence, multi-site scheduling will ideally outperform all other scheduling strategies.

Further, we also examined an improved multi-site strategy. In this algorithm, the earliest potential completion time of the job running on a single machine is compared to the completion time of a multi-site job execution with the additional overhead. The better alternative is chosen, see Figure 11.4. Note that this is only an estimation as the real starting time of the job may vary depending on the completion time of jobs ahead in the queue.

11.3.1 Machine Configurations

In our evaluation we considered several Grid configurations to determine the robustness of our results. In order to allow comparisons between the results of different configurations, we use the same workloads and require that the sum of all nodes in each configuration comprises exactly 512. Those nodes are partitioned into various machine configurations as shown in Table 11.1.
The configurations *m64-8*, *m128-4* and *m256-2* represent several sites with 8, 4 and 2 identical machines respectively. They are balanced as there is an equal number of nodes at each machine. The configurations *m384-6* and *m256-5* are examples of a large computing center with several smaller client sites.
Finally, the reference configuration *m512-1* consists of a single site with one large machine. In this case no Grid computing is used. This reference configuration yields the best scheduling output that is possible without partitioning the compute resources into several machines and without any overhead due to Grid scheduling.

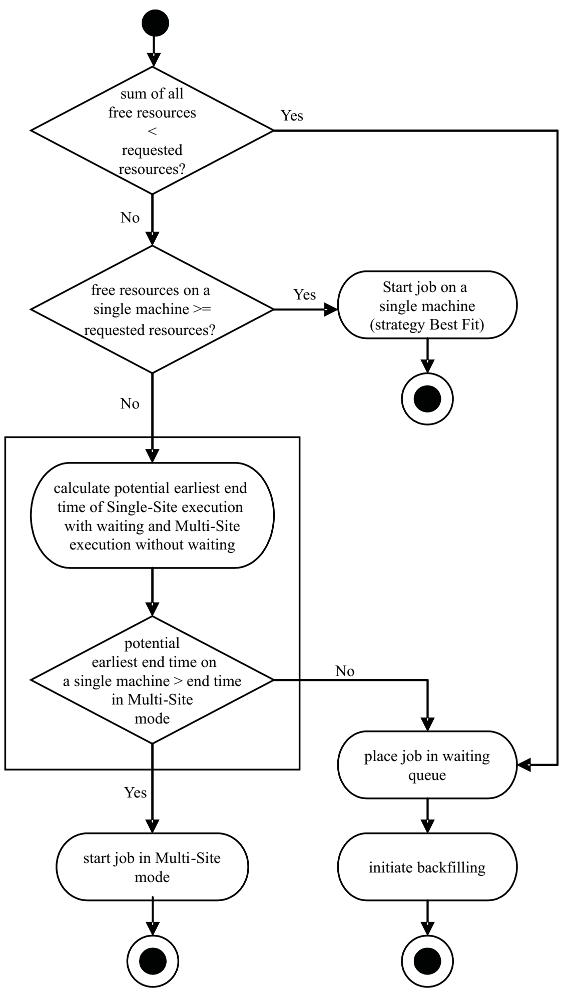

sum of all free resources < requested resources?
Yes
No
free resources on a single machine >= requested resources?
Yes
Start job on a single machine (strategy Best Fit)
No
calculate potential earliest end time of Single-Site execution with waiting and Multi-Site execution without waiting
potential earliest end time on a single machine > end time in Multi-Site mode
No
place job in waiting queue
Yes
start job in Multi-Site mode
initiate backfilling

Figure 11.4: Algorithm for Multi-Site Scheduling.

11.3.2 Workload Model

The evaluation of Grid scheduling strategies is not only based on the chosen machine configuration but also on the chosen workloads. In the following, we present the workload models which we used in the simulations of our evaluation.

Due to the lack of appropriate workload models for Computation Grids, we derive suitable input data from real job traces that are described in Section 4.5.2. In order to use these traces for our study, it is necessary to modify them to simulate submissions at independent sites with local users. To this end, the jobs from the real traces are assigned in a round-robin fashion to the different sites. However, as already stated in Chapter 5 many workloads favor jobs requiring 2^x nodes. Configurations with smaller machines would be put into disadvantage if the number of nodes on these machines is not a power of 2. To this end, we selected configurations with a total number of 512 nodes.

As discussed above, the quality of a scheduler highly depends on the used workload. To minimize the risk that singular and abnormal workload characteristics affect the validity of

identifier	configuration	max. size	sum
m64-18	$4 \cdot 64 + 6 \cdot 32 + 8 \cdot 8$	64	512
m64-8	$8 \cdot 64$	64	512
m128-4	$4 \cdot 128$	128	512
m256-2	$2 \cdot 256$	256	512
m256-5	$1 \cdot 256 + 4 \cdot 64$	256	512
m384-6	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	512
m512-1	$1 \cdot 512$	512	512

Table 11.1: Resource Configurations.

our results, the evaluation simulations have been done for 4 different workload sets which are all taken or derived from the same user group of the corresponding real installation. Each workload set consists of 10000 jobs. Furthermore, each workload covers a period of more than three months in real time.
Specifically, we use:

- three extracts of original CTC traces and
- one synthetic workload generated by a probabilistic workload model on the basis of the CTC traces. This workload model is described by Krallmann [94] and used in some other Grid scheduling publications, see, for example, Hamscher et al. [74]. It has been generated to prevent that potential singular effects in real traces, like a down-time of the real system, affect the accuracy of the results. However, as discussed in Chapter 5, such down times are often necessary to represent the real scheduling behavior of a system. Hence, the results of the scheduling using this generated trace have to be interpreted with care.

In our simulations, special care has to be taken for handling the "wide" jobs which are contained in the original workload traces. These are jobs within the workload which require more processing nodes than available on a machine. For instance, the widest job in the CTC traces requests 336 processing nodes. It cannot be executed in all of our Grid configurations. To permit a valid comparison of the simulation results, no job must be neglected. Therefore, we assume that the corresponding workloads of wide jobs are still generated at single sites. The wide jobs are split up into several parts of the local machine size to allow their execution (*Modification 1*). We also examined two other workload modifications. In one case, the job size is limited by the size of the largest machine in the configuration (*Modification 2*). Here, users can submit jobs that are wider than the local machine size. In another case, all jobs are split up into several parts with maximum 64 nodes to allow their execution on all examined configurations (*Modification 3*). Every configuration has a machine that consists of at least 64 nodes. To allow the comparison of different configurations for job sharing the following modifications have been applied to each aforementioned workload.
The workloads with modification *1* and *3* were executed in all 3 scheduling cases. The workloads with modification *2* were simulated for the *job-sharing* and *multi-site* case. Note that all of these modifications do not alter the overall amount of workload. With our modifications large jobs are split up, but they are still generated at the same site. Depending on the case, a user may submit jobs larger than locally available. The simulations allow the examination of the impact caused by wider jobs on the schedule.

identifier	description
W1	An extract of the original CTC traces from job 10000 to 20000.
W2	An extract of the original CTC traces from job 30000 to 40000.
W3	An extract of the original CTC traces from job 60000 to 70000.
W4	The synthetically generated workload derived from the CTC workload traces.

Table 11.2: The Used Workloads.

The input workloads are summarized in Table 11.2 and an identifier is introduced for each workload. The selection of workloads and the application to different machine configurations prevents the over- or underutilization of machines. If there are not enough jobs to be executed on a machine, the results may lead to wrong interpretations due to unrealistic modeling. The backlog of a scheduling system is a good indicator to check the utilization. The backlog is the workload that is queued at any time instant as there are not enough free resources to start the jobs. A small or even no backlog indicates that the system is not fully utilized. In our evaluations, we could show that all traces provide enough workload to keep a sufficient backlog on all systems (see also results from Hamscher et al. [74] and Ernemann et al. [42]). The average wait time of all jobs is between 45 minutes and more than two hours, which indicates the existence of an appropriate backlog. The examination of the actual schedules also shows that there was no job starvation. That is, the backlog did not grow significantly during the simulation, which would have been the case if the machines were not capable to master the workload.

11.3.3 Simulation Results

For the evaluation of the different structures and algorithms, discrete event simulations have been performed, see also [42, 41, 40]. We use the average weighted response time (AWRT), see Equation 4.2 on page 46, as the measure in this study.

11.3.3.1 Job-Sharing

The usage of job-sharing improved the average weighted response time in all examined machine configurations and in all workloads in comparison to the local job execution. In the configuration *m128-4*, for example, the improvement is over 50 % (see Figure 11.5). The achieved results are similar for all other simulations. As mentioned before large jobs that are wider than the local machine have been split up into smaller jobs which are sequentially executed on the local system. This leads to an increase of the AWRT in comparison to the original local job execution as the workload of these wide jobs is still generated and submitted at the same time as the original wide job, but the sequential execution leads to a delay of job parts. Job-sharing on the other hand allows the transfer of jobs to remote machines.

11.3.3.2 Multi-Site Computing

Next, we examine the influence of using multi-site execution. The results show that further improvements on the AWRT can be achieved in comparison to job-sharing. As a reference, the result for a single machine with 512 nodes is used (Figure 11.5). As expected, the average weighted response time without overhead for multi-site is near to the *m512-1* result, see

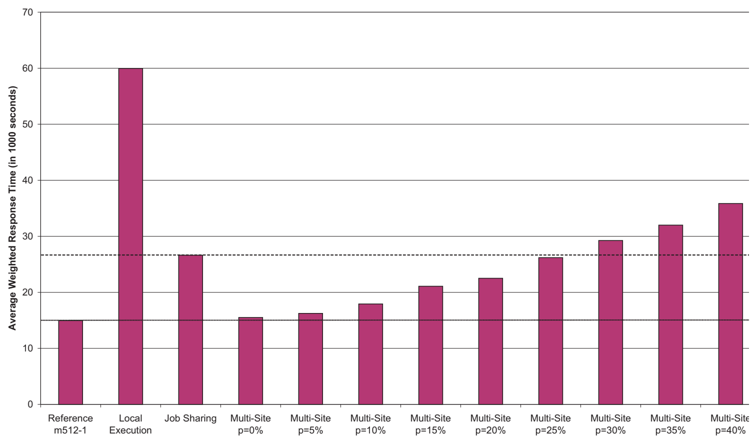


Figure 11.5: Average Weighted Response Time for the *m128-4* Configuration and Workload *W4* with Modification *2*.

Figure 11.5. The difference is caused as the multi-site strategy is quite simple and does not fully exploit the potential of all available machines.

Moreover, multi-site execution is also beneficial compared to job-sharing even for an overhead of about 25 % on the execution time (Figure 11.5). Similar results are achieved for other configurations, see Ernemann et. al. [41, 42]. The above mentioned results, as given in Figure 11.5, show the least effective improvements in comparison to other examined workloads. The average weighted response time in other configurations delivers even better results. Therefore, the tolerable overhead on multi-site executed jobs can even be larger.

Examining the sensitivity on different machine configurations, the simulation results show that configurations with larger machines provide significantly better scheduling results than machine configurations that are higher fragmented, see Figure 11.6 for all machine configurations and workload W4.

The increase of the AWRT results from two effects. First, the overall workload increases as all multi-site jobs have a longer execution time due to the overhead (p). Second, we can make the observation that the number of multi-site jobs increases with additional overhead. Table 11.3 shows the number of multi-site jobs for machine configuration m128-4 for different workloads and different multi-site parameters p. The effect can be seen for all workloads. This process is not monotone, but the difference between the number of multi-site jobs for p=0 % and p=60 % is always at least 30 %. This behavior results from the scheduling policy to schedule all jobs as soon as possible after submission. With increasing p, the execution in single-site mode becomes more beneficial compared with the multi-site execution. However, each job must be started as soon as possible. Thus, if multi-site executions are initiated, the increased execution time of all those multi-site jobs results in a reduction of the number of idle machines within the schedule. Therefore, the probability of free resources within one machine decreases as well. Hence, more jobs are started in multi-site mode.

The previous results show that jobs running in multi-site mode are in the majority jobs with a higher resource demand [42, 41]. This can be concluded because 5 % to 10 % of all jobs

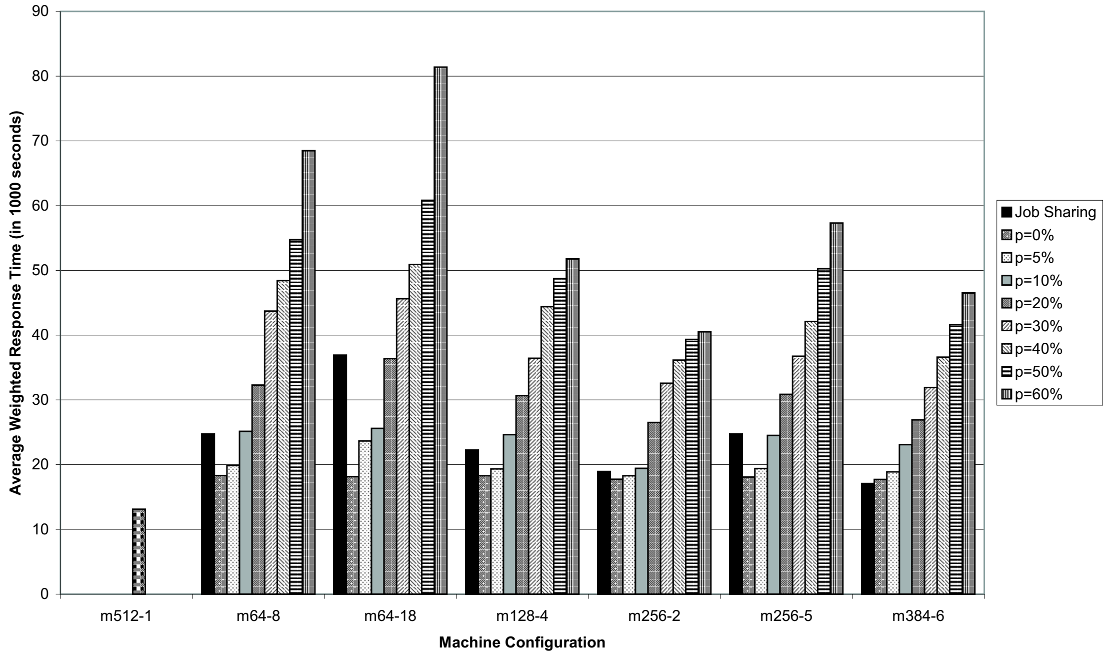


Figure 11.6: The Average Weighted Response Time in Seconds for Workload *W4* and all Machine Configurations and Multi-Site Scheduling.

are multi-site jobs while they are responsible for about 20 % to 40 % of the whole amount of workload depending on the used job trace. This effect even increases for a higher multi-site overhead (p) and is responsible for the above mentioned results. However, for machine configuration m384-6, job sharing always yields better results than multi-site scheduling. This effect results from the algorithm for multi-site scheduling. Here, a job that is considered for multi-site execution can be started earlier than previously submitted but not yet started jobs. For the completion of those waiting jobs the algorithm determines that it is beneficial to wait until enough resources are available on a single machine. Thus, the execution time does not increase for those jobs. In that case, this strategy does not consider backfilling for the waiting jobs.

The job sharing and multi-site execution of jobs in configurations with bigger machines is advantageous to configurations with smaller machines. In our example, the m64-8 configurations produce up to 25 % better AWRT results in comparison to m64-18. The comparison between the configurations m64-8, m128-4 and m256-2 shows that the use of bigger machines produces favorably better scheduling results.

Figure 11.7 shows the average weighted response time for the multi-site scheduling that calculates the estimated completion time of all jobs for the single-site and the multi-site execution before starting a job in multi-site mode. The improvements can be seen in comparison to Figure 11.6. In general, the average weighted response time is lowered for each configuration by the application of this improved strategy. That is, the Grid in these settings can handle multi-site jobs with a higher penalty while still improving against the job sharing scheduling case. However, as it can be seen in the figure, for configurations with large parallel machines

file	W1 $[jobs] \hat{=} [10^{-2}\%]$	W2 $[jobs] \hat{=} [10^{-2}\%]$	W3 $[jobs] \hat{=} [10^{-2}\%]$	W4 $[jobs] \hat{=} [10^{-2}\%]$
p=0 %	539	431	840	774
p=5 %	543	436	850	769
p=10 %	582	448	928	827
p=15 %	572	490	917	906
p=20 %	583	546	946	946
p=25 %	601	567	951	1042
p=30 %	622	521	979	1026
p=35 %	637	523	1036	1063
p=40 %	647	534	1106	1012
p=45 %	673	597	1008	1181
p=50 %	755	579	1029	1188
p=55 %	748	578	1086	1233
p=60 %	746	638	1114	1177

Table 11.3: Number of Multi-Site Jobs for Different Workloads and Different Parameters Using Machine Configuration *m128-4*.

job sharing often still yields better results than multi-site computing.
Note that in resource configuration *m384-6* multi-site execution is not necessary, as the largest job within the workload requests only 336 nodes and can therefore be executed on the machine with 384 nodes. Even an overhead of 300 % in resource configuration *m384-6* yields 4 % execution of all jobs in multi-site mode. This is only a 42 % reduction compared to an overhead of 30 % in the same configuration, whereas the resource consumption of the multi-site jobs decreases to 30 % in the same case. Overall, increasing the overhead ten times from 30 % to 300 % only results in an increase of the average weighted response time of about 7 % in this configuration.
Note that we do not conclude that multi-site is suitable for all applications. In terms of latency WAN networks are in the order of 2-3 magnitudes slower than common fast interconnection networks between nodes inside a parallel computer, e.g. an IBM SP Switch. Thus, the actual overhead caused by multi-site may be much higher. The question if multi-site execution is suitable, depends on many factors as e.g. the actual communication pattern, the requirement in data I/O of input/output data. Nevertheless, the results indicate that multi-site execution may be beneficial in terms of response time reduction for applications with a limited demand in communication as presented in our results.
Our evaluations shows good results for the average weighted response time. Other examinations not presented here show also that high utilization of the machines were achieved.

11.4 Market-Oriented Scheduling

In the following we examine the Global Grid scenario in more detail. In such a Grid environment, there are much more different users and providers which typically do not know each other. Here, the complete scheduling considerations presented in Section 11.1 are taken into account. While the optimization goal for a single parallel machine or in a small Grid is usu-

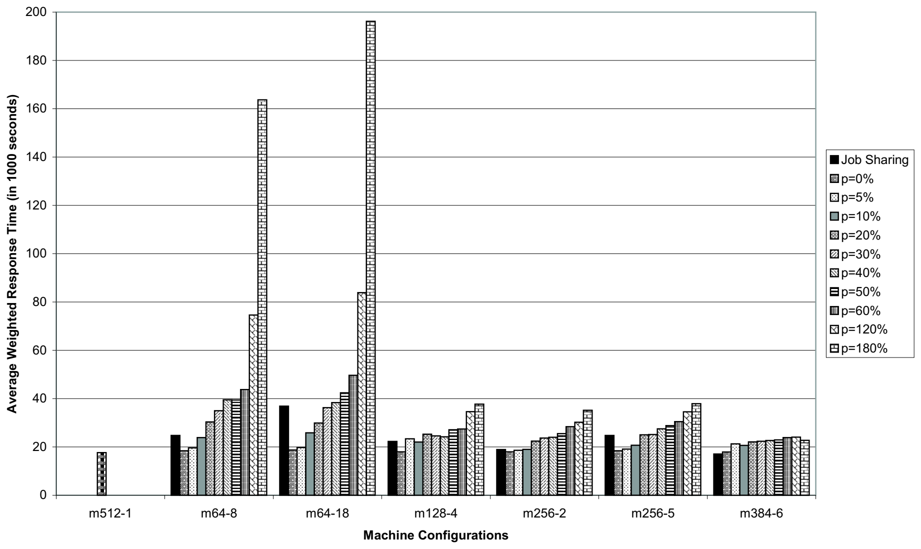


Figure 11.7: Comparison of Adaptive Schedules for Workload *W4*.

ally the minimization of the completion time of a computational job, in a large-scale Grid we might encounter more complex scheduling objectives. Here, various criteria have to be considered like, for instance, cost, quality of service or additional time constraints, as for example given start and completion time limits. The examined algorithms in the previous sections are mostly derivations of list schedulers that are usually predestined for a single overall objective.

For the Global Grid scenario, other scheduling approaches are necessary that may deal better with different user objectives as well as provider and resource policies. The independent users or resource providers in future Grids can be compared to a market in which independent parties trade for goods; in our case a good is the node allocation for a certain time. This is a typical application scenario in which economic models are used. In the following, we want to address the idea of applying economic models to the scheduling task. To this end, a market-economic method for Grid scheduling is presented and analyzed [43]. In our study we use the same evaluation process from the previous sections. However, the quality of an economic scheduling process is more difficult to measure as the former single objective is now dependent on the specific and variable objective formulation and the corresponding cost-metric. While there is a high degree of freedom in these processes, heuristics are often applied to limit the necessary time to exert the market methods. As neither information about Grid workloads nor user objectives, and cost models are available, we compare the market-oriented scheduling strategies with the conventional scheduling strategies as presented in Section 11.3. Therefore, we consider again only the average weighted response time minimization in our simulations. However, in this case, we do not exploit all additional features of the economic model like the support for variable objective formulation.

11.4.1 Market Methods

Market methods for scheduling computational power have been subject of research for quite some time. Such economic methods have also been applied in various contexts including Grids, see the references given by Buyya [18, 19] or by Cheliothis and Kenyon [91]. The supply and demand mechanisms provide the possibility to optimize different objectives of the market participants. It is expected that such methods provide high robustness and flexibility in case of failures and a high adaptability during changes.

It has to be emphasized that a market method is an equilibrium protocol and not a complete algorithm. Various methods, like the execution of *auctions* [162], have been developed to obtain this equilibrium. Common examples for auctions are: the English Auction, the Dutch Auction, the Sealed Bid Auction and the Double Auction. More details about the general equilibrium and its existence are given by Ygge [166].

In the following, we present our economic scheduling method for Grids. In contrast to the above mentioned approaches, we consider the flexible definition of scheduling objectives for each individual job request and each resource offer. Earlier applications of similar methods can be found in the *WALRAS* method and in the *Enterprise* method. As our strategy is derived from these methods, we briefly introduce their main concepts.

The *WALRAS* method is a classic approach that translates a complex, distributed problem into an equilibrium problem. It is based on the assumption of *perfect competition*, that is, the agents representing the market participants do not try to manipulate the prices with the help of speculation. To solve the equilibrium problem, the *WALRAS* method uses a *Double Auction*. During that process, all agents send their utility functions for individual goods to a central auctioneer who calculates the equilibrium prices. A separate auction is started for every good. At the end, the resulting prices are transmitted back to all agents. As the utilities of several goods of an agent may not be independent from each other, the agents can react on the new equilibrium prices by re-adjusting their utility functions. Subsequently, the process starts again. This iteration is repeated until the equilibrium prices are stabilized. As we will show later, in our application we substitute the central auctioneer by a decentralized equilibration algorithm.

The *Enterprise* [104] system is another example for market methods. Here, machines create offers for jobs to be run on those machines. To this end, all jobs describe their necessary environment in detail. After all machines have generated their offers, the jobs select between these offers. The machine that provides the shortest response time has the highest priority and will be chosen by the job. Within the Enterprise system all machines have the same priority scheme which favors jobs with a shorter run time. Although our model is based on this Enterprise method, it is neither restricted to a single objective function nor does it require that all machines share the same objective function.

11.4.2 Economic Scheduling Model

In contrast to Buyya et al. [17, 18, 19] or Waldspurger [161], our scheduling model does not rely on a single central scheduling instance. Moreover, we use a peer-to-peer like approach in which all participants act independently and may have different objectives and policies. Our method also supports the co-allocation of distributed resources in different domains. This feature is useful for multi-site job execution.

The general application flow in our grid job scheduling infrastructure is presented in Figure

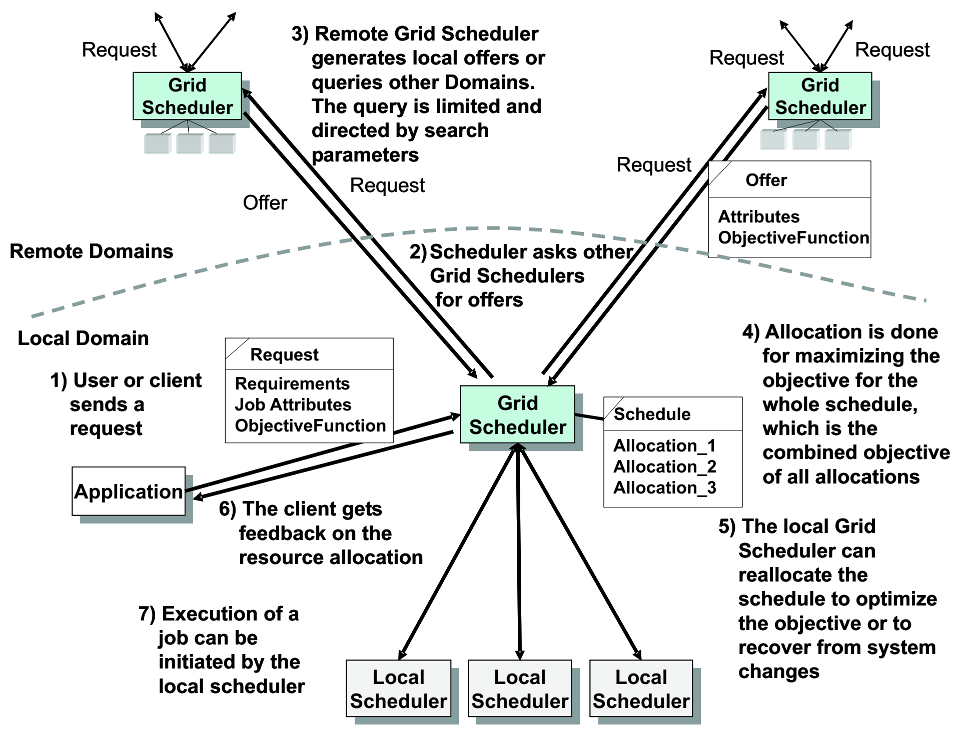

Request
Grid Scheduler
3) Remote Grid Scheduler generates local offers or queries other Domains. The query is limited and directed by search parameters
Request
Request
Grid Scheduler
Request
Offer
Request
Offer
Attributes
ObjectiveFunction
Remote Domains
2) Scheduler asks other Grid Schedulers for offers
Local Domain
4) Allocation is done for maximizing the objective for the whole schedule, which is the combined objective of all allocations
1) User or client sends a request
Request
Requirements
Job Attributes
ObjectiveFunction
Grid Scheduler
Schedule
Allocation_1
Allocation_2
Allocation_3
Application
6) The client gets feedback on the resource allocation
5) The local Grid Scheduler can reallocate the schedule to optimize the objective or to recover from system changes
7) Execution of a job can be initiated by the local scheduler
Local Scheduler
Local Scheduler
Local Scheduler

Figure 11.8: Scheduling Steps.

11.8. We assume that each user has access to at least one computing site which may be his local resource provider. In our scheduling model, a user submits his job request to the scheduler of his local domain. The scheduler first analyzes this request and, if possible, creates new offers for all local machines. After this step, a first selection takes place in which only the best local offer is selected. The remaining request is forwarded to the schedulers of other known domains. This is possible as long as the search depth of involved domains for this request is not exceeded and the *time to live* value for this request is still valid. Many strategies are possible to determine the domains to which the request should be forwarded.

This peer-to-peer approach allows the exploitation of network locality as users can query their local or nearest domain first and the request is then forwarded to related sites. To this end, each domain manages a list of other domains it may query for offers. Such a list can be manually or automatically maintained with the option to consider a network structure or a logical hierarchy. Furthermore, it is possible to include information services that provide addresses for domains and information on the available resources. Note that the presented model can be applied to Grid infrastructures with dedicated information services as well as to completely decentralized peer-to-peer configurations.

The remote domains generate new offers and return their best offer to the requesting domain. A domain does not answer to a request for a job if this request has already been processed before, that is, a domain answers any request for a job only once in case the same request may be forwarded from different domains. Afterwards, a second selection process takes place in order to find the best offers among the returned results of this particular domain.

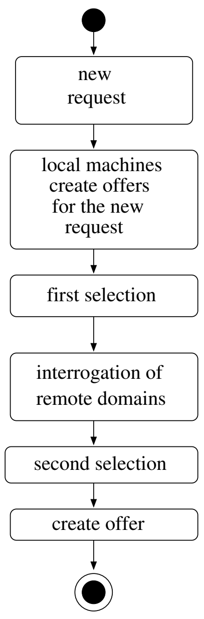

new request
local machines create offers for the new request
first selection
interrogation of remote domains
second selection
create offer

Figure 11.9: General Application Flow.

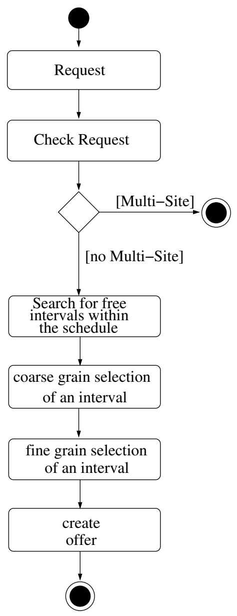

Request
Check Request
[Multi−Site]
[no Multi−Site]
Search for free intervals within the schedule
coarse grain selection of an interval
fine grain selection of an interval
create offer

Figure 11.10: Local Offer Creation.

11.4.3 Economic Scheduling Algorithm

This section includes a description of the scheduling algorithm that has been implemented for the presented infrastructure. The general application flow can be seen in Figure 11.9. Note that this method is an auction with neither a central nor a de-central auctioneer. Moreover, the different objective functions of all participants are used for the equilibration process. For each potential offer o of request j, the utility value $UV_{j,o}$ is evaluated and returned to the originating domain that first has received the user's request. The utility values are calculated by the user supplied utility function UF_j which can be extracted from the job and offer parameters. In addition, the machine value $MV_{i,j}$ of the corresponding machine i can be included.

$$UV_{j,o} = UF_j(MV_{i,j}, o); \qquad MV_{i,j} = MF_i(j)$$

This machine value is derived from the machine objective function MF. The originating domain selects the offer with the highest utility value $UV_{j,o}$. In principle, this domain acts as an auctioneer. A more detailed explanation of the local offer generation is given in Figure 11.10. Within the *Check Request* phase, it is determined if either the best offer will be automatically selected or if the user is going to select the best offer interactively among a given number of possible offers. In the same step, it is checked whether the user supplied budget is sufficient in order to process the job at the local machines within the domain. Additionally, it is determined whether the local resources meet the requirements of the request. Next, the necessary scheduling parameters are extracted. The parameters include the earliest start time of the job, the deadline, the maximum search time, the time until the resources will be reserved for the job (reservation time), the expected run time, and the number of required machines. The utility function is another parameter which is applied in the further selection process.

If not enough resources can be found during the *Check Request* phase, but all other requirements can be satisfied by the local resources, a *multi-site scheduling* can be initiated to gather the additional resources which are necessary to satisfy the request. The ability of integrating such co-allocation strategies into this model will later be discussed in more detail, see Section 11.4.4.

The next step *Search for idle intervals within the schedule* tries to find all idle time intervals within the requested time frame on the suitable resources. For a simple example assume a parallel computer with dedicated nodes. An example schedule is given in Figure 11.11. The black areas within the schedule are already allocated by other jobs. Now, a new incoming job requests three nodes and has an earliest start time A, a deadline D, and a run time less than $(C - B)$. First, idle time intervals are extracted for each node. Next, these elements are combined in order to find possible solutions. To this end, a list is created with triples of the form {time, node number, d}, where d equals (+1) if the node with the specified number is free at the examined time; d equals (-1) otherwise.

This generated list (sourceList) is used to find possible solutions as shown in the pseudo code of Algorithm 11.1.

Algorithm 11.1 Algorithm for Creating Possible Offers.

```
Require: sourceList: list of triples (time,node,+/-1) sorted by increasing time
  currentListOfFreeMachines = ∅;
  time t = smallest time in sourceList;
  while ((not enough offers found) AND (sourceList not empty)) do
    tripleList = ∅;
    tripleList = get and remove all next elements from sourceList with time t;
    test for all elements in currentListOfFreeMachines whether the difference between the
    beginning of the idle interval and time t is greater than or equal to the run time of the
    job;
    if (number of elements in currentListOfFreeMachines, which fulfill the time condition, is
    greater than or equal to the needed number of nodes) then
      create an offer from the elements of the currentListOfFreeMachines;
    end if
    according to value d add (+1) or remove (-1) all elements in tripleList to or from cur-
    rentListOfFreeMachines;
    t = next time t in sourceList;
  end while
```

The given algorithm creates possible offers that are specified by start, end, run time and the requested number of nodes. Note that we have not yet shown how the offer is created from the node elements of idle time intervals in this list. This is achieved by the following algorithm. This algorithm has the goal to determine a set of usable resources by using a list of idle time intervals of resources. Note that the idle times of different resource elements may have different start and end times. The resulting possible node allocations are characterized by an earliest start time and a latest end time. To this end, a derivation of bucket sort is used. In the first step, all intervals of idle resources with the same start time are collected in the same bucket. In the second step, for each bucket the elements with the same end time are collected in new buckets. At the end, each bucket consists of a list of resources that are available between the same start and end time.

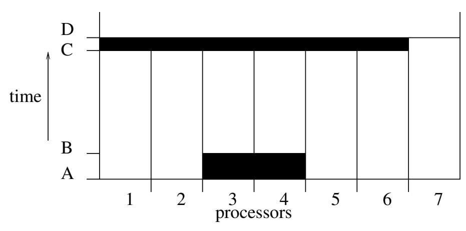


Figure 11.11: Start Situation.

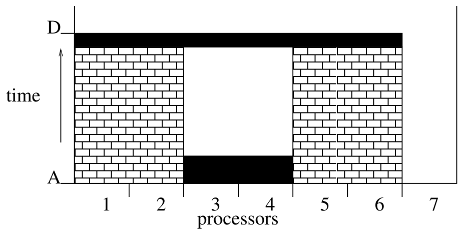


Figure 11.12: Bucket 1.

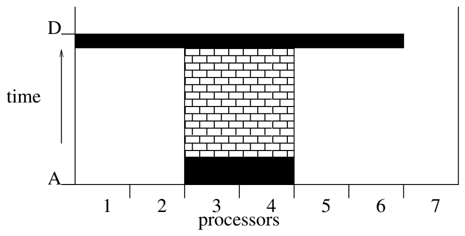


Figure 11.13: Bucket 2.

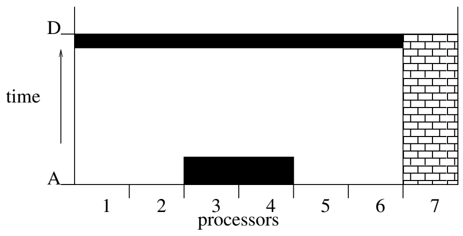


Figure 11.14: Bucket 3.

For the example above, the algorithm creates three buckets as shown in Figures 11.12, 11.13, and 11.14. After the creations of these buckets, suitable offers are generated either with elements from one bucket if the bucket consists of enough resources or by combining elements of different buckets. When using elements from different buckets with potentially different start and end times, the latest start time and the earliest end time must be calculated. In our example only Bucket 1 can satisfy the requirements alone and therefore an offer can be built by using, for instance, Resources 1, 2, and 5.

In order to generate additional offers, all buckets which alone can satisfy a request with its own elements are modified to contain one resource less than the required number. Afterwards, the previous process of offer generation is repeated again. If a bucket does not contain enough elements to satisfy a request, all elements of this bucket are taken into a new bucket. For our example, this is true for the Buckets 2 and 3. If not enough offers have been found and no bucket is further available, it is checked whether the remaining number of elements in all buckets combined suffice the requested number of nodes. If this is the case, the bucket elements are combined and the start and end times are adjusted to the latest start time and the earliest end time. For our example, we use the elements {1,2} from Bucket 1, the elements {3,4} from Bucket 2, and the element 7 from the last bucket. The additional offers can only be generated by using those selected elements. Thus, the set of solutions including the combination from Bucket 1 is: {{1,2,5}, {1,2,3}, {1,2,4}, {1,2,7}, {1,3,4}, {1,3,7}, {1,4,7}, {2,3,4}, {2,3,7}, {3,4,7}}.

After finishing the phase *Search for idle intervals within the schedule* from Figure 11.10, a grain selection of one of these intervals is initiated in the next step. In general, a large number of solutions could be created with only small changes for the start and end time and then selecting the interval with the highest utility value. Due to the time constraints of the scheduling algorithm, this is not feasible in practice. Therefore, we use a heuristic that first selects several initial combinations. The start and end times for these combinations are modified to improve the utility value. The combination with the highest utility value is selected as the resulting offer (in phase *fine selection of an interval* in Figure 11.10). This approach can be parameterized by the number of steps denoting the number of different start and end times within the given time interval. Note that we do not require the utility function to be monotone. Therefore, this selection process is a heuristic approach.

After finishing the algorithm in this last step, several possible offers have been generated.

We have not yet specified the utility functions of the machine providers and the machine users. In our implementation, any mathematical formula, using all valid time and resource variables, is supported. Overall, in our approach among the offers we select the solution with the highest value for the user's utility function. Note that this process does not necessarily yield the global maximum among all possible offers. The linkage to the objective function of the machine provider is created by the price for the machine usage which equals the machine provider's utility function. The user can include this price in his utility function.

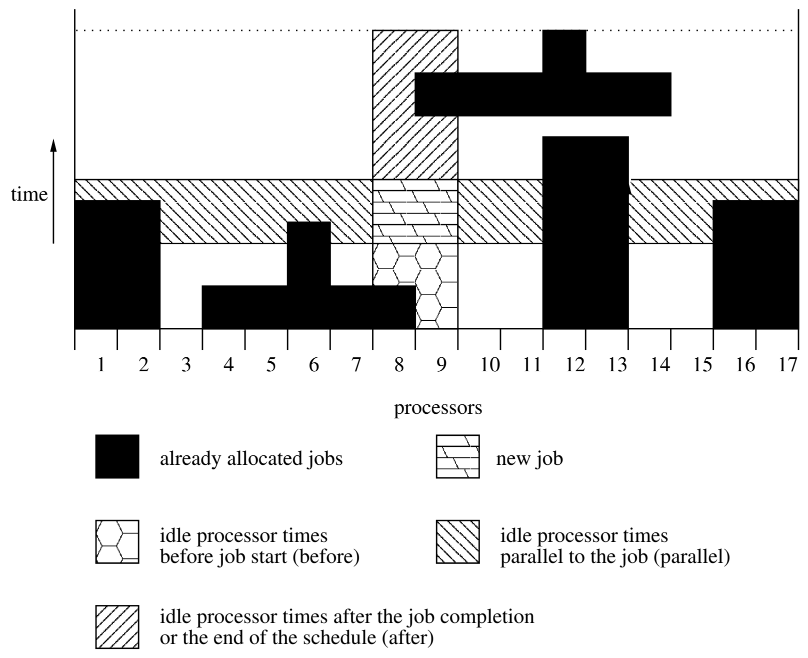

time
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
processors
already allocated jobs
new job
idle processor times before job start (before)
idle processor times parallel to the job (parallel)
idle processor times after the job completion or the end of the schedule (after)

Figure 11.15: Parameters for the Calculation of the Provider Utility Function.

The provider of the machine can formulate a utility function in which additional variables can be used that reflect the characteristics of the current schedule. Figure 11.15 shows those variables that are used in our implementation. The variable *before*, in Figure 11.15, specifies the processor times in the schedule in which the corresponding resources (nodes) are unused before the job allocation. The variable *after* determines the processor times of unused resources after the job end to the start of next job on the according resources or to the end of the schedule. The variable *parallel* specifies the concurrently idle resource times during the allocation of the job. In a graphical depiction of a schedule as in Figure 11.15, this is the idle area parallel to the job for all nodes of the machine. The variable *utilization* specifies the utilization of the machine if the job is allocated. This is defined by the ratio of the sum of all allocated processor times and the whole available processor times from the current time instant to the end of the schedule.

11.4.4 Co-Allocation (Multi-Site Scheduling)

In the following, we cover the situation when none of the local machines is alone able to compute the job. In such a case, a multi-site allocation is sought by co-allocating resources from different machines. As shown in Figure 11.10, a domain might request resources from other domains. This process is only initiated if there is no single machine in any domain able to satisfy the request. The main problem of multi-site scheduling is to find several appropriate resources to execute the whole job in parallel. Note that the different machine providers within market-oriented Grid scenarios do not have any insights regarding the allocation of machines of other providers. Hence, the missing resources are queried from other domains by additional requests for fixed time frames. To this end, several different start times are tried within the possible time interval given in the job request. The process finishes if a solution is found or a time limit is reached.

In our evaluation we used this co-allocation mechanism for executing computational jobs spread over different machines. However, this approach can easily be extended for co-allocating and coordinating different resource types. The same method can be applied to make reservation of network bandwidth, storage or software in conjunction with a computational job. The need for more complex coordination can be found in workflows in which different job steps may have dependencies. Within the presented economic model, the capability to individually request certain resource offers can be used to build sophisticated Grid schedulers for such coordination tasks.

11.4.5 Request and Offer Description

A basic component of an economically driven approach is a description language which is used to specify such requests for resources and the corresponding offers. As mentioned before, it is essential that requests for offers are very flexible. In the following, we briefly show a simple example language which we used for the implementation in our economic scheduling model. It will be essential for future Grid systems to support such communication protocols. This request and offer description is presented in more detail by Ernemann et al. [48, 43].

Our example of a description language allows arbitrary attributes which are specified as nested key value pairs in combination with the ability to specify several cases and constraints. Only a few keys are specific for the management and scheduling environment. The description language is used for requests, as well as for offers and resource descriptions. The syntax is very similar in all cases. We allow complex statements in the formulation of the values. This includes expressions and conditions to allow parameterized attribute specifications that can be evaluated at run time. Examples are the utility function or the job length, which may be derived from other attributes as the job cost or the available processor number/speed.

To better illustrate our description language, we give a brief example for a request formulation in Figure 11.16. The example request includes assignments for the keys *Hops* to *JobBudget*. The utility value depends on two conditions. The job requires either Linux or AIX as operating system and needs between 8 to 32 nodes for Linux, respectively 64 nodes for AIX. For Linux, the (*UtilityValue*) is defined as the negative value of *(-StartTime)*. As the system tries to maximize the *UtilityValue* the job should be started as soon as possible. Furthermore, the job has a parameterized definition of the run time. For AIX, the minimization of the job costs is anticipated in this example.

```
REQUEST "Req001" {
  KEY "Hops" {VALUE "HOPS" {2}}
  KEY "MaxOfferNumber" {VALUE "MaxOfferNumber" {5}}
  KEY "StartTime" {VALUE "StartTime" {900000}}
  KEY "EndTime" {VALUE "EndTime" {900028}}
  KEY "SearchTime" {VALUE "SearchTime" {899956}}
  KEY "JobBudget" {VALUE "JobBudget" {900.89}}
  KEY "Utility" {
     ELEMENT 1 {
       CONDITION{ (OperationSystem EQ "Linux")
         && ((NumberOfProcessors >= 8) && (NumberOfProcessor <= 32))}
       VALUE "UtilityValue" {-StartTime}
       VALUE "RunTime" {43*NumberOfProcessor}
     }
     ELEMENT 2 {
       CONDITION{(OperationSystem EQ "AIX")
         && ((NumberOfProcessors >= 8) && (NumberOfProcessor <= 64))}
       VALUE "UtilityValue" {-JobCost}
       VALUE "RunTime" {86*NumberOfProcessor}
     }
  }
}
```

Figure 11.16: Example of a Request Description.

11.4.6 Simulation and Evaluation

We examine the performance of the economic scheduling method by assuming that the overall objective is actually the reduction of the overall response time. This allows the comparison to the conventional scheduling algorithms as presented in Section 11.3. For the economic scheduling model we have to use corresponding utility functions for minimizing the response time. However, we do not know yet which objective functions for machines and users achieve a low average weighted response time. Therefore, several utility functions have been examined in this work as a first starting point.

11.4.6.1 Resource Configurations

We use the same configurations as presented in Section 11.3.1 in Table 11.1. Again, all configurations use a total of 512 resources.
For the simulations, 6 different provider objective functions were used. Here, we briefly describe the first provider **m**achine **f**unction MF_1 whose included terms are also used in the other functions:

$$NumberOfProcessors \cdot RunTime$$

calculates the resource consumption that the job is using within the schedule. The second term calculates the idle processor times before and after the job as well as the parallel idle times for all other resources within the local schedule (see Figure 11.15.):

$$before + after + parallel.$$

The last term of the formula is:

$$1 - parallel\_rel,$$

where *parallel_rel* describes the relation between the idle processor times in parallel to the job within the schedule (parallel) and the processor times actually used by the job. A small value for this factor describes that the idle processor times in parallel are small in comparison to the job resource consumption. This leads to the following objective function MF_1 and its derivations MF_2 to MF_6:

$$\begin{aligned}
MF_1 &= (NumberOfProcessors \cdot RunTime \\
&+ after + before + parallel) \cdot (1 - parallel\_rel) \\
MF_2 &= (NumberOfProcessors \cdot RunTime \\
&+ after + before + parallel), \\
MF_3 &= (NumberOfProcessors \cdot RunTime \\
&+ after + before) \cdot (1 - parallel\_rel), \\
MF_4 &= (NumberOfProcessors \cdot RunTime \\
&+ parallel) \cdot (1 - parallel\_rel), \\
MF_5 &= (NumberOfProcessors \cdot RunTime \\
&+ after + parallel) \cdot (1 - parallel\_rel), \\
MF_6 &= (NumberOfProcessors \cdot RunTime \\
&+ before + parallel) \cdot (1 - parallel\_rel).
\end{aligned}$$

11.4.6.2 Job Configurations

For the evaluation, we use the same workloads as described in Table 11.2 of Section 11.3.2. Additionally, a utility function for each job is now necessary to represent the preferences of the corresponding user. To this end, the following 5 user **u**tility **f**unctions (UF) have been applied in our simulations. During a simulation, we assume that all users have the same utility function. Again, these are example utility functions to get first information on the impact on the scheduling performance. Further work is necessary for optimizing these functions for real applications.
The first user utility function prefers the earliest start time of the job. All processing costs are ignored.

$$UF_1 = (-StartTime).$$

The second user utility function only considers the calculation costs caused by the job.

$$UF_2 = (-JobCost).$$

The last user utility functions are combinations of the first two, but with different weights.

$$\begin{aligned}
UF_3 &= (-(StartTime + JobCost)) \\
UF_4 &= (-(StartTime + 2 \cdot JobCost)) \\
UF_5 &= (-(2 \cdot StartTime + JobCost)).
\end{aligned}$$

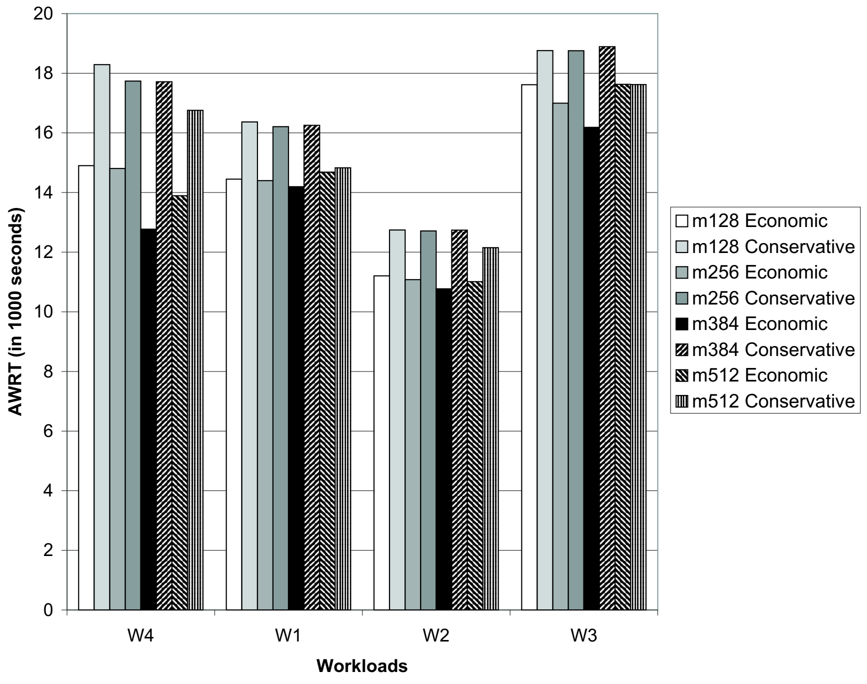


Figure 11.17: Comparison Between Economic and Conventional Scheduling.

11.4.6.3 Results for the Economic Strategies

Figure 11.17 shows a comparison of the average weighted response time for the economic method and for the conventional FCFS/backfilling scheduling system. For both systems the best results achieved have been selected. Note that the used machine and utility functions differ between the various economic simulations. The results show that for all used workloads and all resource configurations, the economically based scheduling system has the capability to outperform the conventional FCFS/backfilling strategy. Note that the economic scheduler is able to outperform backfilling as it is not restricted in the job execution order.

Figure 11.18 gives a comparison between the economic and the conventional scheduling system for only one machine/utility function combination. With the exception of *W3* and *m513*, the used combination of MF_1 and UF_1 outperforms the conventional system for all used workloads and the configurations *m128* and *m512*. Only for the *m512* configuration and the *W3* workload trace, the economic system and the conventional system result in an equal average weighted response time.

Certain combinations of machine and user utility functions can provide good results for different workloads. In the following, we compare different machine/utility functions which are exemplarily shown for the resource configuration *m128*. In Figure 11.19 the average weighted response time is given for all different machine functions in combination with utility function UF_3. The average weighted response time for the machine function MF_2 performs significantly better than all other machine functions. The factor $1 - parallel\_rel$, which is used in all other machine functions, does not work well for this machine configuration. Instead it seems to be beneficial to use absolute values for the processor times, e.g. $(NumberOfProcessors \cdot RunTime + after + before + parallel)$. Unexpectedly, Figure 11.19

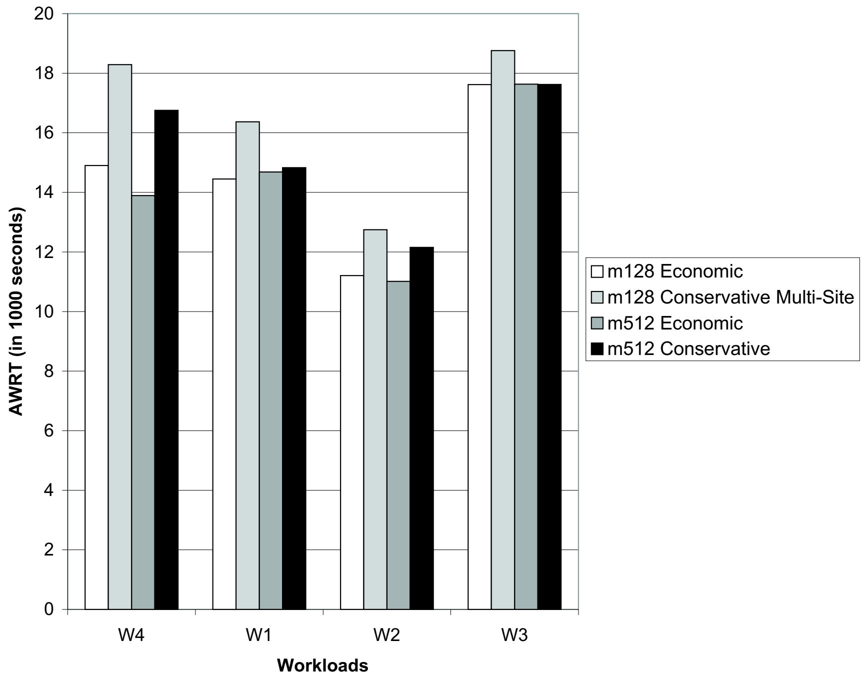


Figure 11.18: Comparison Between Economic and Conventional Scheduling for the Resource Configurations *m128* and *m512* Using *MF1 - UF1*.

also shows that the intention to reduce the free areas within the schedule before a job starts (with attribute $before$) results in very poor average weighted response times (see the results for MF_1, MF_3, MF_6).

Utility function UF_1, which only takes the job start time into account, results in the best average weighted response time. In this case, no attention was paid to the resulting job cost. This means that no minimization of the idle processor times in parallel to the job is regarded. The utility functions which include this job cost yield inferior results in terms of the average weighted response times. The second best result originates from the usage of the utility function UF_3. In opposite to UF_1, the starting time and the job costs are equally weighted. A higher response time was achieved for all other utility combinations, in which either only the job costs (UF_2) or unbalanced weights for the starting time are used.

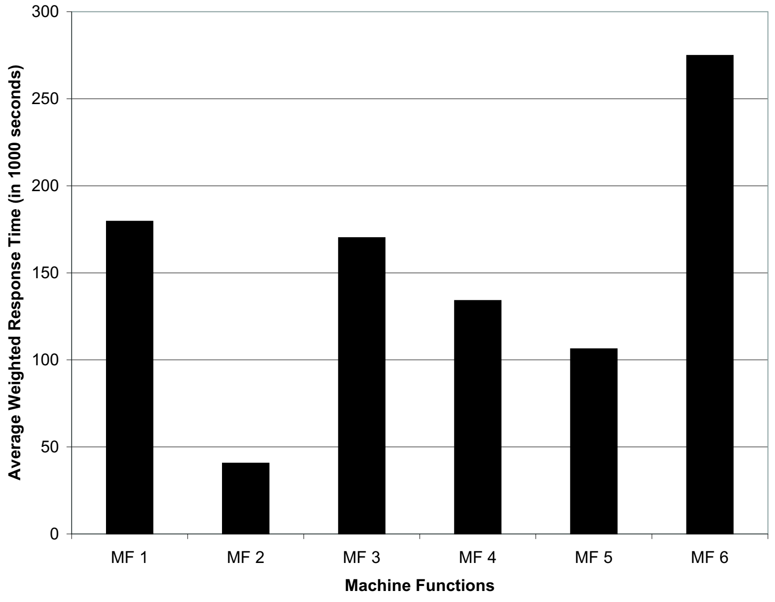


Figure 11.19: The Resulting Average Weighted Response for Resource Configuration *m128*, Utility Function *UF3* and Several Machine Functions for Workload *W4*.

Chapter 12

Conclusion

This work mainly motivates and discusses the concept of multi-objective scheduling systems. Such multi-objective systems can be observed in many real life scenarios. Exemplarily, we introduced the problem of generating flight plans for the allocation of airplanes and airplane crews.

The scheduling of massively parallel processing systems has also been identified as a multi-objective scheduling problem. Here, many different users or user groups want to have access to a limited amount of processing nodes. Furthermore, the machine providers have different commercial relationships with those users. Thus, the machine providers want to process the computational jobs of the different users depending on their assigned preferences. However, existing scheduling strategies are not able to incorporate such preferences in a good way. The usage of quotas or the establishment of partitions normally decreases the utilization of the parallel system. Hence, new multi-objective scheduling strategies must be developed. Several possible concepts to derive such scheduling strategies are presented in this work.

In order to develop such multi-objective strategies, the general scheduling framework has been introduced and extended to several objectives. Our scheduling problems are computationally expensive. This motivates the usage of heuristically driven algorithms. This work provides an overview of all multi-objective scheduling problems with more than 2 objectives that are known to the author. This collection of existing strategies further motivated the development of new strategies that fit to our problem.

Our concept to derive multi-objective scheduling strategies consists of several steps. Within one of those steps we derive the Pareto front of possible scheduling solutions. This is necessary for the machine providers to identify their specific preferences and to derive estimates for possible scheduling improvements compared with the existing standard scheduling strategies. To this end, we used multi-objective Evolutionary Algorithms to generate the Pareto fronts. Thus, this work presented the concepts of such multi-objective Evolutionary Algorithms in more detail.

The state of the art in scheduling massively parallel processing systems has also been presented. The corresponding scheduling strategies are used for comparisons. As motivated in this work, the development of new strategies in this area is based on workload traces from existing parallel machine installations. However, most of those traces originate from different machine environments. Hence, the achieved scheduling results cannot be compared easily. Therefore, this work introduced a method to scale such workload traces to a unique machine size. Those modified workloads build the foundation of the later development of scheduling

strategies.
This development generates the Pareto fronts for all existing scaled workload traces first. To this end, we used two different approaches that use multi-objective Evolutionary Algorithms. The first approach is based on the permutation of job sequences. Here, we found that Evolutionary Algorithms cannot be used as black box optimization tools. Furthermore, problem specific knowledge must be included to generate Pareto fronts with a higher diversity and a better coverage of the possible solution space. The second approach, which is real-value encoded, is relatively simple and leads to a very good coverage of a smaller solution space compared with the first approach.
In order to derive a robust scheduling strategy, our goal was to generate a representative Pareto front by moving and scaling the different generated fronts. However, this approach did not lead to acceptable results. Therefore, we used two other strategies to derive scheduling strategies that can be applied in various environments with a similar performance. The first attempt optimizes a scheduling strategy for a single workload. Then, this strategy is applied to other workload traces and the outcome is analyzed. The second attempt aims to optimize the strategy by using all workload traces in parallel during the development.
For a proof of concept, we defined a scheduling objective that is based on the selection of preferred regions and a linear objective function with a similar strength of the user groups 1 and 2. Note that our methodology is not limited to this scheduling objective.
Based on the defined scheduling objective and the scaled workload traces, we derived six different scheduling strategies. All of these strategies have been analyzed in detail. To this end, we tested whether each of these approaches is able to individually optimize scheduling strategies for a single workload trace. Furthermore, we analyzed how well such a generated strategy can be applied to the other traces without further adaptation. Finally, we used each of the presented six approaches to develop a strategy based on all workloads. The resulting schedules for the individual traces have been discussed in detail.
We found that each of the proposed development strategies is able to improve the scheduling outcome for individual workload traces compared to the standard strategies. Especially, the strategies based on Genetic Fuzzy systems perform very well. The application of scheduling strategies that were optimized for single workload traces to other environments only succeeded with the Michigan and the CCA approach. The scheduling quality for the other approaches at least for one workload trace was much lower relative to the standard scheduling strategies. The development of scheduling strategies based on all workload traces should be done by using the approaches based on the Genetic Fuzzy systems. Only those approaches lead to good results for all workload traces. In general, we recommend the CCA approach as it has a good performance in all of our testing environments. Furthermore, the required number of simulations during the development is relatively small compared with the Michigan approach, which also leads to good scheduling strategies in all of our analyzed scenarios.
As the usage of massively parallel processing systems can still not satisfy the computational demand of the users, Grid Computing extends this concept. Here, several parallel systems of different independent providers are interconnected. This enables the users within a Grid to utilize different heterogeneous resources. However, the complexity of the scheduling strategy further increases in comparison to single parallel installations as different providers have different objectives. Within this work, we analyzed several different scheduling strategies for such Grid scenarios by again using workload traces from real existing machines. The economically driven scheduling approach provides the highest flexibility and adaptability to the Grid scenario. Thus, this approach should be used while building larger Computational

Grids.

Future Work

This thesis uses workload traces from real existing machines during the development of scheduling strategies for parallel computers and for the Grid. As mentioned above, they include all hidden dependencies between jobs and user submissions. However, by using workload traces instead of a workload model the user feedback behavior cannot be included. This is a major disadvantage, as users might react on certain scheduling decisions. Hence, realistic workload models are necessary in order to achieve further improvements of the scheduling strategies. The existence of such models is further necessary in order to extend the concept of user group prioritization within the Grid scenario. Here, users are free to choose the resources of the participating providers. In this context, the modelling of the feedback behavior is even more desired.

As mentioned in Chapter 10, we think that future approaches to develop scheduling strategies should be based on Genetic Fuzzy systems. However, we argue that a larger collection of sorting criteria with a simple FCFS might be better to adapt to the different scheduling scenarios.

The Grid scheduling scenario within this work is concentrated on assigning processor nodes to computational tasks. However, in real Grid environments the usage of the interconnection network as well as the corresponding computer programs and user data must be scheduled. Only a combined scheduling approach that incorporates all those aspects will be able to solve the underlying problem. Again, we motivate the application of the introduced economically driven approach in the Grid scenario. The presented machine and user utility functions for our economic approach were just examples that have proven to increase the scheduling quality. Future research projects could use Evolutionary Algorithms to parameterize more complex utility functions that might further improve the corresponding scheduling outcome.

Bibliography

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. *Local Search in Combinatorial Optimization*, chapter Simulated annealing, pages 91–120. Wiley, 1997.

[2] S. Albers. Online algorithms: A survey. *Mathematical Programming*, 97:3–26, 2003.

[3] P. Alfeld. *Mathematical Methods in Computer Aided Geometric Design*, chapter Scattered data interpolation in three or more variables, pages 1–34. Academic Press Professional, Inc., 1989.

[4] I. Amidor. Scattered data interplolation methods for electronic imaging systems: A survey. *Journal of Electronic Imaging*, 11(2):157–176, April 2002.

[5] E. J. Anderson, C. A. Glass, and C. N. Potts. *Local Search in Combinatorial Optimization*, chapter Machine scheduling, pages 361–414. Wiley, 1997.

[6] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.

[7] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[8] T. Bäck and H.-P. Schwefel. Evolutionary computation: An overview. In T. Fukuda, T. Furuhashi, and D. B. Fogel, editors, *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation (ICEC96), Nagoya*, pages 20–29, Piscataway NJ, 1996. IEEE Press.

[9] T. P. Bagchi. Pareto-optimal solutions for multi-objective production scheduling problems. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO01)*, volume 1993 of *Lecture Notes in Computer Science (LNCS)*, pages 458–471. Springer, 2001.

[10] Z.-Z. Bai and G.-Q. Li. Restrictively preconditioned conjugate gradient methods for systems of linear equations. *Journal of Numerical Analysis*, 23(4):561–580, 2003.

[11] J. Balicki and Z. Kitowski. Multicriteria evolutionary algorithm with tabu search for task assignment. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO01)*, volume 1993 of *Lecture Notes in Computer Science (LNCS)*, pages 373–384. Springer, 2001.

[12] H.-G. Beyer and H.-P. Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[13] J. Blazewicz, W. Cellary, R. Slowinsky, and J. Weglarz. *Scheduling under Resource Constraints: Deterministic models.* Baltzer Science Publishers, 1986.

[14] A. Bonarini. *Fuzzy Modelling: Paradigms and Practice*, chapter Evolutionary Learning of Fuzzy rules: competition and cooperation, pages 265–284. Kluwer Academic Press, 1996.

[15] M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency - Practice and Experience*, 11(15):887–911, 1999.

[16] J. Bruno, E. G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.

[17] R. Buyya, D. Abramson, and J. Giddy. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. In *Proceedings of the 2nd Workshop on Active Middleware Services (AMS00), in conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC00), Pittsburgh.* Kluwer Academic Press, Norwell, August 2000.

[18] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE02)*, 14:1507–1542, November–December 2002.

[19] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. In *Special Issue of the Proceedings of the IEEE on Grid Computing.* IEEE Press, 2005.

[20] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP99)*, volume 1659 of *Lecture Notes in Computer Science (LNCS)*, pages 67–90. Springer, April 1999.

[21] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *Proceedings of the 4th Workshop on Workload Characterization*, pages 140–148. IEEE Press, December 2001.

[22] J. K. Cochran, S.-M. Horng, and J. W. Fowler. A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers and Operations Research*, 30(7):1087–1102, 2003.

[23] C. A. Coello Coello, D. A. Van Veldhuizen, and G. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Publishers, Boston, MA, 2002.

[24] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena. *GENETIC FUZZY SYSTEMS - Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*, volume 19 of *Advances in Fuzzy Systems - Applications and Theory.* World Scientific, Singapore, July 2001.

[25] C. Cotta and J. M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6(1):25–44, 1998.

[26] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP98)*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–68. Springer, 1998.

[27] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons, Chichester, UK, 2001.

[28] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[29] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[30] K. Deb and P. Chakroborty. Time scheduling of transit systems with transfer considerations using genetic algorithms. *Evolutionary Computation*, 6(1):1–24, 1998.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[32] U. Dorndorf, A. Drexl, Y. Nikulin, and E. Pesch. Flight gate scheduling: State-of-the-art and recent developments. Technical Report 584, Institut für Betriebswirtschaftslehre, Kiel, 2004.

[33] A. B. Downey. A parallel workload model and its implications for processor allocation. Technical Report CSD-96-922, University of California at Berkeley, 1996.

[34] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, March 1999.

[35] A. Drexl and Y. Nikulin. Multicriteria airport gate assignment and pareto simulated annealing. Technical Report 586, Institut für Betriebswirtschaftslehre, Kiel, 2005.

[36] M. Drozdowski. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research*, 94:215–230, 1996.

[37] J. Du and J. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, November 1989.

[38] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000.

[39] M. Emmerich, N. Beume, and B. Naujoks. An emo algorithm using the hypervolume measure as selection criterion. In C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, editors, *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO05)*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, pages 62–76. Springer, 2005.

[40] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Enhanced algorithms for multi-site scheduling. In M. Parashar, editor, *Proceedings of the 3rd International Workshop on Grid Computing (GRID02)*, volume 2536 of *Lecture Notes in Computer Science (LNCS)*, pages 219–231. Springer, 2002.

[41] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On advantages of grid computing for parallel job scheduling. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CC-GRID02)*, pages 39–46. IEEE Press, May 2002.

[42] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. On effects of machine configurations on parallel job scheduling in computational grids. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS02)*, pages 169–179. VDE, April 2002.

[43] C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in grid computing. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP02)*, volume 2537 of *Lecture Notes in Computer Science (LNCS)*, pages 128–152. Springer, 2002.

[44] C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global grid computing for job scheduling. In R. Buyya, editor, *Proceedings of 5th IEEE/ACM International Workshop on Grid Computing (GRID04), in Conjunction with (SuperComputing04)*, pages 374–379. IEEE Computer Society, November 2004.

[45] C. Ernemann, M. Krogmann, J. Lepping, and R. Yahyapour. Scheduling on the top 50 machines. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP04)*, volume 3277 of *Lecture Notes in Computer Science (LNCS)*, pages 17–46. Springer, October 2004.

[46] C. Ernemann, U. Schwiegelshohn, M. Emmerich, L. Schönemann, and N. Beume. Scheduling algorithm development based on complex owner defined objectives. Technical Report 191/05, SFB 531, CI-Report, University of Dortmund, January 2005.

[47] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP03)*, volume 2862 of *Lecture Notes in Computer Science (LNCS)*, pages 166–183. Springer, October 2003.

[48] C. Ernemann and R. Yahyapour. *Grid Resource Management - State of the Art and Future Trends*, chapter Applying Economic Scheduling Methods to Grid Environments, pages 491–506. Kluwer Academic Publishers, 2003.

[49] R. Farwig. Rate of convergence of shepard's global interpolation formula. *Mathematics of Computation*, 46(174):577–590, 1986.

[50] D. G. Feitelson. Packing schemes for gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP96)*, volume 1162 of *Lecture Notes in Computer Science (LNCS)*, pages 89–110. Springer, 1996.

[51] D. G. Feitelson. Memory usage in the LANL CM-5 workload. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 78–94. Springer, 1997.

[52] D. G. Feitelson. Workload modeling for performance evaluation. In M. C. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science (LNCS)*, pages 114–141. Springer, 2002.

[53] D. G. Feitelson. Metric and Workload Effects on Computer Systems Evaluation. *Computer*, 36(9):18–25, September 2003.

[54] D. G. Feitelson. Parallel workload archive. http://www.cs.huji.ac.il/labs/parallel/workload/, January 2005.

[55] D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 238–261. Springer, 1997.

[56] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP95)*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 337–360. Springer, 1995.

[57] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP98)*, volume 1459 of *Lecture Notes in Computer Science (LNCS)*, pages 1–24. Springer, 1998.

[58] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 1–34. Springer, 1997.

[59] D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *Proceedings of the 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, pages 542–547. IEEE Computer Society Press, 1998.

[60] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[61] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann, 1993.

[62] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.

[63] C. Franke, J. Lepping, F. Hoffmann, and U. Schwiegelshohn. Development of scheduling strategies with genetic fuzzy systems. Forschungsbericht 0106, Fakultät für Elektrotechnik und Informationstechnik, Universität Dortmund, May 29 2006. ISSN 0941-4169.

[64] C. Franke, J. Lepping, and U. Schwiegelshohn. On advantages of scheduling using genetic fuzzy systems. In E. Frachtenberg and U. Schwiegelshohn, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP06)*, Lecture Notes in Computer Science (LNCS). Springer, 2006. to appear.

[65] C. Franke, U. Schwiegelshohn, and R. Yahyapour. Job scheduling for computational grids. Forschungsbericht 0206, Fakultät für Elektrotechnik und Informationstechnik, Universität Dortmund, May 29 2006. ISSN 0941-4169.

[66] M. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, June 1975.

[67] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[68] R. Gibbons. A historical application profiler for use by parallel schedulers. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 58–77. Springer, 1997.

[69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[70] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 15:287–326, 1979.

[71] M. Guntsch and M. Middendorf. Solving multi-criteria optimization problems with population-based aco. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO03)*, volume 2632 of *Lecture Notes in Computer Science (LNCS)*, pages 464–478. Springer, 2003.

[72] P. Hajela and C.-Y. Lin. Genetic search strategies in multi-criterion optimal design. *Structural Optimization*, 4(2):99–107, 1992.

[73] L. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th SIAM Symposium on Discrete Algorithms (SODA96)*, pages 142–151. SIAM, Philadelphia, January 1996.

[74] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In R. Buyya and M. Baker, editors, *Proceedings of the 7th International Conference on High Performance Computing (HiPC00)*, volume 1971 of *Lecture Notes in Computer Science (LNCS)*, pages 191–202, Bangalore, Indien, 2000. Springer.

[75] R. L. Hardy. Multiquadrics equations of topography and other irregular surfaces. *Journal of Geophysical Research*, 76:1905–1915, 1971.

[76] E. Hart, P. Roos, and J. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, 6(1):61–80, 1998.

[77] E. Hart, P. Ross, and D. Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.

[78] A. Hertz, E. Taillard, and D. de Werra. *Local Search in Combinatorial Optimization*, chapter Tabu search, pages 121–136. Wiley, 1997.

[79] F. Hoffmann. Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE*, 89(9):1318–1333, September 2001.

[80] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9:297–314, 1962.

[81] J. Horn, N. Nafpliotis, and D. E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation (ICEC95), in conjunction with the IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, 1994. IEEE Service Center.

[82] S. Hotovy. Workload evolution on the cornell theory center IBM SP2. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP96)*, volume 1162 of *Lecture Notes in Computer Science (LNCS)*, pages 27–40. Springer, 1996.

[83] A. Iosup, D. H. J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour. On modeling synthetic workloads for grid performance evaluation. In E. Frachtenberg and U. Schwiegelshohn, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP06)*, Lecture Notes in Computer Science (LNCS). Springer, 2006. to appear.

[84] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan. Modeling of workload in MPPs. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 95–116. Springer, 1997.

[85] T. Jansen and R. P. Wiegand. Exploring the explorative advantage of the cooperative coevolutionary (1+1) ea. In E. Cantú-Paz et al., editor, *Genetic and Evolutionary Computation Conference (GECCO03)*, volume 2723 of *Lecture Notes in Computer Science (LNCS)*, pages 310–321. Springer, 2003.

[86] Y. Jin, W. von Seelen, and B. Sendhoff. On generating fc^3 fuzzy rule systems from data using evolution strategies. *IEEE Transactions on System, Man and Cybernetics*, 29(6):829–845, December 1999.

[87] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer System Science*, 9:256–278, 1974.

[88] C.-F. Juang, J.-Y. Lin, and C.-T. Lin. Genetic Reinforcement Learning through Symbiotic Evolution for Fuzzy Controller Design. *IEEE Transactions on System, Man and Cybernetics*, 30(2):290–302, April 2000.

[89] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, November 1986.

[90] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[91] C. Kenyon and G. Cheliotis. *Grid Resource Management - State of the Art and Future Trends*, chapter Grid Resource Commercialization - Economic Engineering and Delivery Scenarios, pages 465–478. Kluwer Academic Publishers, 2003.

[92] H. Kita, Y. Yabumoto, N. Mori, and Y. Nishikawa. Multi-objective optimization by means of the thermodynamical genetic algorithm. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN96), Berlin*, volume 1141 of *Lecture Notes in Computer Science (LNCS)*, pages 504–512. Springer, 1996.

[93] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[94] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the design and evaluation of job scheduling systems. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP99)*, volume 1659 of *Lecture Notes in Computer Science (LNCS)*. Springer, 1999.

[95] F. Kursawe. A variant of evolution strategies for vector optimization. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN90), Dortmund*, pages 193–197. Springer, 1991.

[96] M. Laumanns. *Analysis and Application of Evolutionary Multiobjective Optimization Algorithms.* PhD thesis, Swiss Federal Institute of Technology Zürich, 2003.

[97] M. Laumanns, G. Rudolph, and H.-P. Schwefel. A spatial predator-prey approach to multi-objective optimization. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN98), Amsterdam*, pages 241–249. Springer, 1998.

[98] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP05)*, volume 3277 of *Lecture Notes in Computer Science (LNCS)*, pages 253–263. Springer, April 2005.

[99] D. A. Lifka. The ANL/IBM SP scheduling system. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP95)*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 295–303. Springer, 1995.

[100] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS88)*, pages 104–111, 1988.

[101] M. Livny and R. Raman. High-throughput resource management. In I. Foster and C. Kesselman, editors, *The Grid - Blueprint for a New Computing Infrastructure*, pages 311–337. Morgan Kaufmann, 1999.

[102] V. Lo, J. Mache, and K. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP98)*, volume 1459 of *Lecture Notes in Computer Science (LNCS)*, pages 25–46. Springer, 1998.

[103] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, November 2003.

[104] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In *The Ecology of Computation*, volume 2 of *Studies in Computer Science and Artifical Intelligence*, pages 177–255. North Holland, 1988.

[105] D. McLaughlin, S. Sardesai, and P. Dasgupta. Preemptive scheduling for distributed systems. In *Proceedings of the 11th International Conference on Parallel and Distributed Computing Systems*, September 1998.

[106] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, October 1959.

[107] C. A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[108] K. M. Miettinen. *Nonlinear Multiobjective Optimization*. International Series in Operations Research & Management Science. Kluwer, 1999.

[109] D. Montana. Introduction to the special issue: Evolutionary algorithms for scheduling. *Evolutionary Computation*, 6(1):5–9, 1998.

[110] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical of Computer Science*, 130(1):17–47, 1994.

[111] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transaction on Parallel & Distributed Systems*, 12(6):529–543, June 2001.

[112] A. W. Mu'alem and D. G. Feitelson. Bicriteria scheduling for parallel jobs. In *Multidisciplinary International Conference on Scheduling: Theory & Applications (MISTA03)*, volume 2, pages 606–619, Nottingham, August 2003. Springer.

[113] T. Murata and H. Ishibuchi. Multi-objective genetic algorithms (moga95). In *Proceedings of the 2th IEEE Conference on Evolutionary Computation (ICEC95)*, pages 289–294. IEEE Service Center, November 1995.

[114] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[115] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[116] W. P. M. Nuijten and E. H. L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[117] A. Osyczka. *Evolutionary Algorithms for Single and Multicriteria Design Optimization*, volume 79 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 2002.

[118] A. Osyczka and S. Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural Optimization*, 10(2):94–99, 1999.

[119] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.

[120] L. Panait, R.-P. Wiegand, and S. Luke. A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In K. Deb and R. Poli et al., editors, *Genetic and Evolutionary Computation Conference (GECCO04)*, volume 3102 of *Lecture Notes in Computer Science (LNCS)*, pages 573–584. Springer, 2004.

[121] J. Paredis. Coevolutionary computation. *Artificial Life*, 2(4):355–375, 1995.

[122] C. Peterson and B. Söderberg. *Local Search in Combinatorial Optimization*, chapter Artifical neural Networks, pages 173–213. Wiley, 1997.

[123] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice-Hall, 2nd edition, 2002.

[124] M. A. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature (PPSN94)*, volume 866 of *Lecture Notes in Computer Science (LNCS)*, pages 249–257. Springer, 1994.

[125] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, TU Berlin, 1971.

[126] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzbook, Stuttgart, 1973.

[127] I. Rechenberg. *Evolutionsstrategie '94.* Frommann-Holzboog, Stuttgart, 1994.

[128] C. R. Reeves and T. Yamada. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, 6(1):45–60, 1998.

[129] G. Rudolph. An evolutionary algorithm for integer programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature (PPSN00)*, pages 139–148, Berlin, 1994. Springer.

[130] G. Rudolph. Evolutionary search under partially ordered fitness sets. In M. F. Sebaaly, editor, *Proceedings of the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI01)*, pages 818–822, Millet, Kanada, 2001. ICSC Academic Press.

[131] R. Schaback. Improved error bounds for scattered data interpolation by radial basis functions. *Mathematics of Computation*, 68(225):201–216, January 1999.

[132] J. D. Schaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, TN, 1984.

[133] J. M. Schopf. *Grid Resource Management - State of the Art and Future Trends*, chapter Ten Actions When Grid Scheduling - The User as a Grid Scheduler, pages 15–23. Kluwer Academic Publishers, 2003.

[134] H.-P. Schwefel. *Evolutionsstrategie und Numerische Optimierung*. PhD thesis, TU Berlin, 1975.

[135] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, 1981.

[136] H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[137] H.-P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Advances in Artificial Life - Proceedings of the 3rd European Conference on Artificial Life (ECAL95)*, pages 893–907, Berlin, 1995. Springer.

[138] U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP98)*, volume 1459 of *Lecture Notes in Computer Science (LNCS)*, pages 180–198. Springer, March 1998.

[139] U. Schwiegelshohn and R. Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.

[140] U. Schwiegelshohn and R. Yahyapour. *Grid Resource Management - State of the Art and Future Trends*, chapter Attributes for Communication Between Grid Scheduling Instances, pages 41–52. Kluwer Academic Publishers, 2003.

[141] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 23rd ACM National Conference*, pages 517–524, New York, USA, 1968. ACM Press.

[142] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, June 1994.

[143] D. B. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, December 1995.

[144] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY – LoadLeveler API project. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP96)*, volume 1162 of *Lecture Notes in Computer Science (LNCS)*, pages 41–47. Springer, 1996.

[145] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, Department of Computer Science, University of Pittsburgh, 1980.

[146] B. Song, C. Ernemann, and R. Yahyapour. Modeling of parameters in supercomputer workloads. In *Proceedings of the Workshop on Parallel Systems and Algorithms (PASA04) in conjunction with (ARCS04): Organic and Pervasive Computing*, volume P-41 of *Lecture Notes in Informatics*, pages 400–409. Gesellschaft für Informatik, March 2004.

[147] B. Song, C. Ernemann, and R. Yahyapour. Parallel computer workload modeling with Markov chains. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP04)*, Lecture Notes in Computer Science (LNCS). Springer, 2005.

[148] B. Song, C. Ernemann, and R. Yahyapour. User group-based workload analysis and modeling. In *Proceedings of Cluster Computing and Grid (CCGRID05)*. IEEE Press, 2005. CD-ROM.

[149] H. Späth. *The Cluster Dissection and Analysis: Theory,FORTRAN Programs, Examples*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[150] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[151] D. Srinivasan and A. Tettamanzi. A heuristics-guided evolutionary approach to multiobjective generation scheduling. *IEEE Proceedings on Generation, Transmission and Distribution*, 143(6):553–559, 1996.

[152] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21:115–122, 1997.

[153] T. Takagi and M. Sugeno. Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, 1985.

[154] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP Scheduler Using Slack-Based Backfilling. In *Proceedings of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, pages 513–517. IEEE Computer Society, 1999.

[155] H. Tamaki, E. Nishino, and S. Abe. A genetic algorithm approach to multi-objective scheduling problems with earliness and tardiness penalties. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 46–52. IEEE, 1999.

[156] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, 2002.

[157] J. J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response times. In *Proceedings of the 6th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA94)*, pages 200–209. ACM Press, New York, USA, June 1994.

[158] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1997.

[159] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. *Handbook of Transportation Science*, volume 23 of *International Series in Operations Research and Management Science*, chapter Airline Crew Scheduling, pages 493–521. Kluwer Academic Publishers, Norwell, MA, 1999.

[160] D. V. Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–148, 2000.

[161] C. A. Waldspurger, T. Hogg, B. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[162] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS98)*, pages 612–621. IEEE Computer Society, 1998.

[163] S. Webster. A priority rule for minimizing weighted flow time in a class of parallel machine scheduling problems. *European Journal of Operational Research*, 70:327–334, 1993.

[164] K. Windisch, V. Lo, R. Moore, D. G. Feitelson, and B. Nitzberg. A comparison of workload traces from two production parallel machines. In *6th Symposium on Frontiers Massively Parallel Computers*, pages 319–326. IEEE Computer Society, October 1996.

[165] P. C. Wong and R. D. Bergeron. 30 years of multidimensional multivariate visualization. In G. M. Nielson, H. Hagan, and H. Muller, editors, *Scientific Visualization, Overviews, Methodologies, and Techniques*, pages 3–33. IEEE Computer Society, 1997.

[166] F. Ygge. *Market-Oriented Programming and its Application to Power Load Management.* PhD thesis, Department of Computer Science, Lund University, 1998.

[167] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[168] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tino, A. Kabán, and H.-P. Schwefel, editors, *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN04)*, volume 3242 of *Lecture Notes in Computer Science (LNCS)*, pages 832–842. Springer, 2004.

[169] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Proceedings of the EUROGEN 200: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001. International Center for Numerical Methods in Engineering (Cmine).

Appendix A

Scaling of Workload Traces - Detailed Results

workload	m	f	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
CTC	430	ref	EASY	79285	29306750	66	13905	53442	8335013015
	1024	2.41		135157	29306750	67	13242	52897	19890060461
		2.42		135358	29306750	67	13475	52979	20013239358
		2.43		135754	29306750	67	14267	53771	20130844161
		2.45		136922	29306750	68	14480	54036	20322861231
		2.46		136825	29306750	68	13751	53267	20455740107
		2.47		137664	29306750	69	15058	54540	20563974522
		2.48		137904	29306750	69	15071	54611	20486963613
	430	ref	FCFS	79285	29306750	66	19460	58996	8335013015
	1024	2.43		135754	29306750	67	17768	57272	20130844161
		2.44		136524	29306750	67	18818	58326	20291066216
		2.45		136922	29306750	68	19503	59058	20322861231
		2.46		136825	29306750	68	18233	57749	20455740107
		2.47		137664	29306750	69	19333	58815	20563974522
		2.48		137904	29306750	69	19058	58598	20486963613
		2.49		138547	29306750	69	19774	59291	20675400432
KTH	100	ref	EASY	28482	29363625	69	24677	75805	2024854282
	1024	10.68		166184	29363625	72	24756	75880	21649282727
		10.69		165766	29363625	72	24274	75233	21668432748
		10.70		166323	29363625	72	24344	75549	21665961992
		10.71		165396	29363625	72	24672	75826	21708443586
		10.72		166443	29363625	72	24648	75775	21663836681
		10.75		167581	29363625	72	24190	75273	21763427500
		10.78		170046	29363625	73	24417	75546	21829946042
		10.80		168153	29363625	73	25217	76284	21871159818
		10.83		168770	29363625	73	25510	76587	21904565195
	100	ref	FCFS	28482	29381343	69	400649	451777	2024854282
	1024	10.71		165396	29379434	72	167185	218339	21708443586
		10.72		166443	29380430	72	104541	155669	21663836681

Continued on next page

workload	m	f	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
KTH	1024	10.80	FCFS	168153	29374047	73	291278	342345	21871159818
		10.85		167431	29366917	73	295568	346661	21968343948
		10.88		167681	29381624	73	404008	455149	22016195800
		10.89		167991	29366517	73	424255	475405	22051851208
		10.90		169405	29378230	73	281495	332646	22080508136
		10.92		168894	29371367	74	415358	466515	22127579593
		10.96		169370	29381584	74	539856	590999	22204787743
		10.99		170417	29380278	74	491738	542886	22263296356
NASA	128	ref	EASY	42049	7945421	47	6	9482	474928903
	1024	8.00		188706	7945421	47	2	9478	3799431224
		8.01		188659	7945421	47	436	9910	3805309069
		8.04		189104	7945421	47	370	9850	3813901379
		8.05		190463	7945421	47	466	9952	3815152286
		8.06		190221	7945421	47	527	10001	3825085688
		8.07		190897	7945421	47	380	9847	3829707646
		8.08		191454	7945421	47	483	9967	3829000061
		8.09		190514	7945507	47	736	10220	3838797287
		8.10		190580	7945421	47	243	9730	3835645184
	128	ref	FCFS	42049	7945421	47	6	9482	474928903
	1024	8.00		188258	7945421	47	4	9480	3799431224
		8.01		188659	7945421	47	562	10036	3805309069
		8.02		189563	7945421	47	629	10126	3806198375
		8.03		189864	7945421	47	427	9901	3810853391
		8.04		189104	7945421	47	534	10013	3813901379
		8.05		190463	7945421	47	562	10048	3815152286
		8.06		190221	7945421	47	721	10194	3825085688
		8.07		190897	7945421	47	531	9998	3829707646
		8.08		191454	7945421	47	587	10070	3829000061
		8.09		190514	7945507	47	605	10088	3838797287
SDSC00	128	ref	EASY	67655	63192267	83	76059	116516	6749918264
		8.12		308872	63209190	85	70622	111043	54813430352
		8.14		309778	63189633	85	71757	112264	54908840905
		8.15		310917	63195547	85	78663	119080	55003341172
		8.16		310209	63189633	85	76235	116714	55030054463
		8.18		310513	63189633	85	74827	115312	55206637895
		8.19		310286	63247375	85	77472	118119	55258239565
		8.20		311976	63194139	86	78585	119254	55368328613
		8.21		312219	63204664	86	75787	116408	55369411171
	1024	8.22		313024	63200276	86	75811	116267	55499902234
	128	ref	FCFS	67655	68623991	77	2182091	2222548	6749918264
	1024	8.55		321966	68877042	82	2133228	2173666	57703096198
		8.56		323298	69093787	82	2154991	2195442	57785593002
		8.58		323903	69074629	82	2180614	2221139	58020939264
		8.59		323908	69499787	82	2346320	2386846	57999342465
		8.60		325858	69428033	82	2338591	2379182	58011833809
		8.61		325467	69146937	82	2248848	2289373	58074546998
		8.63		325458	69258234	82	2219200	2259628	58211844138

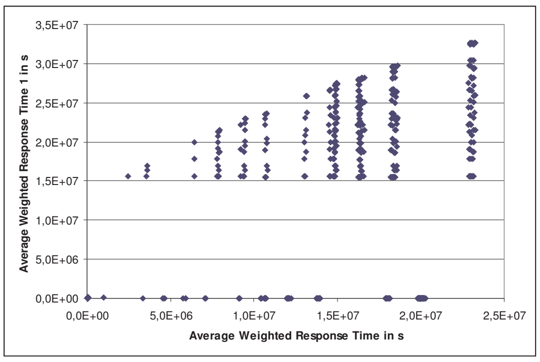
Continued on next page

workload	m	f	Policy	n	C_{max} in seconds	UTIL in %	AWWT in seconds	AWRT in seconds	RC
SDSC95	416	ref	EASY	75730	31662080	63	13723	46907	8284847126
	1024	2.46		130380	31662080	63	13287	46492	20351822499
		2.47		131399	31662080	63	13144	46288	20464087105
		2.48		131884	31662080	63	13840	46985	20534988559
		2.49		131730	31662080	64	13957	47245	20722722130
		2.50		132536	31662080	64	14409	47682	20734539617
		2.52		133289	31662080	64	14432	47628	20794582470
	416	ref	FCFS	75730	31662080	63	17474	50658	8284847126
		2.48		131884	31662080	63	17327	50472	20534988559
		2.49		131730	31662080	64	17053	50341	20722722130
		2.50		132536	31662080	64	17624	50896	20734539617
		2.52		133289	31662080	64	17676	50872	20794582470
		2.53		133924	31662080	65	17639	50820	20955732920
SDSC96	416	ref	EASY	37910	31842431	62	9134	48732	8163457982
	1024	2.46		65498	31842431	62	9055	48736	20026074751
		2.48		66007	31842431	62	8799	48357	20184805564
		2.50		66457	31842431	63	9386	49134	20353508244
		2.51		66497	31842431	63	9874	49315	20502723327
		2.52		66653	31842431	63	9419	48715	20629070916
	416	ref	FCFS	37910	31842431	62	10594	50192	8163457982
	1024	2.47		65842	31842431	62	9674	49361	20120648801
		2.48		66007	31842431	62	10008	49566	20184805564
		2.49		66274	31842431	63	11312	51211	20374472890
		2.50		66457	31842431	63	11321	51069	20353508244
		2.52		66653	31842431	63	11089	50386	20629070916

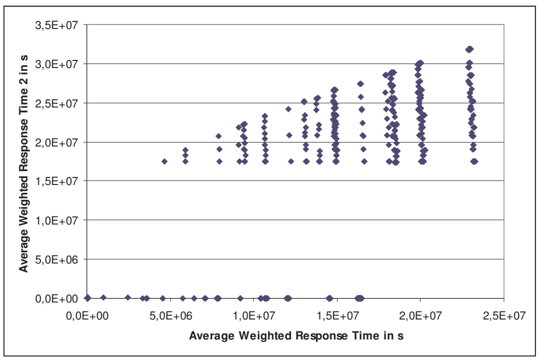
Table A.1: All Results for Increased Scaling Factors with $d = 50$.

Appendix B

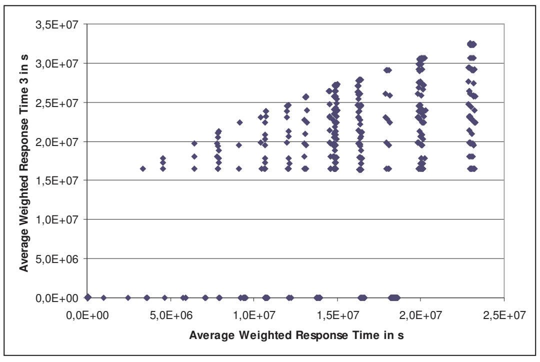
Complete CTC Pareto Front



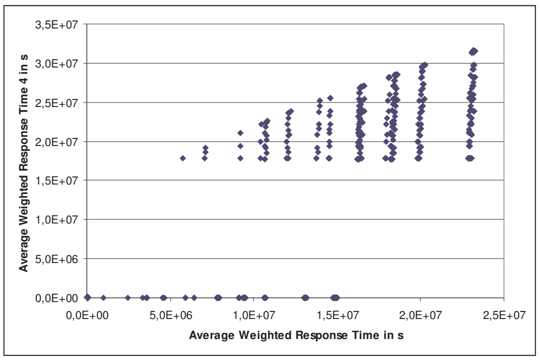
(a) AWRT vs. AWRT1.



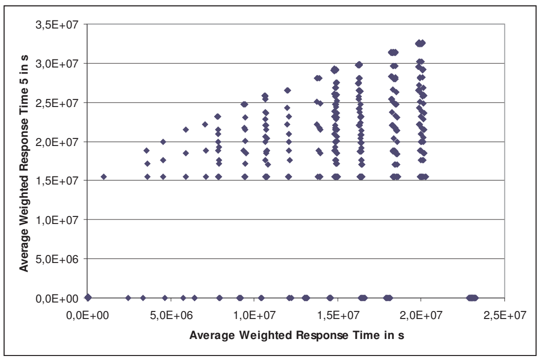
(b) AWRT vs. AWRT2.



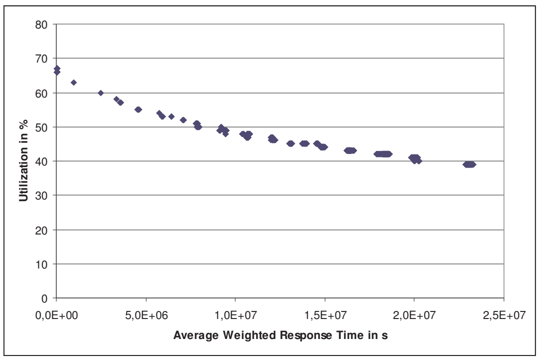
(c) AWRT vs. AWRT3.



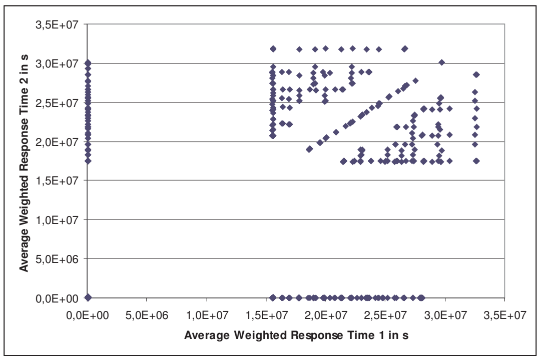
(d) AWRT vs. AWRT4.



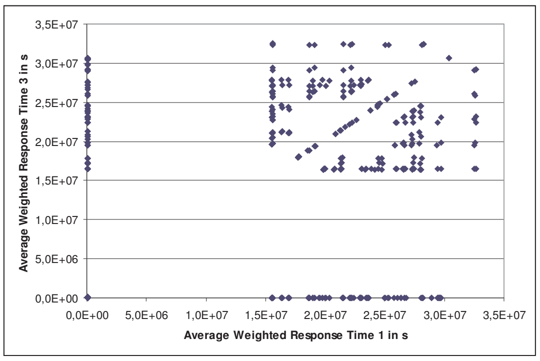
(e) AWRT vs. AWRT5.



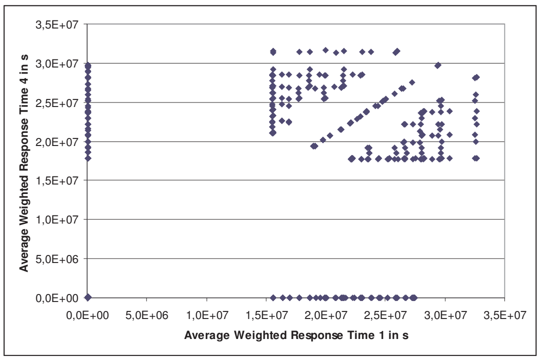
(f) AWRT vs. UTIL.



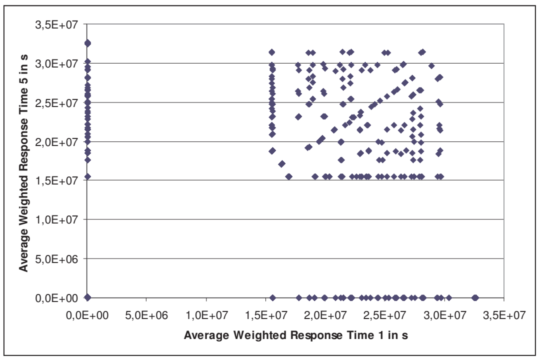
(g) AWRT1 vs. AWRT2.



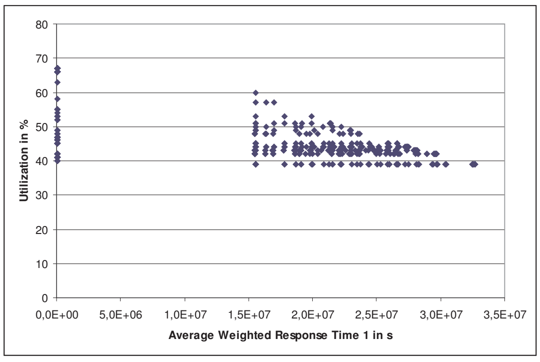
(h) AWRT1 vs. AWRT3.



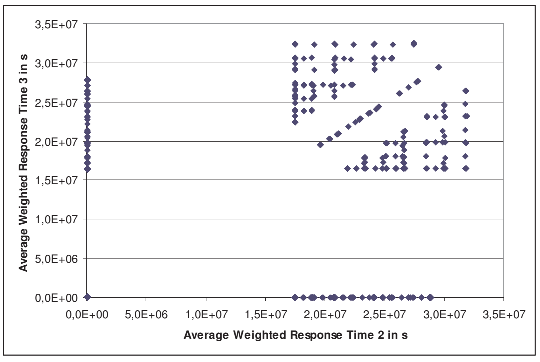
(i) AWRT1 vs. AWRT4.



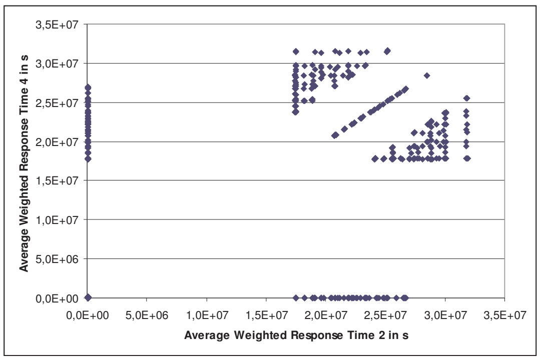
(j) AWRT1 vs. AWRT5.



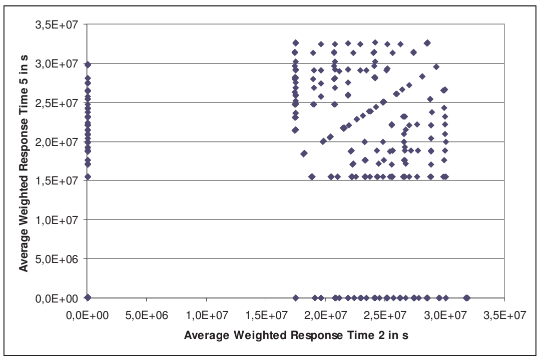
(k) AWRT1 vs. UTIL.



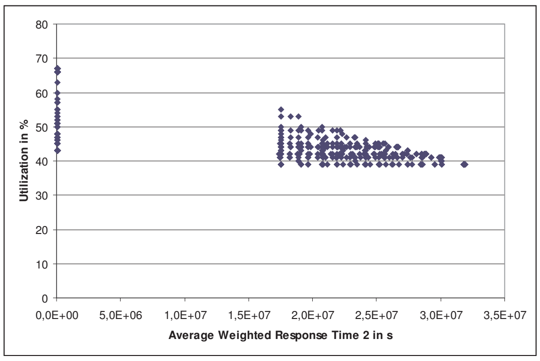
(l) AWRT2 vs. AWRT3.



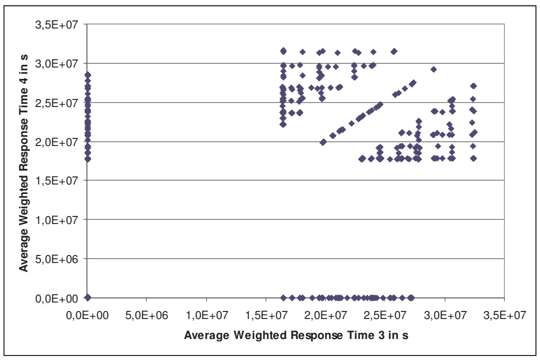
(m) AWRT2 vs. AWRT4.



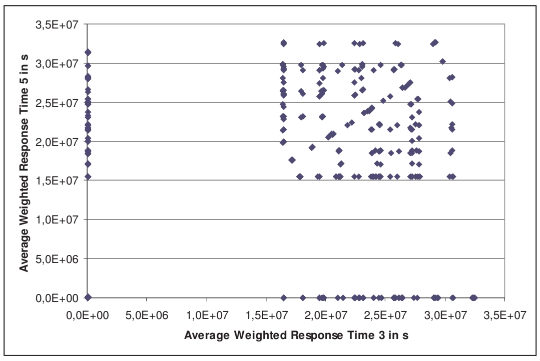
(n) AWRT2 vs. AWRT5.



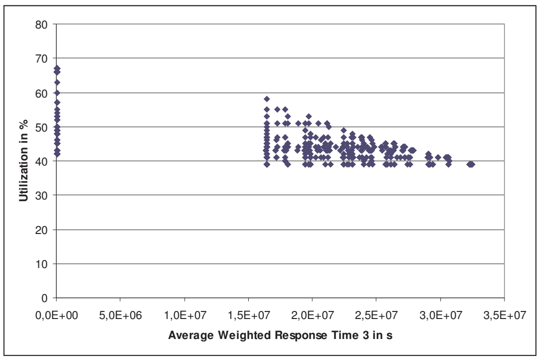
(o) AWRT2 vs. UTIL.



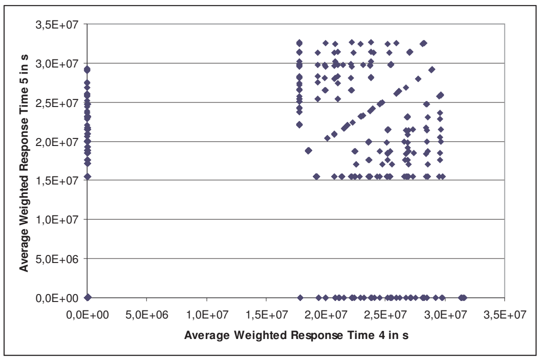
(p) AWRT3 vs. AWRT4.



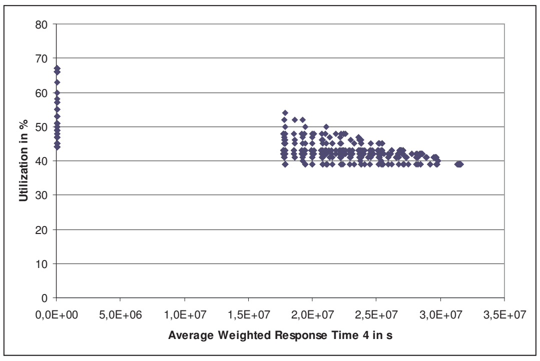
(q) AWRT3 vs. AWRT5.



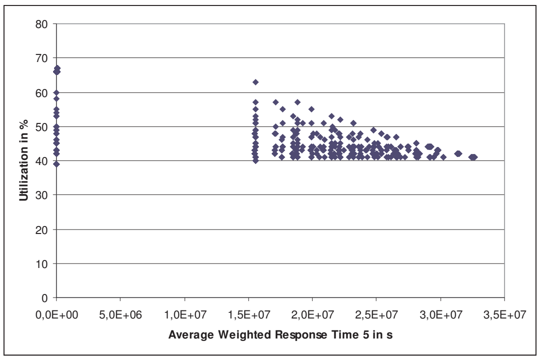
(r) AWRT3 vs. UTIL.



(s) AWRT4 vs. AWRT5.



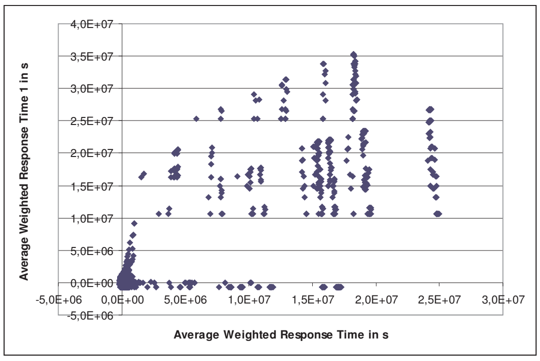
(t) AWRT4 vs. UTIL.



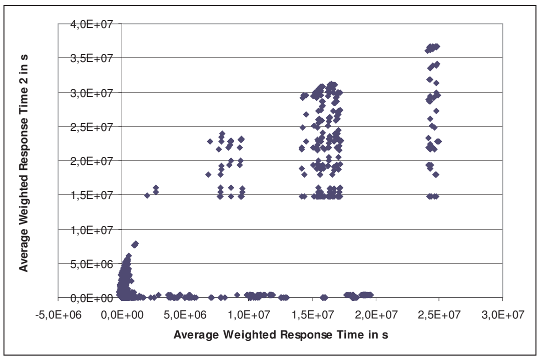
(u) AWRT5 vs. UTIL.

Appendix C

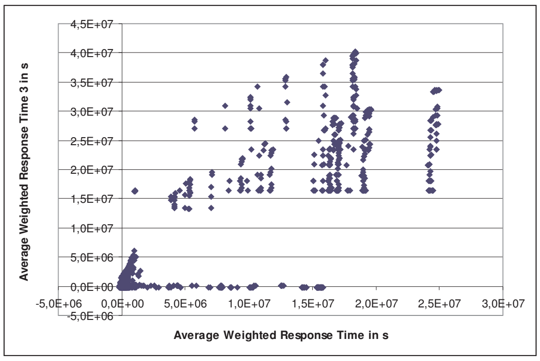
Complete Pareto Front After Transformation using all Workloads



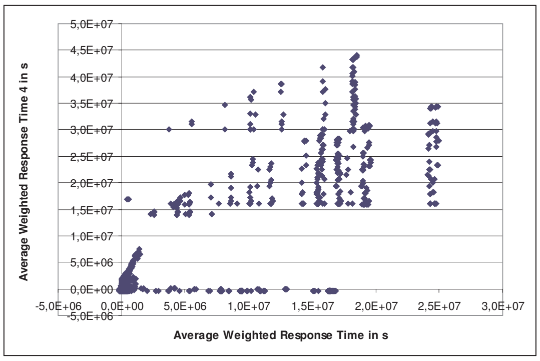
(a) AWRT vs. AWRT1.



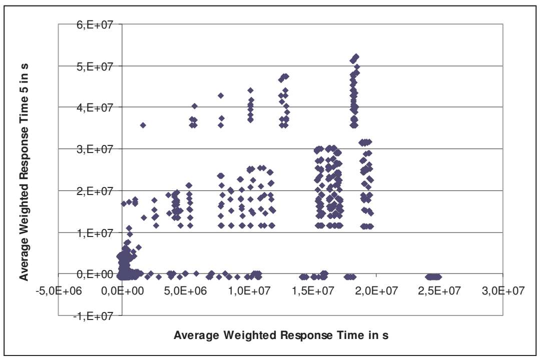
(b) AWRT vs. AWRT2.



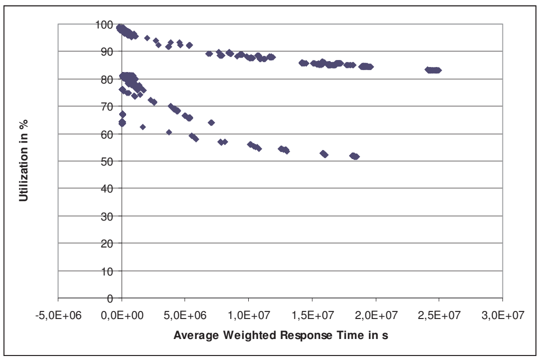
(c) AWRT vs. AWRT3.



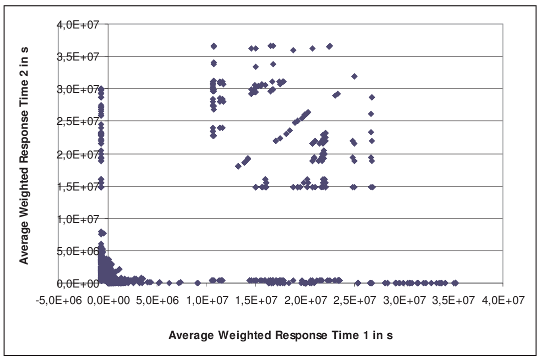
(d) AWRT vs. AWRT4.



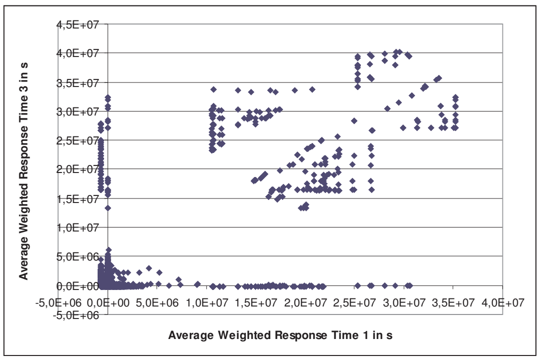
(e) AWRT vs. AWRT5.



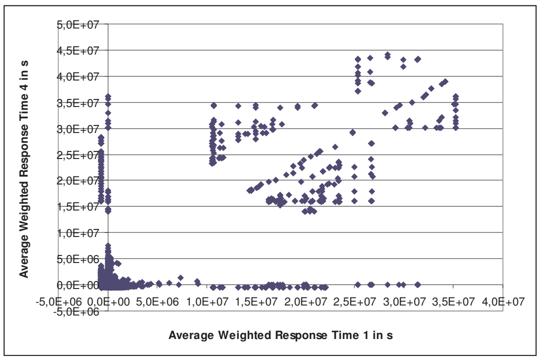
(f) AWRT vs. UTIL.



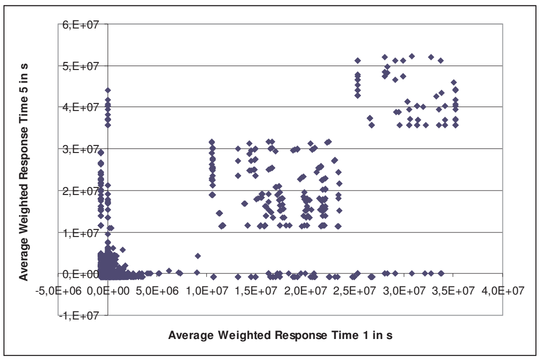
(g) AWRT1 vs. AWRT2.



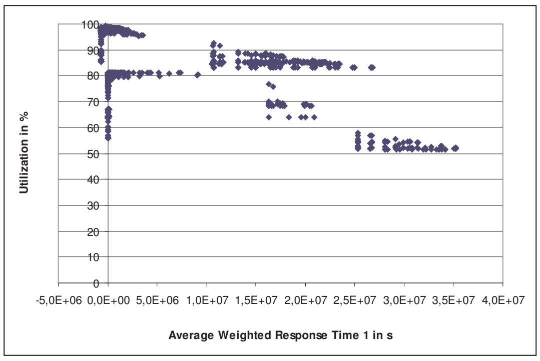
(h) AWRT1 vs. AWRT3.



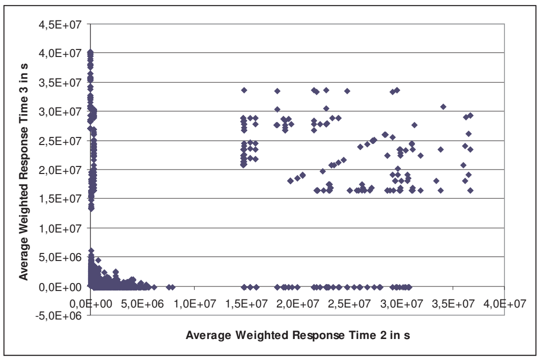
(i) AWRT1 vs. AWRT4.



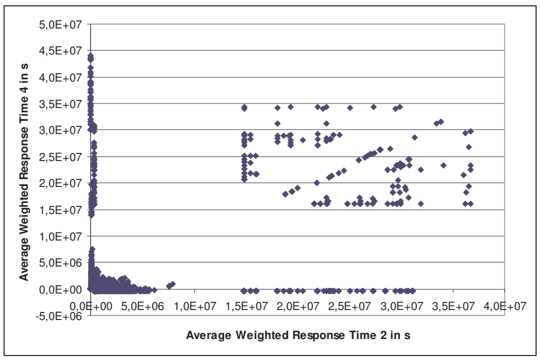
(j) AWRT1 vs. AWRT5.



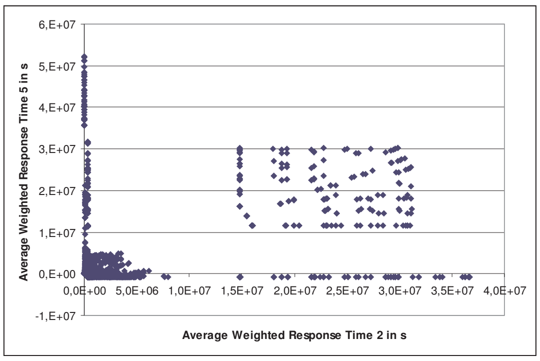
(k) AWRT1 vs. UTIL.



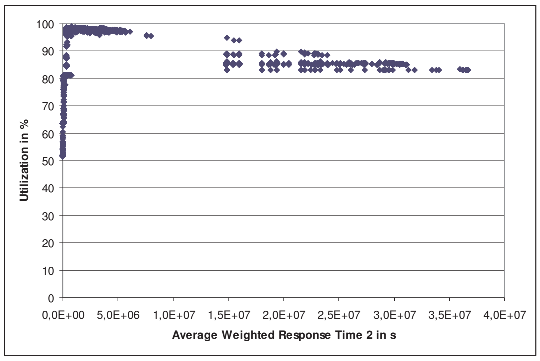
(l) AWRT2 vs. AWRT3.



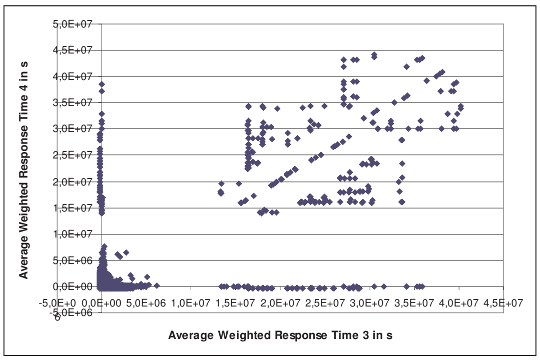
(m) AWRT2 vs. AWRT4.



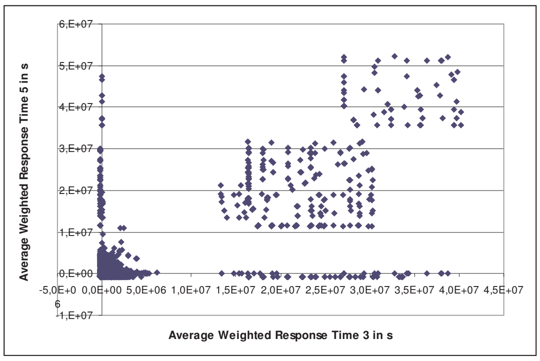
(n) AWRT2 vs. AWRT5.



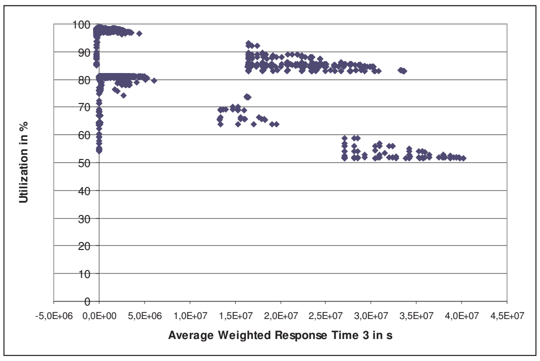
(o) AWRT2 vs. UTIL.



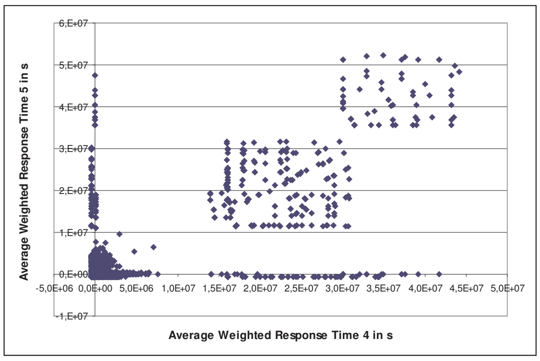
(p) AWRT3 vs. AWRT4.



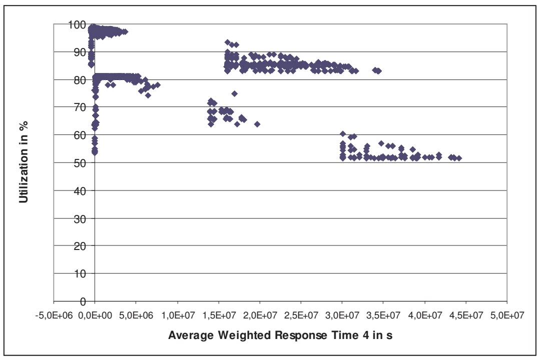
(q) AWRT3 vs. AWRT5.



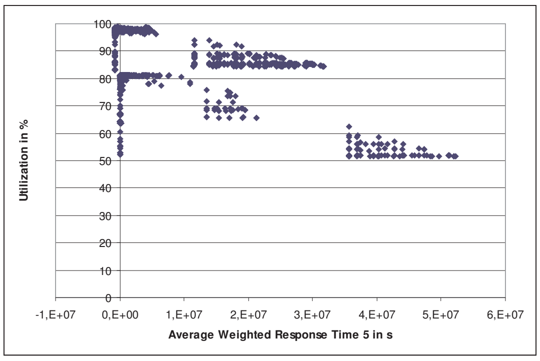
(r) AWRT3 vs. UTIL.



(s) AWRT4 vs. AWRT5.



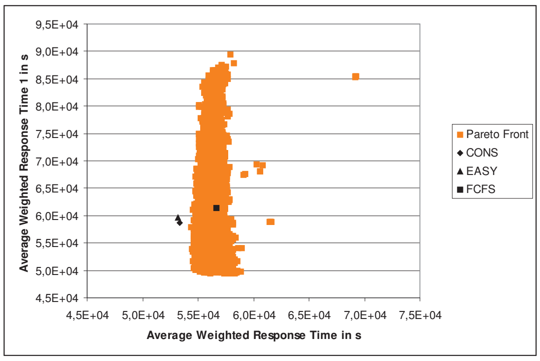
(t) AWRT4 vs. UTIL.



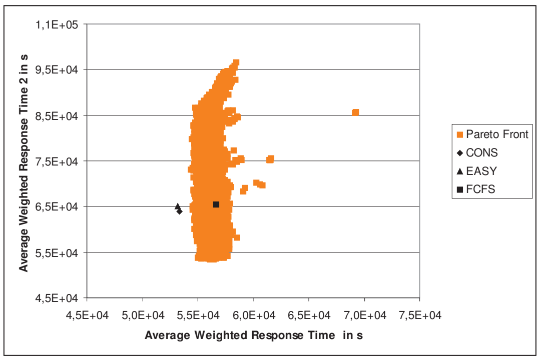
(u) AWRT5 vs. UTIL.

Appendix D

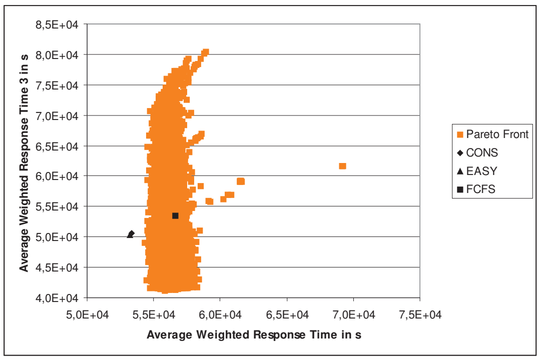
Reduced Pareto Front for the CTC Workload and Results for Standard Algorithms



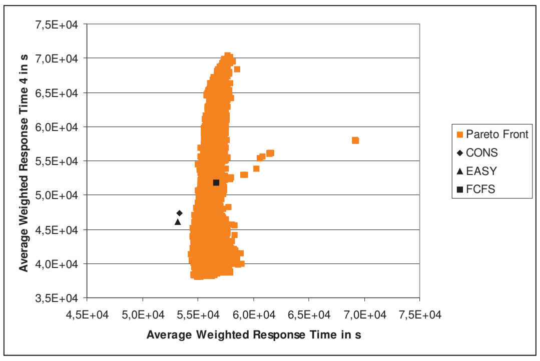
(a) AWRT vs. AWRT1.



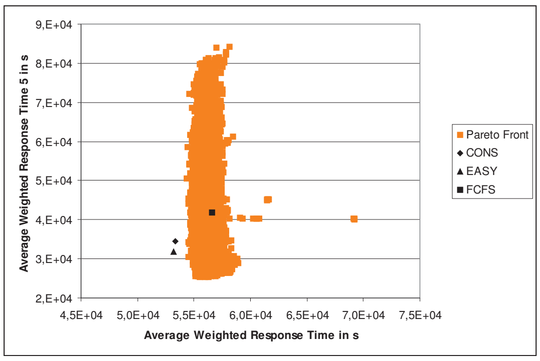
(b) AWRT vs. AWRT2.



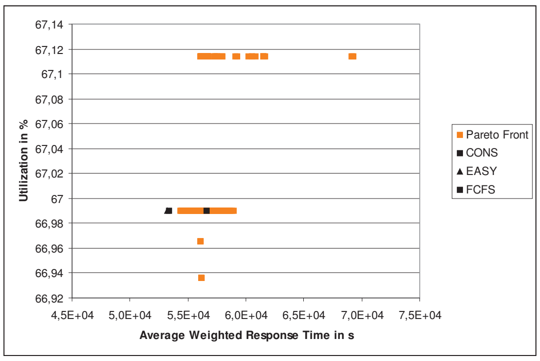
(c) AWRT vs. AWRT3.



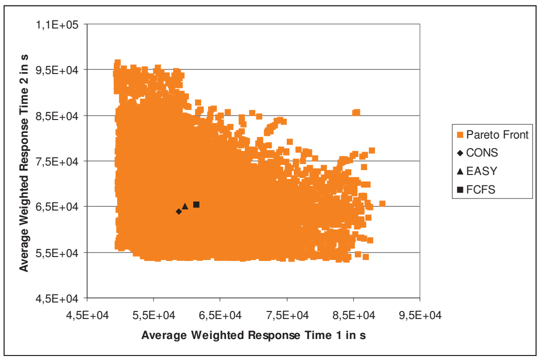
(d) AWRT vs. AWRT4.



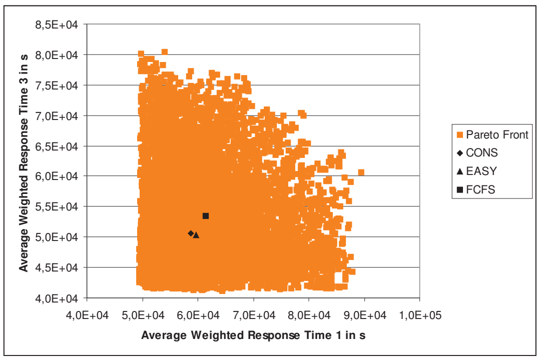
(e) AWRT vs. AWRT5.



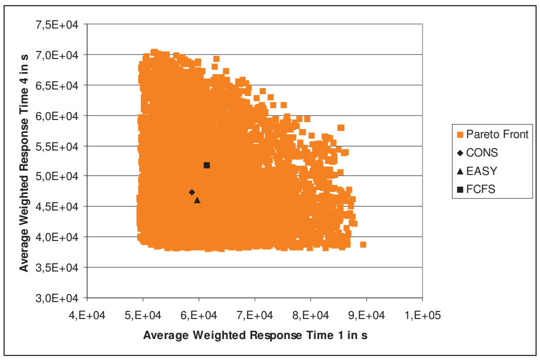
(f) AWRT vs. UTIL.



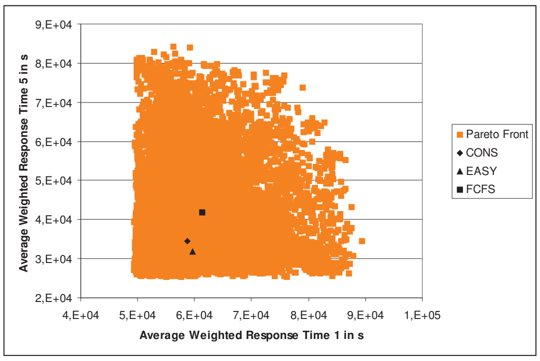
(g) AWRT1 vs. AWRT2.



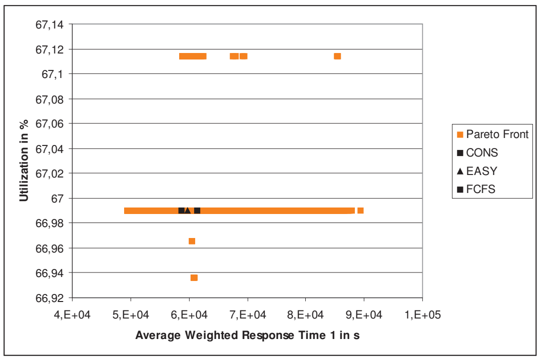
(h) AWRT1 vs. AWRT3.



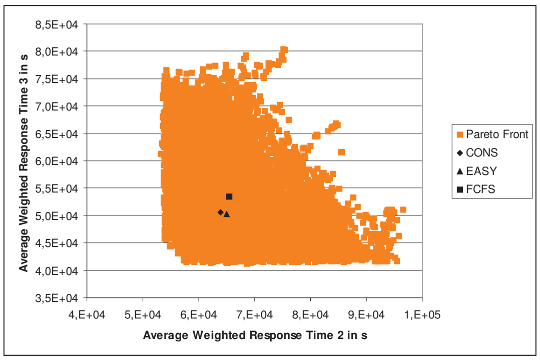
(i) AWRT1 vs. AWRT4.



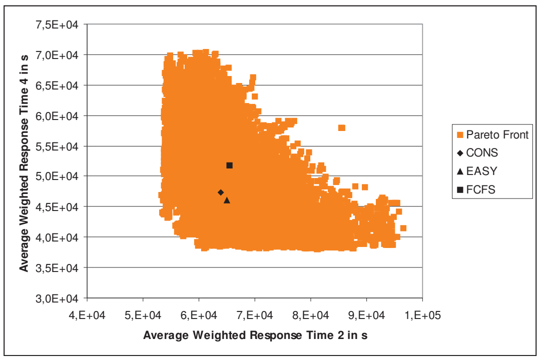
(j) AWRT1 vs. AWRT5.



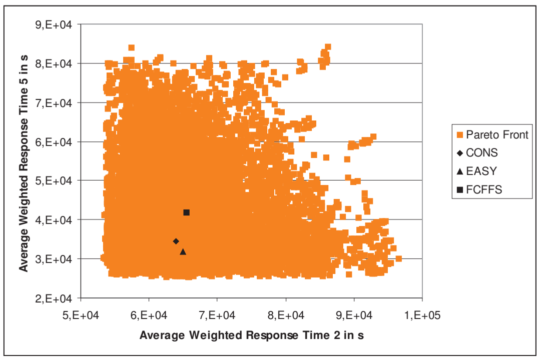
(k) AWRT1 vs. UTIL.



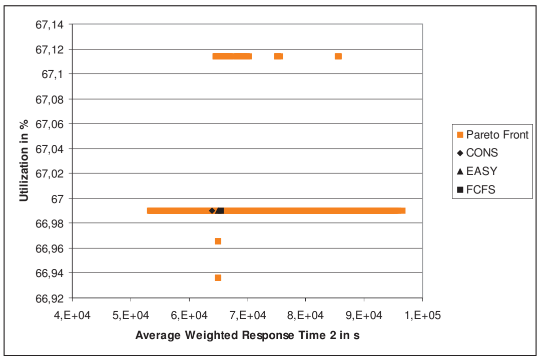
(l) AWRT2 vs. AWRT3.



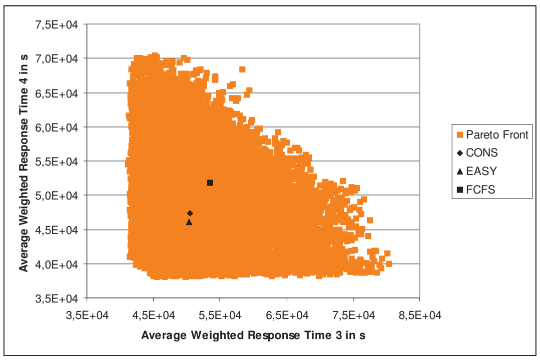
(m) AWRT2 vs. AWRT4.



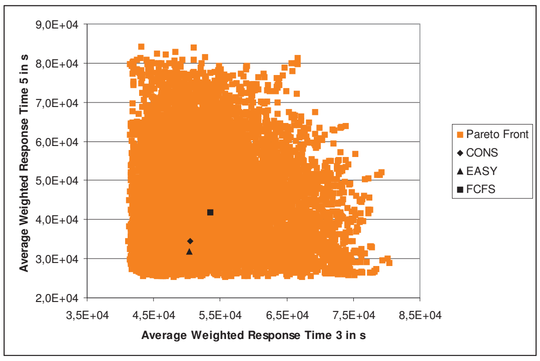
(n) AWRT2 vs. AWRT5.



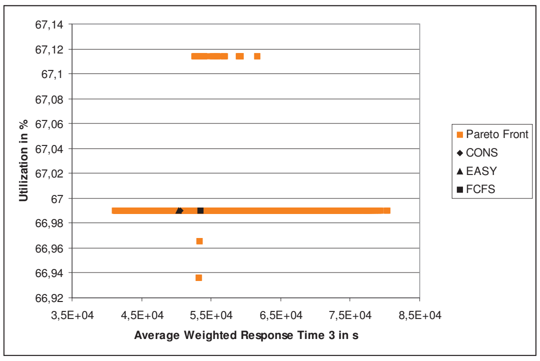
(o) AWRT2 vs. UTIL.



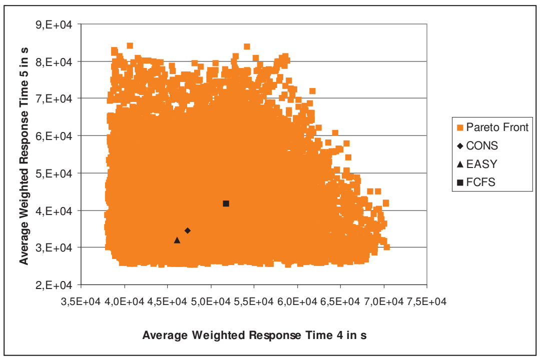
(p) AWRT3 vs. AWRT4.



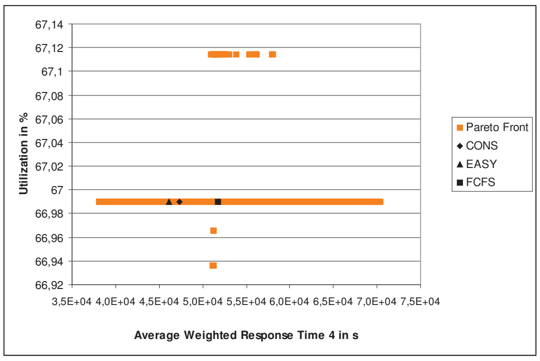
(q) AWRT3 vs. AWRT5.



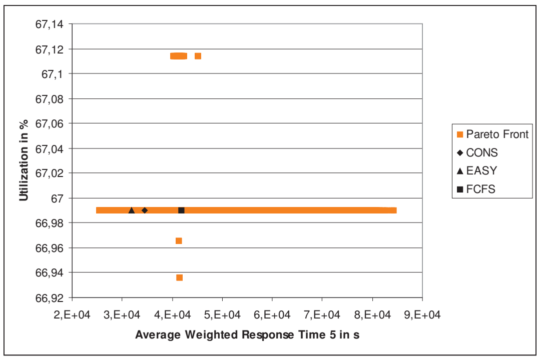
(r) AWRT3 vs. UTIL.



(s) AWRT4 vs. AWRT5.



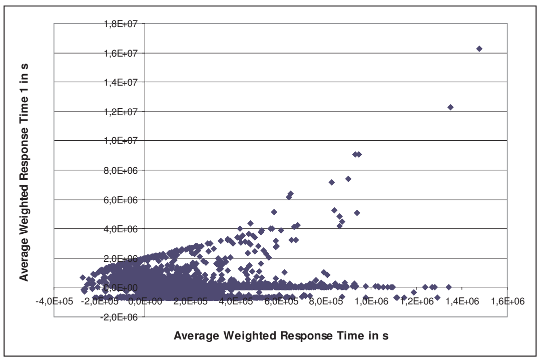
(t) AWRT4 vs. UTIL.



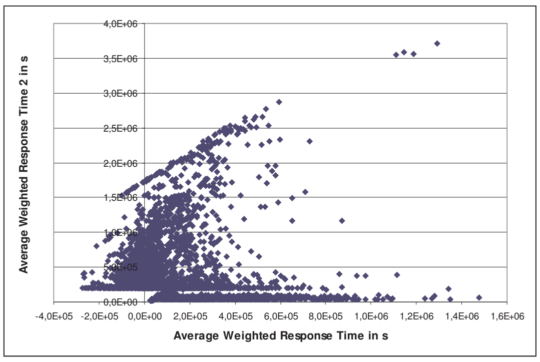
(u) AWRT5 vs. UTIL.

Appendix E

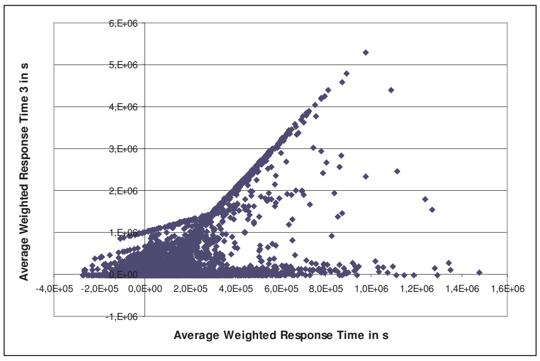
Reduced Pareto Front After Transformation using All Workloads



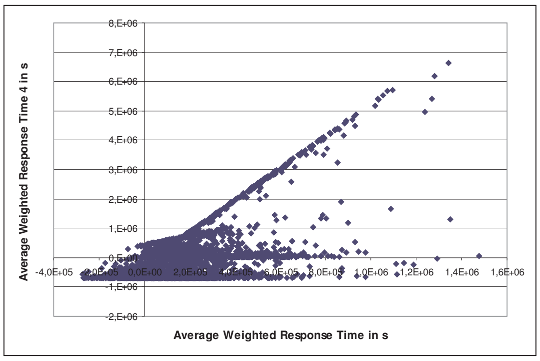
(a) AWRT vs. AWRT1.



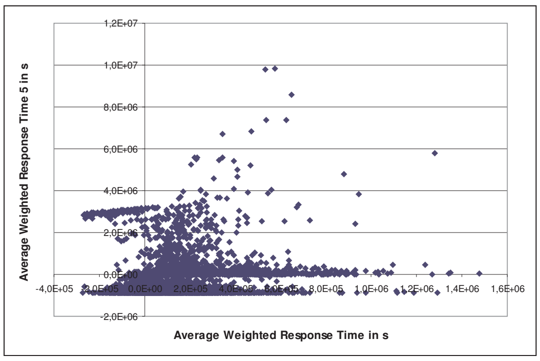
(b) AWRT vs. AWRT2.



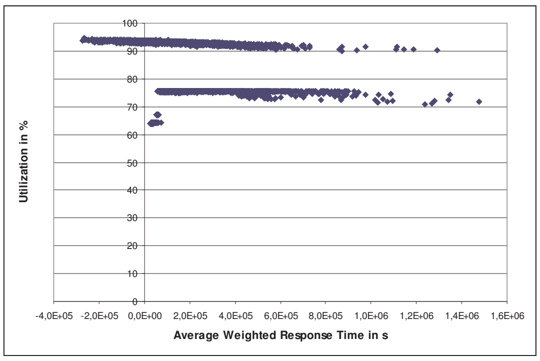
(c) AWRT vs. AWRT3.



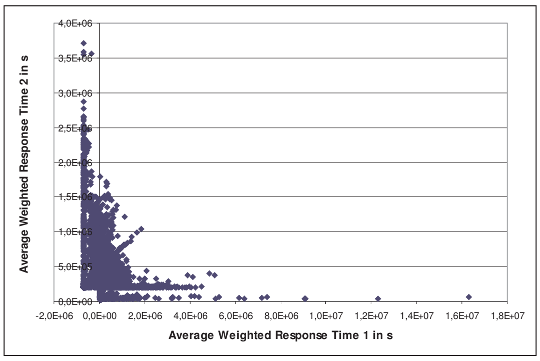
(d) AWRT vs. AWRT4.



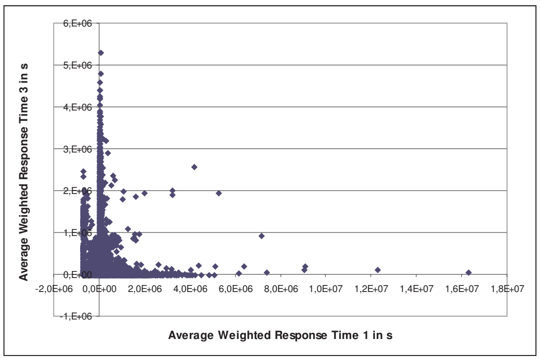
(e) AWRT vs. AWRT5.



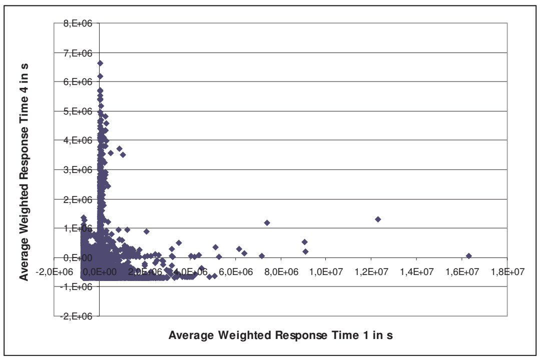
(f) AWRT vs. UTIL.



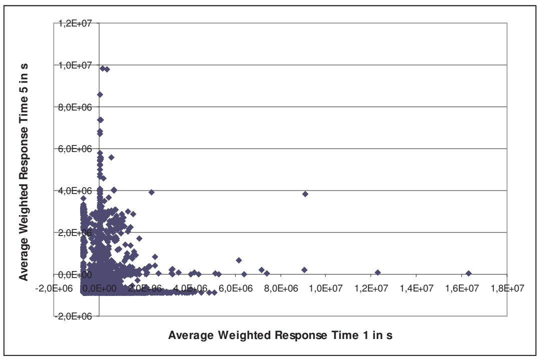
(g) AWRT1 vs. AWRT2.



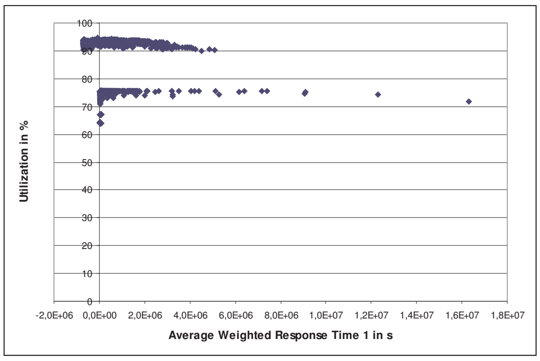
(h) AWRT1 vs. AWRT3.



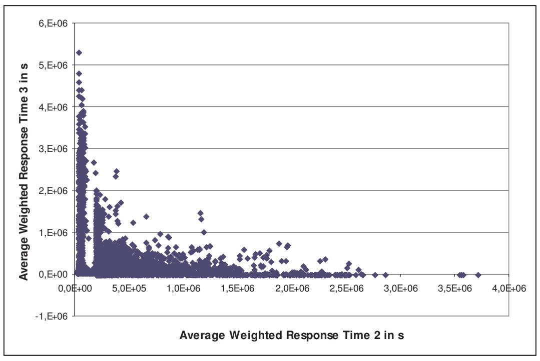
(i) AWRT1 vs. AWRT4.



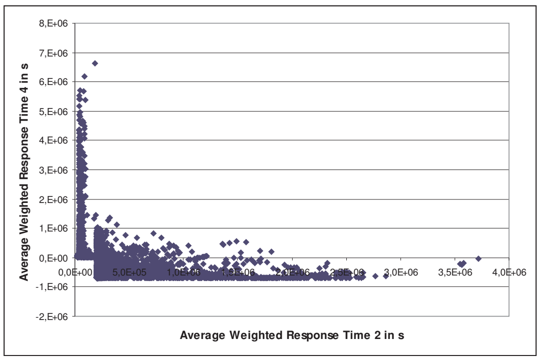
(j) AWRT1 vs. AWRT5.



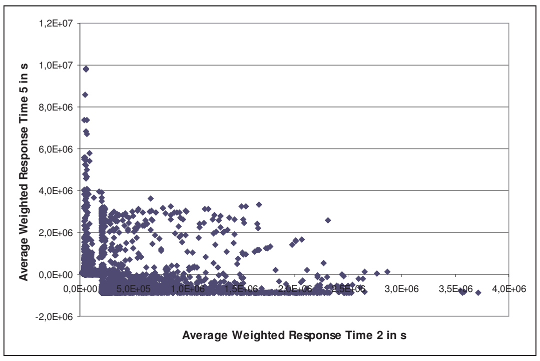
(k) AWRT1 vs. UTIL.



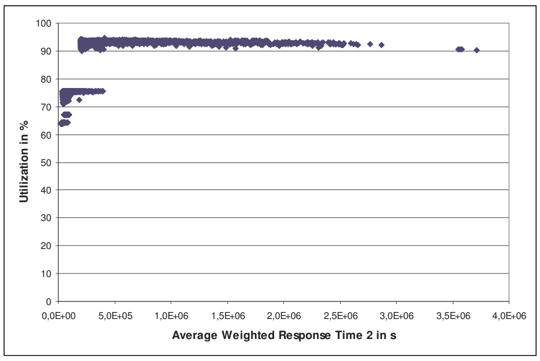
(l) AWRT2 vs. AWRT3.



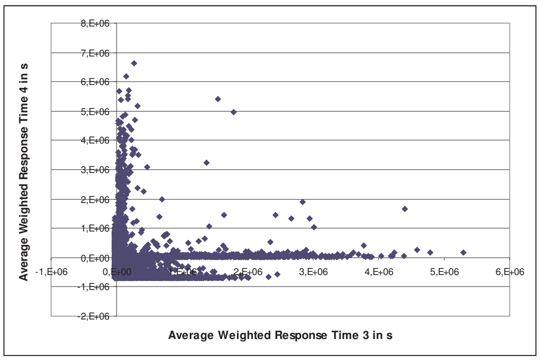
(m) AWRT2 vs. AWRT4.



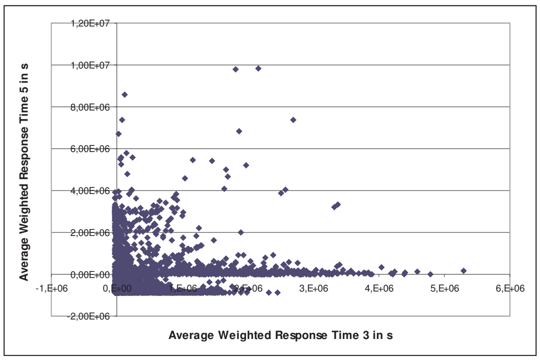
(n) AWRT2 vs. AWRT5.



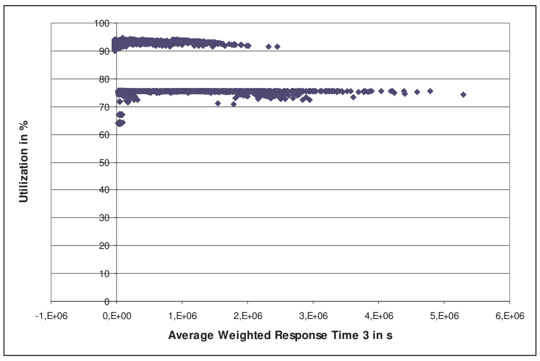
(o) AWRT2 vs. UTIL.



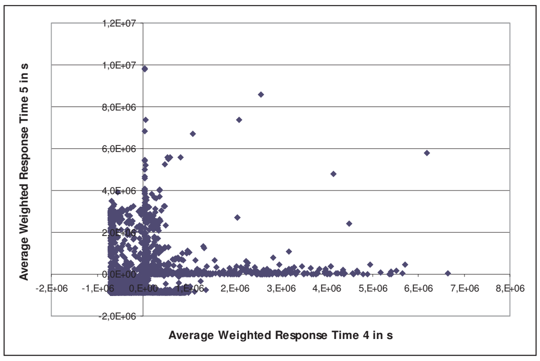
(p) AWRT3 vs. AWRT4.



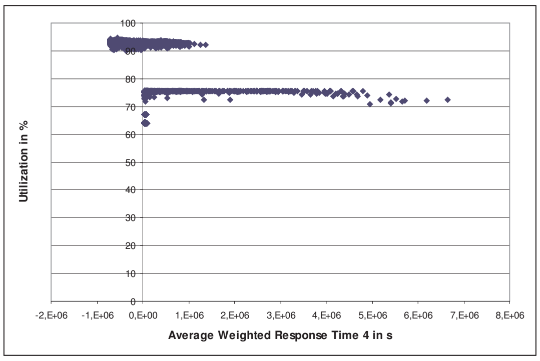
(q) AWRT3 vs. AWRT5.



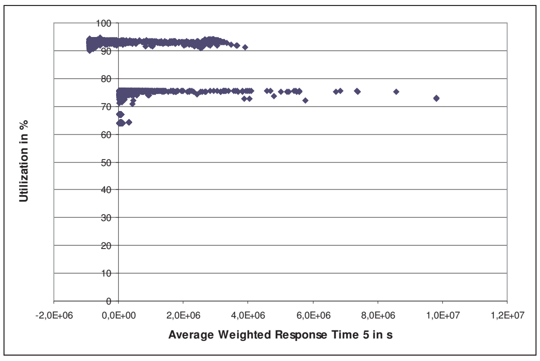
(r) AWRT3 vs. UTIL.



(s) AWRT4 vs. AWRT5.



(t) AWRT4 vs. UTIL.



(u) AWRT5 vs. UTIL.

Appendix F

Results for the Standard Scheduling Strategies

Strategy	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC								
CONS	53339.98	58721.59	63892.55	50544.89	47321.24	34493.34	66.99	842786.13
EASY	53186.81	59681.27	64976.06	50317.46	46120.01	31855.67	66.99	856717.01
FCFS	56628.18	61404.04	65466.71	53527.30	51754.98	41814.96	66.99	875907.26
KTH								
CONS	73673.07	65027.56	80929.91	56074.17	75756.39	42298.36	71.91	973995.33
EASY	72741.20	65780.96	79927.68	54751.16	74706.34	40079.55	71.91	977520.40
FCFS	121948.02	140110.55	123977.14	99766.99	130932.73	92891.79	71.91	1897014.06
LANL								
CONS	13047.71	12157.95	13845.04	10480.35	12261.62	19565.04	57.26	176959.77
EASY	12441.07	11873.23	13209.65	10182.37	11541.49	16987.81	57.26	171570.91
FCFS	14940.59	13587.08	15719.91	12209.99	14429.66	23723.52	57.26	198750.52
SDSC00								
CONS	380995.77	499218.09	427545.77	283933.96	324866.66	393598.87	83.88	6702363.98
EASY	112115.38	140315.14	131577.23	98227.48	88384.43	73899.22	84.65	1929460.32
FCFS	1447752.5	1977583.3	1648640.2	990922.68	1207998.1	1627589.8	79.86	26370393.8
SDSC95								
CONS	47180.46	45328.07	48836.64	53320.51	45998.57	40779.84	62.95	648627.33
EASY	46963.73	45506.81	48389.3	52621.21	44832.38	39545.96	62.95	648625.37
FCFS	48862.41	46257.04	50996.48	56207.66	48710.19	45785.27	62.95	666556.33
SDSC96								
CONS	47682.64	44840.21	44688.45	84376.58	48517.47	75383.57	61.14	627155.92
EASY	47629.55	44758.70	44904.93	84515.25	48140.66	71770.79	61.14	627206.84
FCFS	48340.80	45531.11	45084.50	84703.12	49886.58	78738.10	61.14	635649.15

Table F.1: Absolute results for all workloads using the standard algorithms.

Appendix G

Best Solutions within the Pareto Fronts of All Workloads

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	55846.41	49652.04	56330.98	60691.71	59698.30	32726.87	66.99	721844.27
KTH	832985.93	48027.37	76180.28	4332150.50	166533.76	298256.18	71.37	784994.80
LANL	15017.17	9493.29	11290.87	16244.49	32815.08	28287.38	57.27	140096.38
SDSC00	1809506.30	56859.21	81736.64	3304774.30	3563947.30	89549.66	78.86	895538.67
SDSC95	50393.29	41620.90	48839.23	66642.56	97271.85	52066.75	62.95	611565.90
SDSC96	48678.71	44924.89	37967.59	107826.84	57319.80	98239.65	61.14	601119.24

Table G.1: Absolute results for the best solutions, regarding to the defined linear objective function, within the Pareto fronts of all workloads. All AWRT values are given in seconds. The UTIL is given in %.

Appendix H

Results for Adapting a Greedy Strategy

H.1 Results for Optimizing a Greedy Scheduling Strategy using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
SOPT(CTC) applied to CTC, linear objective							
319213.77	81372.52	138048.78	621559.19	698099.64	210240.23	71.92	1365920.33
SOPT(CTC) applied to CTC, region based objective							
654651.62	140146.21	529618.99	772436.29	1155975.50	786146.82	71.36	3519938.06
SOPT(CTC) applied to KTH, linear objective							
319213.77	81372.52	138048.78	621559.19	698099.64	210240.23	71.92	1365920.33
SOPT(CTC) applied to KTH, region based objective							
654651.62	140146.21	529618.99	772436.29	1155975.50	786146.82	71.36	3519938.06
SOPT(CTC) applied to LANL, linear objective							
15179.50	10881.21	13173.43	15576.71	22058.94	37017.68	57.27	161505.85
SOPT(CTC) applied to LANL, region based objective							
15073.44	12315.83	13766.07	14668.02	19678.65	30805.13	57.27	178222.55
SOPT(CTC) applied to SDSC00, linear objective							
1379456.80	142186.91	167517.78	1668284.90	3303098.20	1197680.40	79.62	2091940.22
SOPT(CTC) applied to SDSC00, region based objective							
1535197.60	470185.88	1234798.10	1884381.70	2422670.80	2156024.40	78.31	9641051.20
SOPT(CTC) applied to SDSC95, linear objective							
48976.58	44488.02	49990.19	62851.18	57985.21	48863.95	62.95	644840.93
SOPT(CTC) applied to SDSC95, region based objective							
48578.54	44862.41	49349.33	60567.23	56188.69	45581.69	62.95	646021.38
SOPT(CTC) applied to SDSC96, linear objective							
48612.16	44593.22	47432.77	93182.98	53008.97	80527.75	61.14	635663.22
SOPT(CTC) applied to SDSC96, region based objective							
48573.59	44841.31	47005.92	92547.13	48558.91	78962.25	61.14	636436.81

Table H.1: Results for all workloads using the Greedy strategy optimized for the CTC workload trace. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC, linear objective								
CONS	-3.55	10.16	3.04	-11.34	-14.15	-1.72	0	8.00
EASY	-3.85	11.60	4.66	-11.84	-17.12	-10.14	0	9.50
FCFS	2.46	14.08	5.38	-5.13	-4.37	16.09	0	11.48
SOPT(CTC) applied to CTC, region based objective								
CONS	-3.94	8.77	0.85	-16.24	-8.00	-8.74	0	6.37
EASY	-4.24	10.24	2.50	-16.77	-10.81	-17.75	0	7.89
FCFS	2.09	12.76	3.23	-9.76	1.25	10.30	0	9.91
SOPT(CTC) applied to KTH, linear objective								
CONS	-333.28	-25.14	-70.58	-1008.46	-821.51	-397.04	0	-40.24
EASY	-338.835	-23.70	-72.72	-1035.24	-834.46	-424.56	0	-39.73
FCFS	-161.76	41.92	-11.35	-523.01	-433.17	-126.3	0	28.0
SOPT(CTC) applied to KTH, region based objective								
CONS	-788.59	-115.52	-554.42	-1277.53	-1425.91	-1758.57	0.77	-261.39
EASY	-799.97	-113.05	-562.62	-1310.81	-1447.36	-1861.47	0.77	-260.09
FCFS	-436.83	-0.03	-327.19	-674.24	-782.88	-746.30	0.77	-85.55
SOPT(CTC) applied to LANL, linear objective								
CONS	-8.57	14.78	10.97	-39.69	-64.87	-66.47	0	13.57
EASY	-9.10	14.62	10.52	-40.24	-66.18	-67.75	0	13.32
FCFS	-1.60	19.92	16.20	-27.57	-52.87	-56.04	0	18.74
SOPT(CTC) applied to LANL, region based objective								
CONS	-7.82	3.55	6.97	-31.54	-47.08	-38.54	0	4.63
EASY	-8.34	3.36	6.49	-32.05	-48.25	-39.59	0	4.35
FCFS	-0.89	9.36	12.43	-20.13	-36.38	-29.85	0	10.33
SOPT(CTC) applied to SDSC00, linear objective								
CONS	-262.07	-43.31	60.82	-487.56	-916.76	-204.29	5.08	22.59
EASY	-1130.39	-1.33	-27.32	-1598.39	-3637.19	-1520.69	5.94	-8.42
FCFS	4.72	92.81	89.84	-68.36	-173.44	26.41	0.31	92.07
SOPT(CTC) applied to SDSC00, region based objective								
CONS	-302.94	-373.89	-188.81	-563.67	-645.74	-447.77	6.65	-256.76
EASY	-1269.30	-235.09	-838.46	-1818.39	-2641.06	-2817.52	7.49	-399.68
FCFS	-6.04	76.22	25.10	-90.16	-100.55	-32.47	1.96	63.44
SOPT(CTC) applied to SDSC95, linear objective								
CONS	-1.47	3.08	0.33	-13.78	-20.76	-8.86	0	2.25
EASY	-1.48	3.13	0.32	-13.81	-21.00	-9.38	0	2.27
FCFS	-0.23	3.82	1.97	-11.82	-19.04	-6.72	0	3.26
SOPT(CTC) applied to SDSC95, region based selection								
CONS	-0.65	2.27	1.61	-9.64	-17.01	-1.54	0	2.07
EASY	-0.65	2.31	1.60	-9.67	-17.25	-2.03	0	2.10
FCFS	0.58	3.01	3.23	-7.76	-15.35	0.44	0	3.08
SOPT(CTC) applied to SDSC96, linear objective								
CONS	-1.95	0.55	-6.14	-10.44	-9.26	-6.82	-0.07	-1.36
EASY	-2.06	0.37	-5.63	-10.26	-10.11	-12.20	-0.07	-1.35
FCFS	-0.56	2.06	-5.21	-10.01	-6.26	-2.27	-0.07	0
SOPT(CTC) applied to SDSC96, region based objective								
CONS	-1.87	0	-5.19	-9.68	-0.09	-4.75	-0.07	-1.48
EASY	-1.98	-0.18	-4.68	-9.50	-0.87	-10.02	-0.07	-1.47
FCFS	-0.48	1.52	-4.26	-9.26	2.66	-0.28	-0.07	-0.12

Table H.2: Relative results for all workloads using the Greedy strategy optimized for the CTC workload trace.

H.2 Optimizing a Greedy Strategy for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
ALLOPT applied to CTC, linear objective							
55301.26	55104.02	61644.73	56354.42	53310.46	34913.51	66.99	797619.16
ALLOPT applied to KTH, linear objective							
173973.49	69373.98	101523.67	359589.91	271387.74	88983.09	71.92	1099834.44
ALLOPT applied to LANL, linear objective							
15226.17	11296.57	12836.71	16418.06	21332.37	38669.80	57.27	164312.57
ALLOPT applied to SDSC00, linear objective							
1128705.50	106215.53	103511.90	1568765.40	2633270.80	160564.69	79.37	1476202.90
ALLOPT applied to SDSC95, linear objective							
48686.53	45167.61	48470.83	62108.00	57228.19	46793.74	62.95	645559.46
ALLOPT applied to SDSC96, linear objective							
48496.24	44966.97	44760.14	92746.26	54717.88	75382.74	61.14	628710.23

Table H.3: Results for all workloads using the Greedy strategy optimized for all workloads together. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj\,[\%]}$
ALLOPT applied to CTC, linear objective								
CONS	-3.68	6.16	3.52	-11.49	-12.66	-1.22	0	5.36
EASY	-3.98	7.67	5.13	-12.00	-15.59	-9.60	0	6.90
FCFS	2.34	10.26	5.84	-5.28	-3.01	16.50	0	8.94
ALLOPT applied to KTH, linear objective								
CONS	-136.14	-6.68	-25.45	-541.28	-258.24	-110.37	0	-12.92
EASY	-139.17	-5.46	-27.02	-556.77	-263.27	-122.02	0	-12.51
FCFS	-42.66	50.49	18.11	-260.43	-107.27	4.21	0	42.02
ALLOPT applied to LANL, linear objective								
CONS	-8.91	11.53	13.25	-47.24	-59.44	-73.90	0	12.07
EASY	-9.43	11.36	12.80	-47.81	-60.71	-75.23	0	11.82
FCFS	-1.91	16.86	18.34	-34.46	-47.84	-63.00	0	17.33
ALLOPT applied SDSC00, linear objective								
CONS	-196.25	-7.05	75.79	-452.51	-710.57	59.21	5.39	45.37
EASY	-906.74	24.30	21.33	-1497.07	-2879.34	-117.28	6.24	23.49
FCFS	22.04	94.63	93.72	-58.31	-117.99	90.13	0.63	94.40
ALLOPT applied to SDSC95, linear objective								
CONS	-0.87	1.60	3.36	-12.43	-19.18	-4.24	0	2.14
EASY	-0.88	1.65	3.35	-12.46	-19.42	-4.74	0	2.17
FCFS	0.36	2.36	4.95	-10.50	-17.49	-2.20	0	3.15
ALLOPT applied to SDSC96, linear objective								
CONS	-1.71	-0.28	-0.16	-9.92	-12.78	0	-0.07	-0.25
EASY	-1.82	-0.47	0.32	-9.74	-13.66	-5.03	-0.07	-0.24
FCFS	-0.32	1.24	0.72	-9.50	-9.68	4.26	-0.07	1.09

Table H.4: Relative results for all workloads by optimizing the Greedy algorithms for all workloads in parallel by averaging the objective values.

Appendix I

Results for the Iterative Approach

I.1 Results for Optimizing the Iterative Approach for All Individual Workload Traces

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	54400.76	50975.42	59358.16	53566.2	54315.07	44734.47	67	747186.8
KTH	85969.92	69280.74	92753.13	71047.02	89961.7	64283.16	71.92	1063819.92
LANL	14805.47	9621.79	11648.92	14331.4	16472.52	69217.55	57.27	142813.58
SDSC00	766506.73	132819.73	779247.7	803640.45	1299342	1422446.6	82.89	4445188.1
SDSC95	47678.57	44372.18	50265.6	54946.19	51074.52	41411.24	62.95	644784.2
SDSC96	48320.74	44712.95	42472.13	100347.07	48990.32	74794.47	61.1	617018.06

Table I.1: Absolute results for all workloads using the iterative approach. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-1.99	13.19	7.1	-5.98	-14.78	-29.69	-0.01	11.34
EASY	-2.28	14.59	8.65	-6.46	-17.77	-40.43	-0.01	12.78
FCFS	3.93	16.98	9.33	-0.07	-4.95	-6.98	-0.01	14.7
SOPT(KTH) applied to KTH								
CONS	-16.69	-6.54	-14.61	-26.7	-18.75	-51.98	0	-9.22
EASY	-18.19	-5.32	-16.05	-29.76	-20.42	-60.39	0	-8.83
FCFS	29.5	50.55	25.19	28.79	31.29	30.8	0	43.92
SOPT(LANL) applied to LANL								
CONS	-13.47	20.86	15.86	-36.75	-34.34	-253.78	0	19.3
EASY	-19	18.96	11.82	-40.75	-42.72	-307.45	0	16.76
FCFS	0.9	29.18	25.9	-17.37	-14.16	-191.77	0	28.14
SOPT(SDSC00) applied to SDSC00								
CONS	-101.19	73.39	-82.26	-183.04	-299.96	-261.39	1.19	33.68
EASY	-583.68	5.34	-492.24	-718.14	-1370.1	-1824.85	2.08	-130.39
FCFS	47.06	93.28	52.73	18.9	-7.56	12.6	-3.78	83.14
SOPT(SDSC95) applied to SDSC95								
CONS	-1.06	2.11	-2.93	-3.05	-11.04	-1.55	0	0.59
EASY	-1.52	2.49	-3.88	-4.42	-13.92	-4.72	0	0.59
FCFS	2.42	4.07	1.43	2.24	-4.85	9.55	0	3.27
SOPT(SDSC96) applied to SDSC96								

Continued on next page

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
CONS	-1.34	0.28	4.96	-18.93	-0.97	0.78	0.07	1.62
EASY	-1.45	0.1	5.42	-18.73	-1.76	-4.21	0.07	1.62
FCFS	0.04	1.8	5.79	-18.47	1.8	5.01	0.07	2.93

Table I.2: Relative results for all workloads using the iterative approach in comparison to the standard algorithms. The resulting scheduling strategies are optimized for all workload traces individually.

I.2 Results for Optimizing the Iterative Approach using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	54400.76	50975.42	59358.16	53566.2	54315.07	44734.47	66.99	747186.8
KTH	161298.12	71753.14	108834.6	168243.75	345432.62	281573.22	71.8	1152869.76
LANL	14446.73	10922.24	14368.75	12877.66	17934.26	28341.27	57.27	166697.38
SDSC00	1168152	530464.41	883302.75	1028014.2	1971139.8	2843025.2	80.4	8837855.1
SDSC95	48289.04	43677.53	49312.87	61714.44	58540.08	49056.08	62.95	634026.83
SDSC96	48726.86	44773.7	49018.88	88050.39	56446.85	80015.55	61.14	643812.52

Table I.3: Absolute results for all workloads using the iterative approach with the CTC optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-1.99	13.19	7.1	-5.98	-14.78	-29.69	0	11.34
EASY	-2.28	14.59	8.65	-6.46	-17.77	-40.43	0	12.78
FCFS	3.93	16.98	9.33	-0.07	-4.95	-6.98	0	14.7
SOPT(CTC) applied to KTH								
CONS	-118.94	-10.34	-34.48	-200.04	-355.98	-565.68	0.16	-18.37
EASY	-121.74	-9.08	-36.17	-207.29	-362.39	-602.54	0.16	-17.94
FCFS	-32.27	48.79	12.21	-68.64	-163.82	-203.12	0.16	39.23
SOPT(CTC) applied to LANL								
CONS	-10.72	10.16	-3.78	-22.87	-46.26	-44.86	0	5.8
EASY	-16.12	8.01	-8.77	-26.47	-55.39	-66.83	0	2.84
FCFS	3.31	19.61	8.6	-5.47	-24.29	-19.46	0	16.13
SOPT(CTC) applied to SDSC00								
CONS	-206.6	-6.26	-106.6	-262.06	-506.75	-622.32	4.15	-31.86
EASY	-941.92	-278.05	-571.32	-946.56	-2130.19	-3747.16	5.02	-358.05
FCFS	19.31	73.18	46.42	-3.74	-63.17	-74.68	-0.67	66.49
SOPT(CTC) applied to SDSC95								
CONS	-2.35	3.64	-0.98	-15.74	-27.26	-20.29	0	2.25
EASY	-2.82	4.02	-1.91	-17.28	-30.58	-24.05	0	2.25
FCFS	1.17	5.58	3.3	-9.8	-20.18	-7.14	0	4.88
SOPT(CTC) applied to SDSC96								
CONS	-2.19	0.15	-9.69	-4.35	-16.34	-6.14	0	-2.66
EASY	-2.3	-0.03	-9.16	-4.18	-17.25	-11.49	0	-2.65
FCFS	-0.8	1.66	-8.73	-3.95	-13.15	-1.62	0	-1.28

Table I.4: Relative results for all workloads using the iterative approach with the CTC optimized settings in comparison to the standard algorithms.

I.3 Optimization using the Iterative Approach for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	55483.98	58551.8	68697.13	51446.34	48908.24	38140.78	66.99	860306.5
KTH	94015.54	86456.82	102522.4	72038.11	95728.07	66072.49	71.92	1274657.7
LANL	14928.39	12292.97	15350.09	11386.62	17035.39	29166.68	57.27	184330.11
SDSC00	522934.58	639873.6	621761.22	379899.84	478682.58	545000	82.68	8885780.8
SDSC95	48304.22	44767.66	51473.35	53849.09	52238.59	46277.13	62.95	653570
SDSC96	48163.52	45028.72	43656.17	89902.65	52717.47	78141.35	61.14	624911.86

Table I.5: Absolute results for all workloads using the iterative approach. The optimization uses all workload traces together. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-4.02	0.29	-7.52	-1.78	-3.35	-10.57	0	-2.08
EASY	-4.32	1.89	-5.73	-2.24	-6.05	-19.73	0	-0.42
FCFS	2.02	4.65	-4.93	3.89	5.5	8.79	0	1.78
SOPT(KTH) applied to KTH								
CONS	-27.61	-32.95	-26.68	-28.47	-26.36	-56.21	0	-30.87
EASY	-29.25	-31.43	-28.27	-31.57	-28.14	-64.85	0	-30.4
FCFS	22.91	38.29	17.31	27.79	26.89	28.87	0	32.81
SOPT(LANL) applied to LANL								
CONS	-14.41	-1.11	-10.87	-8.65	-38.93	-49.08	0	-4.16
EASY	-19.99	-3.54	-16.2	-11.83	-47.6	-71.69	0	-7.44
FCFS	0.08	9.52	2.35	6.74	-18.06	-22.94	0	7.26
SOPT(SDSC00) applied to SDSC00								
CONS	-37.25	-28.18	-45.43	-33.8	-47.35	-38.56	1.43	-32.58
EASY	-366.43	-356.03	-372.54	-286.76	-441.59	-638	2.32	-360.53
FCFS	63.88	67.64	62.29	61.66	60.37	66.49	-3.53	66.3
SOPT(SDSC95) applied to SDSC95								
CONS	-2.38	1.24	-5.4	-0.99	-13.57	-13.48	0	-0.76
EASY	-2.85	1.62	-6.37	-2.33	-16.52	-17.02	0	-0.76
FCFS	1.14	3.22	-0.94	4.2	-7.24	-1.07	0	1.95
SOPT(SDSC96) applied to SDSC96								
CONS	-1.01	-0.42	2.31	-6.55	-8.66	-3.66	0	0.36
EASY	-1.12	-0.6	2.78	-6.37	-9.51	-8.88	0	0.37
FCFS	0.37	1.1	3.17	-6.14	-5.67	0.76	0	1.69

Table I.6: Relative results for all workloads using the iterative approach in comparison to the standard algorithms. The optimization uses all workloads together.

Appendix J

Results for the Probability based Approach

J.1 Results for Optimizing the Probability based Approach for All Individual Workload Traces

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	54535.76	53780.18	59448.48	53185.9	53417.77	45390.11	66.99	775595.77
KTH	88043.96	60596.25	93862.87	72500.73	101408.38	59037.76	71.92	981413.97
LANL	14780.98	9997.5	12536.12	14323.03	18923.72	51405.94	57.27	150119.51
SDSC00	838340.38	69673.97	115066.63	371471.72	2452272.8	2580000	81.66	1157006.19
SDSC95	47937.31	42870.26	50308.19	59237.94	58109.18	49094.22	62.95	629935.38
SDSC96	48369.99	44727.38	43228.86	99394	48493.67	75568.73	61.14	620189.24

Table J.1: Absolute results for all workloads using the probability based procedure. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-2.24	8.41	6.96	-5.23	-12.88	-31.59	0	7.97
EASY	-2.54	9.89	8.51	-5.7	-15.82	-42.49	0	9.47
FCFS	3.7	12.42	9.19	0.64	-3.21	-8.55	0	11.45
SOPT(KTH) applied to KTH								
CONS	-19.51	6.81	-15.98	-29.29	-33.86	-39.57	0	-0.76
EASY	-21.04	7.88	-17.43	-32.42	-35.74	-47.3	0	-0.4
FCFS	27.8	56.75	24.29	27.33	22.55	36.44	0	48.27
SOPT(LANL) applied to LANL								
CONS	-13.28	17.77	9.45	-36.67	-54.33	-162.74	0	15.17
EASY	-18.81	15.8	5.1	-40.66	-63.96	-202.6	0	12.5
FCFS	1.07	26.42	20.25	-17.31	-31.14	-116.69	0	24.47
Optimized for SDSC00								
CONS	-120.04	86.04	73.09	-30.83	-654.86	-554.79	2.66	82.74
EASY	-647.75	50.34	12.55	-278.17	-2674.55	-3387.5	3.54	40.03
FCFS	42.09	96.48	93.02	62.51	-103	-58.35	-2.24	95.61
SOPT(SDSC95) applied to SDSC95								
CONS	-1.6	5.42	-3.01	-11.1	-26.33	-20.39	0	2.88

Continued on next page

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
EASY	-2.07	5.79	-3.97	-12.57	-29.61	-24.14	0	2.88
FCFS	1.89	7.32	1.35	-5.39	-19.3	-7.23	0	5.49
SOPT(SDSC96) applied to SDSC96								
CONS	-1.44	0.25	3.27	-17.8	0.05	-0.25	0	1.11
EASY	-1.55	0.07	3.73	-17.6	-0.73	-5.29	0	1.12
FCFS	-0.06	1.77	4.12	-17.34	2.79	4.03	0	2.43

Table J.2: Relative results for all workloads using the probability based procedure in comparison to the standard algorithms.

J.2 Results for Optimizing the Probability based Approach using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	54535.76	53780.18	59448.48	53185.9	53417.77	45390.11	66.99	775595.72
KTH	483867.34	72448.69	219202.93	500402.74	1427585	901525.28	71.92	1601298.66
LANL	14451.97	10186.93	14515.58	13144.97	18531.45	28247.5	57.27	159931.61
SDSC00	1560737.8	372139.36	1602530.1	1930339.5	2198176.1	2930000	80.25	10131514
SDSC95	47823.23	42400.9	49790.97	61088.9	59876.93	48054.78	62.95	623172.83
SDSC96	48257.94	43496.01	51800.84	89624.77	56190.12	80805.12	61.14	642163.47

Table J.3: Absolute results for all workloads using the probabilistic approach with the CTC optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-2.24	8.41	6.96	-5.23	-12.88	-31.59	0	7.97
EASY	-2.54	9.89	8.51	-5.7	-15.82	-42.49	0	9.47
FCFS	3.7	12.42	9.19	0.64	-3.21	-8.55	0	11.45
SOPT(CTC) applied to KTH								
CONS	-556.78	-11.41	-170.86	-792.39	-1784.44	-2031.35	0	-64.41
EASY	-565.19	-10.14	-174.25	-813.96	-1810.93	-2149.34	0	-63.81
FCFS	-296.78	48.29	-76.81	-401.57	-990.32	-870.51	0	15.59
SOPT(CTC) applied to LANL								
CONS	-10.76	16.21	-4.84	-25.42	-51.13	-44.38	0	9.62
EASY	-16.16	14.2	-9.89	-29.1	-60.56	-66.28	0	6.78
FCFS	3.27	25.02	7.66	-7.66	-28.43	-19.07	0	19.53
SOPT(CTC) applied to SDSC00								
CONS	-309.65	25.46	-274.82	-579.86	-576.64	-644.87	4.34	-51.16
EASY	-1292.08	-165.22	-1117.94	-1865.17	-2387.06	-3867.31	5.2	-425.1
FCFS	-7.8	81.18	2.8	-94.8	-81.97	-80.13	-0.48	61.58
SOPT(CTC) applied to SDSC95								
CONS	-1.36	6.46	-1.95	-14.57	-30.17	-17.84	0	3.92
EASY	-1.83	6.83	-2.9	-16.09	-33.56	-21.52	0	3.92
FCFS	2.13	8.34	2.36	-8.68	-22.92	-4.96	0	6.51
SOPT(CTC) applied to SDSC96								
CONS	-1.21	3	-15.92	-6.22	-15.81	-7.19	0	-2.39
EASY	-1.32	2.82	-15.36	-6.05	-16.72	-12.59	0	-2.38

Continued on next page

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
FCFS	0.17	4.47	-14.9	-5.81	-12.64	-2.63	0	-1.02

Table J.4: Relative results for all workloads using the probabilistic approach with the CTC optimized settings in comparison to the standard algorithms.

J.3 Optimization using the Probability based Approach for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	53834.34	56843.27	73508.78	49069.06	43436.79	31121.03	66.99	862467.85
KTH	108101.90	148512.76	113205.80	78159.96	105153.19	71543.71	71.92	1937950.8
LANL	15094.02	12318.24	15660.68	12330.11	15780.78	30006.74	57.27	185825.08
SDSC00	1180185.3	1488859.2	1513779	897579.84	945511.06	965060.64	80.52	20943708
SDSC95	49766.82	47620.09	52848.39	53599.35	47271.91	43745.87	62.95	687594.47
SDSC96	48547.26	45407.40	46120.57	86365.70	52350.01	75291.59	61.14	638556.24

Table J.5: Absolute results for all workloads using the probabilistic approach. The optimization uses all workload traces together. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-0.93	3.20	-15.05	2.92	8.21	9.78	0.00	-2.34
EASY	-1.22	4.76	-13.13	2.48	5.82	2.31	0.00	-0.67
FCFS	4.93	7.43	-12.28	8.33	16.07	25.57	0.00	1.53
SOPT(CTC) applied to KTH								
CONS	-46.73	-128.38	-39.88	-39.39	-38.80	-69.14	0.00	-98.97
EASY	-48.61	-125.77	-41.64	-42.75	-40.76	-78.50	0.00	-98.25
FCFS	11.35	-6.00	8.69	21.66	19.69	22.98	0.00	-2.16
SOPT(CTC) applied to LANL								
CONS	-15.68	-1.32	-13.11	-17.65	-28.70	-53.37	0.00	-5.01
EASY	-21.32	-3.75	-18.55	-21.09	-36.73	-76.64	0.00	-8.31
FCFS	-1.03	9.34	0.38	-0.98	-9.36	-26.49	0.00	6.50
SOPT(CTC) applied to SDSC00								
CONS	-209.76	-198.24	-254.06	-216.12	-191.05	-145.19	4.02	-212.48
EASY	-952.65	-961.08	-1050.49	-813.78	-969.77	-1205.91	4.88	-985.47
FCFS	18.48	24.71	8.18	9.42	21.73	40.71	-0.81	20.58
SOPT(CTC) applied to SDSC95								
CONS	-5.48	-5.06	-8.21	-0.52	-2.77	-7.27	0.00	-6.01
EASY	-5.97	-4.64	-9.22	-1.86	-5.44	-10.62	0.00	-6.01
FCFS	-1.85	-2.95	-3.63	4.64	2.95	4.45	0.00	-3.16
SOPT(CTC) applied to SDSC96								
CONS	-1.81	-1.26	-3.20	-2.36	-7.90	0.12	0.00	-1.82
EASY	-1.93	-1.45	-2.71	-2.19	-8.74	-4.91	0.00	-1.81
FCFS	-0.43	0.27	-2.30	-1.96	-4.94	4.38	0.00	-0.46

Table J.6: Relative results for all workloads using the probabilistic approach in comparison to the standard algorithms. The optimization uses all workloads together.

Appendix K

Results for the Michigan Approach

K.1 Results for Optimizing the Michigan Approach for All Individual Workload Traces

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	56067.5	49750.49	56705.27	49313.61	58276.98	80794.31	66.99	724325.94
KTH	105019.36	48891.73	73412.39	82851.13	215939.66	479282.09	71.92	782566.89
LANL	14123.25	9532.76	11251.01	12556.67	17037.52	62746.78	57.27	140331.68
SDSC00	1067404	55682.35	76868.81	103289.86	2353014.5	12000000	80.5	864298.69
SDSC95	47439.84	41380.64	48197.06	63535.15	64261.04	55634.23	62.95	606594.66
SDSC96	48238.37	43880.87	45836.32	98219.99	55418.29	81347.34	61.14	622153.91

Table K.1: Absolute results for all workloads using the Symbiotic Evolution. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.11	15.28	11.25	2.44	-23.15	-134.23	0	14.06
EASY	-5.42	16.64	12.73	2	-26.36	-153.63	0	15.45
FCFS	0.99	18.98	13.38	7.87	-12.6	-93.22	0	17.31
SOPT(KTH) applied to KTH								
CONS	-42.55	24.81	9.29	-47.75	-185.04	-1033.1	0	19.65
EASY	-44.37	25.67	8.15	-51.32	-189.05	-1095.83	0	19.94
FCFS	13.88	65.1	40.79	16.96	-64.92	-415.96	0	58.75
SOPT(LANL) applied to LANL								
CONS	-8.24	21.59	18.74	-19.81	-38.95	-220.71	0	20.7
EASY	-13.52	19.71	14.83	-23.32	-47.62	-269.36	0	18.21
FCFS	5.47	29.84	28.43	-2.84	-18.07	-164.49	0	29.39
SOPT(SDSC00) applied to SDSC00								
CONS	-180.16	88.85	82.02	63.62	-624.3	-2953.26	4.03	87.1
EASY	-852.06	60.32	41.58	-5.15	-2562.25	-16162.14	4.9	55.21
FCFS	26.27	97.18	95.34	89.58	-94.79	-638.37	-0.8	96.72
SOPT(SDSC95) applied to SDSC95								
CONS	-0.55	8.71	1.31	-19.16	-39.7	-36.43	0	6.48
EASY	-1.01	9.07	0.4	-20.74	-43.34	-40.68	0	6.48
FCFS	2.91	10.54	5.49	-13.04	-31.93	-21.51	0	9
SOPT(SDSC96) applied to SDSC96								

Continued on next page

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
CONS	-1.17	2.14	-2.57	-16.41	-14.22	-7.91	0	0.8
EASY	-1.28	1.96	-2.07	-16.22	-15.12	-13.34	0	0.81
FCFS	0.21	3.62	-1.67	-15.96	-11.09	-3.31	0	2.12

Table K.2: Relative results for all workloads using the Michigan approach in comparison to the standard algorithms.

K.2 Results for Optimizing the Michigan Approach using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	56067.5	49750.49	56705.27	49313.61	58276.98	80794.31	66.99	724325.94
KTH	310640.28	48990.2	81826.42	103892.48	755286.82	6812337.1	71.91	817207.63
LANL	15506.93	9509	11351.81	13881.09	20003.26	79905.11	57.27	140497.29
SDSC00	1537684.1	58757.2	86919.28	141581.83	4035413.5	13200000	78.99	935249.15
SDSC95	50359.6	41606.84	48981.84	65500.27	96219.11	60987.81	62.95	611995.75
SDSC96	48473.24	42947.73	50244.96	94852.21	71478.37	100717.09	61.14	630457.16

Table K.3: Absolute results for all workloads using the Michigan approach with the CTC optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.11	15.28	11.25	2.44	-23.15	-134.23	0	14.06
EASY	-5.42	16.64	12.73	2	-26.36	-153.63	0	15.45
FCFS	0.99	18.98	13.38	7.87	-12.6	-93.22	0	17.31
SOPT(CTC) applied to KHT								
CONS	-321.65	24.66	-1.11	-85.28	-896.99	-16005.44	0	16.1
EASY	-327.05	25.53	-2.38	-89.75	-911.01	-16897.04	0	16.4
FCFS	-154.73	65.03	34	-4.14	-476.85	-7233.63	0	56.92
SOPT(CTC) applied to LANL								
CONS	-18.85	21.79	18.01	-32.45	-63.14	-308.41	0	20.6
EASY	-24.64	19.91	14.06	-36.32	-73.32	-370.37	0	18.11
FCFS	-3.79	30.01	27.79	-13.69	-38.63	-236.82	0	29.31
SOPT(CTC) applied to SDSC00								
CONS	-303.6	88.23	79.67	50.14	-1142.18	-3246.81	5.84	86.05
EASY	-1271.52	58.12	33.94	-44.14	-4465.75	-17725.63	6.69	51.53
FCFS	-6.21	97.03	94.73	85.71	-234.06	-709.36	1.11	96.45
SOPT(CTC) applied to SDSC95								
CONS	-6.74	8.21	-0.3	-22.84	-109.18	-49.55	0	5.65
EASY	-7.23	8.57	-1.22	-24.48	-114.62	-54.22	0	5.65
FCFS	-3.06	10.05	3.95	-16.53	-97.53	-33.2	0	8.19
SOPT(CTC) applied to SDSC96								
CONS	-1.66	4.22	-12.43	-12.42	-47.32	-33.61	0	-0.53
EASY	-1.77	4.05	-11.89	-12.23	-48.48	-40.33	0	-0.52
FCFS	-0.27	5.67	-11.45	-11.98	-43.28	-27.91	0	0.82

Table K.4: Relative results for all workloads using the Michigan approach with the CTC optimized settings in comparison to the standard algorithms.

K.3 Optimization using the Michigan Approach for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	54917.7	50042.74	56752	48100.73	57149.34	70677.04	66.99	727435.42
KTH	104309.26	49493.6	73827.12	81198.33	199620.08	621065.37	71.92	790244.45
LANL	14041.27	9543.32	11278.81	12552.57	16480.21	62142.67	57.27	140548.41
SDSC00	1057520.1	56208.71	77992.19	107241.77	2353008.8	11700000	80.63	874055.91
SDSC95	47513.3	41424.05	48315.96	63459.32	64525.55	55773.13	62.95	607504.33
SDSC96	48346.64	42867.32	50240.32	95984.56	67291.44	95529.59	61.14	629634.42

Table K.5: Absolute results for all workloads using the Michigan approach. The optimization uses all workloads together. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-2.96	14.78	11.18	4.84	-20.77	-104.9	0	13.69
EASY	-3.25	16.15	12.66	4.41	-23.91	-121.87	0	15.09
FCFS	3.02	18.5	13.31	10.14	-10.42	-69.02	0	16.95
SOPT(KTH) applied to KTH								
CONS	-41.58	23.89	8.78	-44.81	-163.5	-1368.3	0	18.87
EASY	-43.4	24.76	7.63	-48.3	-167.21	-1449.58	0	19.16
FCFS	14.46	64.68	40.45	18.61	-52.46	-568.59	0	58.34
SOPT(LANL) applied to LANL								
CONS	-7.61	21.51	18.54	-19.77	-34.4	-217.62	0	20.58
EASY	-12.86	19.62	14.62	-23.28	-42.79	-265.81	0	18.08
FCFS	6.02	29.76	28.25	-2.81	-14.21	-161.95	0	29.28
SOPT(SDSC00) applied to SDSC00								
CONS	-177.57	88.74	81.76	62.23	-624.3	-2872.84	3.88	86.96
EASY	-843.24	59.94	40.73	-9.18	-2562.24	-15733.8	4.75	54.7
FCFS	26.95	97.16	95.27	89.18	-94.79	-618.92	-0.96	96.69
SOPT(SDSC95) applied to SDSC95								
CONS	-0.71	8.61	1.07	-19.01	-40.28	-36.77	0	6.34
EASY	-1.17	8.97	0.15	-20.6	-43.93	-41.03	0	6.34
FCFS	2.76	10.45	5.26	-12.9	-32.47	-21.81	0	8.86
SOPT(SDSC96) applied to SDSC96								
CONS	-1.39	4.4	-12.42	-13.76	-38.7	-26.72	0	-0.4
EASY	-1.51	4.23	-11.88	-13.57	-39.78	-33.1	0	-0.39
FCFS	-0.01	5.85	-11.44	-13.32	-34.89	-21.33	0	0.95

Table K.6: Relative results for all workloads using the Michigan approach in comparison to the standard algorithms. The optimization uses all workloads together.

Appendix L

Results for the Pittsburgh Approach

L.1 Results for Optimizing the Pittsburgh Approach for All Individual Workload Traces

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	56428.44	49639.2	56722.8	49541.76	59212.09	81268.33	66.99	723283.13
KTH	104694.56	49108.6	73183.87	86308.84	223735.78	338484.7	71.92	783821.47
LANL	14124.43	9528.29	11249.62	12635.59	16989.57	62694.38	57.27	140281.4
SDSC00	1168495.7	54967.82	75822.84	110717.73	2947318.9	10700000	79.94	852969.57
SDSC95	47487.81	41389.15	48351.07	63511.55	64362.7	54789.25	62.95	607295.79
SDSC96	48540.72	44152.26	44365.2	101444.88	60730.68	74705.6	61.14	618983.39

Table L.1: Absolute results for all workloads using the Pittsburgh approach. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.79	15.47	11.22	1.98	-25.13	-135.61	0	14.18
EASY	-6.09	16.83	12.7	1.54	-28.39	-155.11	0	15.58
FCFS	0.35	19.16	13.36	7.45	-14.41	-94.35	0	17.42
SOPT(KTH) applied to KTH								
CONS	-42.11	24.48	9.57	-53.92	-195.34	-700.23	0	19.53
EASY	-43.93	25.35	8.44	-57.64	-199.49	-744.53	0	19.82
FCFS	14.15	64.95	40.97	13.49	-70.88	-264.39	0	58.68
SOPT(LANL) applied to LANL								
CONS	-8.25	21.63	18.75	-20.56	-38.56	-220.44	0	20.73
EASY	-13.53	19.75	14.84	-24.09	-47.2	-269.06	0	18.24
FCFS	5.46	29.87	28.44	-3.49	-17.74	-164.27	0	29.42
SOPT(SDSC00) applied to SDSC00								
CONS	-206.7	88.99	82.27	61.01	-807.24	-2615.05	4.71	87.27
EASY	-942.23	60.83	42.37	-12.72	-3234.66	-14360.76	5.57	55.79
FCFS	19.29	97.22	95.4	88.83	-143.98	-556.58	-0.08	96.77
SOPT(SDSC95) applied to SDSC95								
CONS	-0.65	8.69	0.99	-19.11	-39.92	-34.35	0	6.37

Continued on next page

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
EASY	-1.12	9.05	0.08	-20.7	-43.56	-38.55	0	6.37
FCFS	2.81	10.52	5.19	-12.99	-32.13	-19.67	0	8.89
SOPT(SDSC96) applied to SDSC96								
CONS	-1.8	1.53	0.72	-20.23	-25.17	0.9	0	1.3
EASY	-1.91	1.35	1.2	-20.03	-26.15	-4.09	0	1.31
FCFS	-0.41	3.03	1.6	-19.77	-21.74	5.12	0	2.62

Table L.2: Relative results for all workloads using the Pittsburgh approach in comparison to the standard algorithms.

L.2 Results for Optimizing the Pittsburgh Approach using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	56428.44	49639.2	56722.8	49541.76	59212.09	81268.33	66.99	723283.13
KTH	975402.8	69167.43	441969.85	1062178.3	2854982.6	1913462.2	71.7	2459553.73
LANL	15472.68	9504.32	11354.33	13883.95	20192.67	78641.74	57.27	140460.51
SDSC00	2009534	524137.67	1742827	1865873.2	3399389.2	5830000	78.55	12212684.7
SDSC95	48203.6	41557.26	48998.8	65675.8	66855.13	58308.31	62.95	611567.8
SDSC96	48472.03	42947.77	50244.96	94833.5	71477.81	100716.95	61.14	630457.57

Table L.3: Absolute results for all workloads using the Pittsburgh approach with the CTC optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.79	15.47	11.22	1.98	-25.13	-135.61	0	14.18
EASY	-6.09	16.83	12.7	1.54	-28.39	-155.11	0	15.58
FCFS	0.35	19.16	13.36	7.45	-14.41	-94.35	0	17.42
SOPT(CTC) applied to KTH								
CONS	-1223.96	-6.37	-446.11	-1794.24	-3668.64	-4423.73	0.3	-152.52
EASY	-1240.92	-5.15	-452.96	-1840.01	-3721.61	-4674.16	0.3	-151.61
FCFS	-699.85	50.63	-256.49	-964.66	-2080.5	-1959.88	0.3	-29.65
SOPT(CTC) applied to LANL								
CONS	-18.59	21.83	17.99	-32.48	-64.68	-301.95	0	20.63
EASY	-24.37	19.95	14.05	-36.35	-74.96	-362.93	0	18.13
FCFS	-3.56	30.05	27.77	-13.71	-39.94	-231.49	0	29.33
SOPT(CTC) applied to SDSC00								
CONS	-427.44	-4.99	-307.64	-557.15	-946.4	-1381.88	6.37	-82.21
EASY	-1692.38	-273.54	-1224.57	-1799.54	-3746.14	-7792.75	7.21	-532.96
FCFS	-38.8	73.5	-5.71	-88.3	-181.41	-258.36	1.66	53.69
SOPT(CTC) applied to SDSC95								
CONS	-2.17	8.32	-0.33	-23.17	-45.34	-42.98	0	5.71
EASY	-2.64	8.68	-1.26	-24.81	-49.12	-47.44	0	5.71
FCFS	1.35	10.16	3.92	-16.84	-37.25	-27.35	0	8.25
SOPT(CTC) applied to SDSC96								
CONS	-1.66	4.22	-12.43	-12.39	-47.32	-33.61	0	-0.53
EASY	-1.77	4.05	-11.89	-12.21	-48.48	-40.33	0	-0.52

Continued on next page

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
FCFS	-0.27	5.67	-11.45	-11.96	-43.28	-27.91	0	0.82

Table L.4: Relative results for all workloads using the Pittsburgh approach with the CTC optimized settings in comparison to the standard algorithms.

L.3 Optimization using the Pittsburgh Approach for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}
CTC	54903.84	50335.83	57967.01	48886.87	56486.14	65207.88	66.99	735226.35
KTH	106720.02	49919.10	76827.04	83329.44	198581.94	656675.48	71.92	806499.16
LANL	14076.94	9640.11	11561.56	12255.27	16235.06	61187.25	57.27	142647.30
SDSC00	1075764.3	56240.10	77709.24	107033.75	2409309.1	11800000	80.48	873237.96
SDSC95	47730.41	41745.63	49028.22	63608.21	62075.04	55079.82	62.95	613569.14
SDSC96	48252.75	43231.81	48721.36	94841.74	64658.85	93911.70	61.14	627203.48

Table L.5: Absolute results for all workloads using the Pittsburgh approach with the ALLOPT optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
ALLOPT applied to CTC								
CONS	-2.93	14.28	9.27	3.28	-19.37	-89.04	0.00	12.76
EASY	-3.23	15.66	10.79	2.84	-22.48	-104.70	0.00	14.18
FCFS	3.05	18.03	11.46	8.67	-9.14	-55.94	0.00	16.06
ALLOPT applied to KTH								
CONS	-44.86	23.23	5.07	-48.61	-162.13	-1452.48	0.00	17.20
EASY	-46.71	24.11	3.88	-52.20	-165.82	-1538.43	0.00	17.50
FCFS	12.49	64.37	38.03	16.48	-51.67	-606.93	0.00	57.49
ALLOPT applied to LANL								
CONS	-7.89	20.71	16.49	-16.94	-32.41	-212.74	0.00	19.39
EASY	-13.15	18.81	12.48	-20.36	-40.67	-260.18	0.00	16.86
FCFS	5.78	29.05	26.45	-0.37	-12.51	-157.92	0.00	28.23
ALLOPT applied to SDSC00								
CONS	-182.36	88.73	81.82	62.30	-641.63	-2904.31	4.06	86.97
EASY	-859.52	59.92	40.94	-8.97	-2625.94	-15901.45	4.92	54.74
FCFS	25.69	97.16	95.29	89.20	-99.45	-626.53	-0.77	96.69
ALLOPT applied to SDSC95								
CONS	-1.17	7.90	-0.39	-19.29	-34.95	-35.07	0.00	5.40
EASY	-1.63	8.27	-1.32	-20.88	-38.46	-39.28	0.00	5.40
FCFS	2.32	9.75	3.86	-13.17	-27.44	-20.30	0.00	7.95
ALLOPT applied to SDSC96								
CONS	-1.20	3.59	-9.02	-12.40	-33.27	-24.58	0.00	-0.01
EASY	-1.31	3.41	-8.50	-12.22	-34.31	-30.85	0.00	0.00
FCFS	0.18	5.05	-8.07	-11.97	-29.61	-19.27	0.00	1.33

Table L.6: Relative results for all workloads using the Pittsburgh approach with the ALLOPT optimized settings in comparison to the standard algorithms.

L.4 Optimization using the Pittsburgh Approach with a Parallel Rule Reduction using a Modified Fitness Function.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}	N_r
CTC	56067.5	49750.49	56705.27	49313.61	58276.98	80794.31	66.99	724325.94	1
KTH	104443.6	49287.33	73294.73	77453.37	197771.41	711198.66	71.92	786052.27	1
LANL	14123.25	9532.76	11251.01	12556.67	17037.52	62746.78	57.27	140331.68	1
SDSC00	1038476.1	55016.49	75688.93	105801.16	2420830	10700000	80.94	852920.63	23
SDSC95	47548.76	41440.64	48328.86	63698.29	64511.91	55912.5	62.95	607721.85	1
SDSC96	48285.55	42803.15	50095.46	96169.3	67244.83	95123.41	61.14	628413.34	1

Table L.7: Absolute results for all workloads using the Pittsburgh approach with a modified fitness function. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.11	15.28	11.25	2.44	-23.15	-134.23	0	14.06
EASY	-5.42	16.64	12.73	2	-26.36	-153.63	0	15.45
FCFS	0.99	18.98	13.38	7.87	-12.6	-93.22	0	17.31
SOPT(KTH) applied to KTH								
CONS	-41.77	24.21	9.43	-38.13	-161.06	-1581.39	0	19.3
EASY	-43.58	25.07	8.3	-41.46	-164.73	-1674.47	0	19.59
FCFS	14.35	64.82	40.88	22.37	-51.05	-665.62	0	58.56
SOPT(LANL) applied to LANL								
CONS	-8.24	21.59	18.74	-19.81	-38.95	-220.71	0	20.7
EASY	-13.52	19.71	14.83	-23.32	-47.62	-269.36	0	18.21
FCFS	5.47	29.84	28.43	-2.84	-18.07	-164.49	0	29.39
SOPT(SDSC00) applied to SDSC00								
CONS	-172.57	88.98	82.3	62.74	-645.18	-2625.85	3.51	87.27
EASY	-826.26	60.79	42.48	-7.71	-2638.98	-14418.32	4.39	55.79
FCFS	28.27	97.22	95.41	89.32	-100.4	-559.19	-1.34	96.77
SOPT(SDSC95) applied to SDSC95								
CONS	-0.78	8.58	1.04	-19.46	-40.25	-37.11	0	6.31
EASY	-1.25	8.94	0.12	-21.05	-43.9	-41.39	0	6.31
FCFS	2.69	10.41	5.23	-13.33	-32.44	-22.12	0	8.83
SOPT(SDSC96) applied to SDSC96								
CONS	-1.26	4.54	-12.1	-13.98	-38.6	-26.19	0	-0.2
EASY	-1.38	4.37	-11.56	-13.79	-39.68	-32.54	0	-0.19
FCFS	0.11	5.99	-11.11	-13.54	-34.8	-20.81	0	1.14

Table L.8: Relative results for all workloads using the Pittsburgh approach with a modified fitness function in comparison to the standard strategies.

L.5 Optimization using the Pittsburgh Approach with a Parallel Rule Reduction using a Modified Selection Operator.

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}	N_r
CTC	56027.08	49747.9	56646.56	49297.05	58213.22	80780.41	66.99	724065.26	3
KTH	104121.8	49496	73602.53	81098.88	199254.66	622160.67	71.92	789370.14	1

Continued on next page

Workload	AWRT$_{[s]}$	AWRT1$_{[s]}$	AWRT2$_{[s]}$	AWRT3$_{[s]}$	AWRT4$_{[s]}$	AWRT5$_{[s]}$	UTIL$_{[\%]}$	f_{obj}	N_r
LANL	14098.34	9525.9	11230.6	12669.17	16963.31	62295.82	57.27	140181.42	4
SDSC00	1167466.1	54910.14	75939.59	106844.94	2650176.7	12700000	80.14	852859.82	5
SDSC95	47513.8	41425.32	48316.46	63451.78	64526.46	55800.5	62.95	607519.09	1
SDSC96	48285.55	42803.15	50095.46	96169.3	67244.83	95123.41	61.14	628413.34	1

Table L.9: Absolute results for all workloads using the Pittsburgh approach with a modified selection operator. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	AWRT$_{[\%]}$	AWRT1$_{[\%]}$	AWRT2$_{[\%]}$	AWRT3$_{[\%]}$	AWRT4$_{[\%]}$	AWRT5$_{[\%]}$	UTIL$_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.04	15.28	11.34	2.47	-23.02	-134.19	0	14.09
EASY	-5.34	16.64	12.82	2.03	-26.22	-153.58	0	15.48
FCFS	1.06	18.98	13.47	7.9	-12.48	-93.19	0	17.34
SOPT(KTH) applied to KTH								
CONS	-41.33	23.88	9.05	-44.63	-163.02	-1370.89	0	18.96
EASY	-43.14	24.76	7.91	-48.12	-166.72	-1452.31	0	19.25
FCFS	14.62	64.67	40.63	18.71	-52.18	-569.77	0	58.39
SOPT(LANL) applied to LANL								
CONS	-8.05	21.65	18.88	-20.88	-38.34	-218.4	0	20.78
EASY	-13.32	19.77	14.98	-24.42	-46.98	-266.71	0	18.3
FCFS	5.64	29.89	28.56	-3.76	-17.56	-162.59	0	29.47
SOPT(SDSC00) applied to SDSC00								
CONS	-206.42	89	82.24	62.37	-715.77	-3138.28	4.46	87.28
EASY	-941.31	60.87	42.29	-8.77	-2898.47	-17147.58	5.32	55.8
FCFS	19.36	97.22	95.39	89.22	-119.39	-683.11	-0.34	96.77
SOPT(SDSC95) applied to SDSC95								
CONS	-0.71	8.61	1.07	-19	-40.28	-36.83	0	6.34
EASY	-1.17	8.97	0.15	-20.58	-43.93	-41.1	0	6.34
FCFS	2.76	10.45	5.26	-12.89	-32.47	-21.87	0	8.86
SOPT(SDSC96) applied to SDSC96								
CONS	-1.26	4.54	-12.1	-13.98	-38.6	-26.19	0	-0.2
EASY	-1.38	4.37	-11.56	-13.79	-39.68	-32.54	0	-0.19
FCFS	0.11	5.99	-11.11	-13.54	-34.8	-20.81	0	1.14

Table L.10: Relative results for all workloads using the Pittsburgh approach with a modified selection operator in comparison to the standard strategies.

Appendix M

Results for the CCA Approach

M.1 Results for Optimizing the Cooperative Coevolutionary Algorithm Approach for All Individual Workload Traces

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	55290.58	49676.09	56522.7	48723.31	57488.07	74983.13	66.99	722851.67
KTH	93582.95	48433.73	73835.43	76801.6	163626.14	407069.22	71.92	779679.06
LANL	14068.43	9499.16	11274.9	12648.88	16591.92	62397.01	57.27	140091.23
SDSC00	1147231.3	55503.31	74149.63	100895.85	3180365.7	8550000	80.53	851631.65
SDSC95	47515.39	41345.24	48333.02	63677.45	64704.38	56030.24	62.95	606784.45
SDSC96	48500.5	45254.26	42769	98899.38	43133.18	70078.07	61.14	623618.57

Table M.1: Absolute results for all workloads using the CCA. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-3.66	15.4	11.53	3.6	-21.48	-117.38	0	14.23
EASY	-3.96	16.76	13.01	3.17	-24.65	-135.38	0	15.63
FCFS	2.36	19.1	13.66	8.97	-11.08	-79.32	0	17.47
SOPT(KTH) applied to KTH								
CONS	-27.02	25.52	8.77	-36.96	-115.99	-862.38	0	19.95
EASY	-28.65	26.37	7.62	-40.27	-119.03	-915.65	0	20.24
FCFS	23.26	65.43	40.44	23.02	-24.97	-338.22	0	58.9
SOPT(LANL) applied to LANL								
CONS	-7.82	21.87	18.56	-20.69	-35.32	-218.92	0	20.83
EASY	-13.08	20	14.65	-24.22	-43.76	-267.3	0	18.35
FCFS	5.84	30.09	28.28	-3.59	-14.98	-163.02	0	29.51
SOPT(SDSC00) applied to SDSC00								
CONS	-201.11	88.88	82.66	64.47	-878.98	-2073.46	4	87.29
EASY	-923.26	60.44	43.65	-2.72	-3498.33	-11476.18	4.87	55.86
FCFS	20.76	97.19	95.5	89.82	-163.28	-425.61	-0.83	96.77
SOPT(SDSC95) applied to SDSC95								
CONS	-0.71	8.79	1.03	-19.42	-40.67	-37.4	0	6.45
EASY	-1.17	9.14	0.12	-21.01	-44.33	-41.68	0	6.45
FCFS	2.76	10.62	5.22	-13.29	-32.84	-22.38	0	8.97
SOPT(SDSC96) applied to SDSC96								

Continued on next page

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
CONS	-1.72	-0.92	4.3	-17.21	11.1	7.04	0	0.56
EASY	-1.83	-1.11	4.76	-17.02	10.4	2.36	0	0.57
FCFS	-0.33	0.61	5.14	-16.76	13.54	11	0	1.89

Table M.2: Relative results for all workloads using the CCA in comparison to the standard algorithms.

M.2 Results for Optimizing the Cooperative Coevolutionary Algorithm Approach using the CTC Workload Trace and Applying the Resulting Strategy to All Workloads

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	55290.57	49676.09	56522.7	48723.31	57488.07	74983.13	66.99	722851.67
KTH	108292.54	48799.17	79745.46	89594.31	207556.94	491038.29	71.91	806973.51
LANL	14589.25	9516.9	11299.66	13032.94	18724.38	66478.67	57.27	140367.63
SDSC00	1117446.3	55257.35	74705.76	107931.39	2574102.1	11900000	80.58	851396.57
SDSC95	48044.64	41587.63	48807.76	65009.24	66199.8	57974.46	62.95	611107.37
SDSC96	48413.38	42909.22	50161.03	94720.44	71173.48	100368.75	61.14	629736.33

Table M.3: Absolute results for all workloads using the CCA approach with the CTC optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-3.66	15.4	11.53	3.6	-21.48	-117.38	0	14.23
EASY	-3.96	16.76	13.01	3.17	-24.65	-135.38	0	15.63
FCFS	2.36	19.1	13.66	8.97	-11.08	-79.32	0	17.47
SOPT(CTC) applied to KTH								
CONS	-46.99	24.96	1.46	-59.78	-173.98	-1060.89	0	17.15
EASY	-48.87	25.82	0.23	-63.64	-177.83	-1125.16	0	17.45
FCFS	11.2	65.17	35.68	10.2	-58.52	-428.61	0	57.46
SOPT(CTC) applied to LANL								
CONS	-11.81	21.72	18.38	-24.36	-52.71	-239.78	0	20.68
EASY	-17.27	19.85	14.46	-28	-62.24	-291.33	0	18.19
FCFS	2.35	29.96	28.12	-6.74	-29.76	-180.22	0	29.37
SOPT(CTC) applied to SDSC00								
CONS	-193.3	88.93	82.53	61.99	-692.36	-2914.95	3.94	87.3
EASY	-896.69	60.62	43.22	-9.88	-2812.39	-15958.08	4.81	55.87
FCFS	22.82	97.21	95.47	89.11	-113.09	-629.1	-0.89	96.77
SOPT(CTC) applied to SDSC95								
CONS	-1.83	8.25	0.06	-21.92	-43.92	-42.16	0	5.78
EASY	-2.3	8.61	-0.86	-23.54	-47.66	-46.6	0	5.78
FCFS	1.67	10.09	4.29	-15.66	-35.91	-26.62	0	8.32
SOPT(CTC) applied to SDSC96								
CONS	-1.53	4.31	-12.25	-12.26	-46.7	-33.14	0	-0.41
EASY	-1.65	4.13	-11.7	-12.07	-47.84	-39.85	0	-0.4
FCFS	-0.15	5.76	-11.26	-11.83	-42.67	-27.47	0	0.93

Table M.4: Relative results for all workloads using the CCA approach with the CTC optimized settings in comparison to the standard algorithms.

M.3 Optimization using the Cooperative Coevolutionary Algorithm Approach for all Workload Traces Together and Applying the Resulting Strategy to All Traces.

Workload	$AWRT_{[s]}$	$AWRT1_{[s]}$	$AWRT2_{[s]}$	$AWRT3_{[s]}$	$AWRT4_{[s]}$	$AWRT5_{[s]}$	$UTIL_{[\%]}$	f_{obj}
CTC	56052.04	49750.64	56705.38	49265.13	58260.82	80784.89	66.99	724327.88
KTH	104790.82	48774.28	74404.55	81371.64	208309.51	534709.22	71.92	785361.02
LANL	14148.26	9526.29	11256.31	12502.83	17268.45	62815.96	57.27	140288.17
SDSC00	1066988	55878.71	76360.73	103160.35	2371648.5	11900000	80.42	864230.01
SDSC95	47581.74	41379.79	48466.31	63629.56	64776.57	56497.53	62.95	607663.12
SDSC96	48285.38	42802.87	50095.36	96169.30	67247.64	95120.41	61.14	628410.15

Table M.5: Absolute results for all workloads using the CCA approach with the ALLOPT optimized settings. All AWRT values are given in seconds. The UTIL is given in %.

Strategy	$AWRT_{[\%]}$	$AWRT1_{[\%]}$	$AWRT2_{[\%]}$	$AWRT3_{[\%]}$	$AWRT4_{[\%]}$	$AWRT5_{[\%]}$	$UTIL_{[\%]}$	$f_{obj[\%]}$
SOPT(CTC) applied to CTC								
CONS	-5.08	15.28	11.25	2.53	-23.12	-134.20	0.00	14.06
EASY	-5.39	16.64	12.73	2.09	-26.32	-153.60	0.00	15.45
FCFS	1.02	18.98	13.38	7.96	-12.57	-93.20	0.00	17.31
SOPT(CTC) applied to KTH								
CONS	-42.24	24.99	8.06	-45.11	-174.97	-1164.14	0.00	19.37
EASY	-44.06	25.85	6.91	-48.62	-178.84	-1234.12	0.00	19.66
FCFS	14.07	65.19	39.99	18.44	-59.10	-475.63	0.00	58.60
SOPT(CTC) applied to LANL								
CONS	-8.43	21.65	18.70	-19.30	-40.83	-221.06	0.00	20.72
EASY	-13.72	19.77	14.79	-22.79	-49.62	-269.77	0.00	18.23
FCFS	5.30	29.89	28.39	-2.40	-19.67	-164.78	0.00	29.41
SOPT(CTC) applied to SDSC00								
CONS	-180.05	88.81	82.14	63.67	-630.04	-2918.03	4.13	87.11
EASY	-851.69	60.18	41.97	-5.02	-2583.33	-15974.50	4.99	55.21
FCFS	26.30	97.17	95.37	89.59	-96.33	-629.85	-0.69	96.72
SOPT(CTC) applied to SDSC95								
CONS	-0.85	8.71	0.76	-19.33	-40.82	-38.54	0.00	6.32
EASY	-1.32	9.07	-0.16	-20.92	-44.49	-42.87	0.00	6.32
FCFS	2.62	10.54	4.96	-13.20	-32.98	-23.40	0.00	8.84
SOPT(CTC) applied to SDSC96								
CONS	-1.26	4.54	-12.10	-13.98	-38.60	-26.18	0.00	-0.20
EASY	-1.38	4.37	-11.56	-13.79	-39.69	-32.53	0.00	-0.19
FCFS	0.11	5.99	-11.11	-13.54	-34.80	-20.81	0.00	1.14

Table M.6: Relative results for all workloads using the CCA approach with the ALLOPT optimized settings in comparison to the standard algorithms.