Parallele numerische Verfahren zur quantitativen Analyse logistischer Systeme

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften

der Universität Dortmund
am Fachbereich Informatik

von
Markus Fischer

Dortmund
2006

Markus Fischer
Lehrstuhl IV - Modellierung und Simulation
Fachbereich Informatik
Universität Dortmund
44227 Dortmund

Tag der mündlichen Prüfung	24.02.2006
Dekan	Prof. Dr. Bernhard Steffen
Gutachter	Prof. Dr.-Ing. Heinz Beilner, Prof. Dr. Peter Buchholz

Kurzfassung

In der vorliegenden Arbeit wird ein numerisches Verfahren zur Lösung sehr großer Markov-Ketten vorgestellt und seine prinzipielle Eignung und Performance experimentell untersucht. Das Verfahren basiert auf hierarchischen und asynchronen Iterationen, nutzt eine hierarchische Kronecker-Darstellung zur Darstellung der Markov-Kette und ist auf einer parallelen Rechenarchitektur mit verteiltem Speicher implementiert. Das Verfahren kann Markov-Ketten mit 897 Millionen Zuständen in 2-3 Tagen auf einem Cluster mit 7 Workstations (Dualprozessoren) lösen.

Die Effizienz des verteilten, asynchronen Verfahrens im Vergleich zur sequentiellen und synchronen Ausführung liegt zwischen 0.5 und 0.75. Wie aus der Theorie bekannt ist, können asynchrone Iterationen einen numerischen Mehraufwand verursachen, d.h. es sind mehr Schritte bis zum Erreichen einer Genauigkeitsschranke nötig. Dieser Effekt trat in allen Experimenten auf und trug zum Effizienzverlust bei.

Eine Messung der Performance (Lösungszeit) des Verfahrens ist prinzipiell einfach. Um Ursachen für gute und schlechte Performance zu finden, ist es zusätzlich notwendig, einen Einblick in das Laufzeitverhalten des Verfahrens zu erlangen. Das Laufzeitverhalten hat bedingt durch die Asynchronität in Berechnung und Kommunikation einen dynamischen Charakter. Zur Ermittlung des Laufzeitverhaltens wurden Messungen auf der Ebene des Netzwerks und des Programm-Codes vorgenommen, Rohdaten in Logfiles protokolliert und schließlich das Rechen- und Kommunikationsverhalten a-posteriori graphisch aufbereitet und ausgewertet. Dadurch ist es möglich, Kommunikations-, Platz- und Zeitengpässe in ihrer Entstehung zu analysieren und ggf. Hinweise auf bessere Implementierungen zu erhalten. Ansatzpunkte für eine Verbesserung sind eine Steuerung der Kommunikation und alternative Realisierungen der Asynchronität und des Kommunikationspuffers.

Die vorliegende Arbeit ist in einen anwendungsorientierten Sonderforschungsbereich zur "Modellierung großer Netze in der Logistik" eingebettet. Deswegen ist es ein Ziel dieser Arbeit, die Integrierbarkeit und Anwendbarkeit des Markov-Ketten-Instrumentariums im Kontext der quantitativen Analyse logistischer Systeme zu eruieren. Die Anwendbarkeit wird anhand von Analysen einer Stückgut-Umschlaghalle, Lieferketten und Prozessen mit ausfallbehafteten Ressourcen exemplifiziert.

Danksagung

An dieser Stelle möchte ich allen Menschen danken, die durch Ihre Hilfe zum Gelingen dieser Arbeit beitrugen. Mein besonderer Dank gilt meinem Doktorvater Prof. Dr.-Ing. Heinz Beilner für die Schaffung exzellenter Arbeitsbedingungen und seine Unterstützung und Lenkung hin zur Fertigstellung der Dissertation. Des Weiteren danke ich Prof. Dr. Peter Buchholz und Dr. Peter Kemper, die mich bereits während des Informatik-Studiums an die Thematik der Dissertation herangeführt haben und deren Arbeiten das Fundament dieser Dissertation bilden. Ich danke Prof. Dr. Peter Buchholz für die Übernahme des Zweitgutachtens.

Allen Mitarbeitern des Lehrstuhls IV "Modellierung und Simulation" danke ich für die freundliche und kooperative Arbeitsatmosphäre. Besonderer Dank gilt Dipl.-Ing. Zenghui Wu für die zuverlässige Unterstützung bei der Implementierung des Analyseverfahrens und Dipl.-Inf. Jürgen Mäter für die Kooperation mit einer studentischen Projektgruppe.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Für die modellbasierte Analyse von IT-Systemen wurden in der Vergangenheit zahlreiche Modellierungsnotationen und korrespondierende Analyseverfahren entwickelt [20]. Hierbei ist die Interpretation von IT-Systemen als "Diskrete Ereignisgesteuerte Dynamische Systeme" (*DEDS*) [36] essentiell. Etablierte Analyseverfahren wie zum Beispiel analytisch-algebraische Techniken für Warteschlangen-Netze, analytisch-numerische Techniken für stochastische Ketten sowie die Diskrete Ereignisgesteuerte Simulation haben eine gemeinsame Systemsicht: die eines *DEDS*. Die modellbasierte *DEDS*-Analyse ist methodisch so angelegt, dass analytische Modelle wie Warteschlangen-Netze und stochastische Ketten - die als Eingabe der Analyseverfahren dienen - nicht direkt durch einen manuellen Modellierer spezifiziert werden müssen. Es existieren viele höhersprachliche, ggf. an Anwendungen (IT etc.) adaptierte Modellierungsnotationen, aus denen analytische Modelle automatisiert ableitbar sind. Prominente Vertreter sind Petri-Netze für stochastische Ketten und diverse Simulator-Sprachen für die Diskrete Ereignisgesteuerte Simulation.
Mittlerweile sind IT-Systeme oft in wirtschaftliche Prozesse (Fertigung, Transport, Geschäftsprozesse) eingebunden und deswegen als Teilsysteme einer "übergeordneten Welt" zu sehen. Die Entscheider in der betrieblichen Praxis interessiert weniger die Performance involvierter IT-Systeme, als vielmehr die Performance von Geschäfts- und Fertigungsprozessen. Die Notwendigkeit, ausgehend von IT-Systemen Systemgrenzen zu erweitern, macht es erforderlich, die übergeordnete Welt "systemtheoretisch" zu interpretieren. Ein wichtige Beobachtung ist, dass logistische Prozesse und Netzwerke wie auch IT-Systeme als *DEDS* interpretierbar sind, weil auftretende Größen diskret sind (Lagerbestand, Liefermenge) und Ereignisse diskret und asynchron auftreten (Einlagerung, Auftragseingang) [4].
Im Jahr 1998 förderte die "Deutsche Forschungsgemeinschaft" die Gründung des interdisziplinären Sonderforschungsbereichs (SFB) 559 "Modellierung großer Netze der Logistik" [15] an der Universität Dortmund. Zwei Teilprojekte des SFBs untersuchen seitdem, inwieweit das für IT-Systeme entwickelte Portfolio an *DEDS*-Modellierungsnotationen und -Analyseverfahren nutzbringend in der Planung logistischer Systeme (Lieferketten, Geschäftsprozesse, Logistiknetzwerke) einsetzbar ist. Die Herausforderung bei der Planung komplexer logistischer Systeme besteht darin, involvierte Material-, Wert- und Informationsflüsse reibungsfrei und im Sinne individuell festzulegender Performance-Indikatoren optimal zu gestalten [72].
Der SFB 559, insbesondere das Teilprojekt M2 "Effiziente Analyseverfahren", ist der Orientierungsrahmen der vorliegenden Arbeit. Die in der vorliegenden Arbeit verfolgte Systematik zur

Analyse logistischer Systeme ist in die generelle Systematik des SFB 559 eingebettet und in Abb. 1.1 dargestellt.

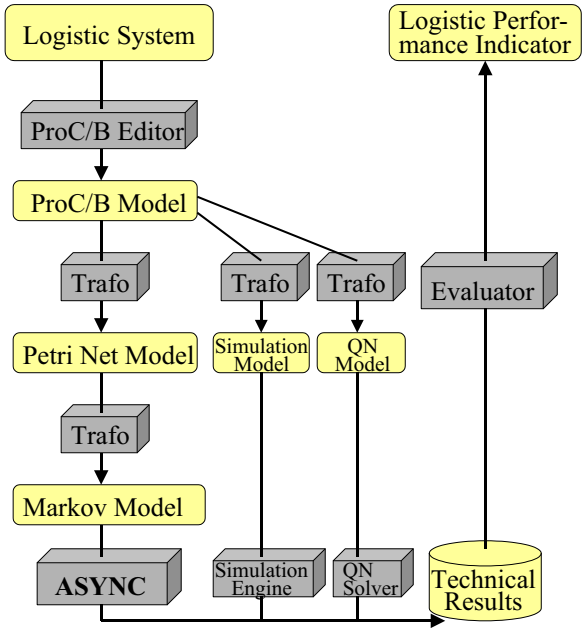

Abbildung 1.1: Systematik zur Analyse logististischer Systeme

Der erste Schritt der Systematik ist, mit der *ProC/B*-Notation [11] eine Schnittstelle zwischen der Informatik- und Logistik-Domäne zu etablieren. Die *ProC/B*-Notation unterstützt eine objekt- und prozess-orientierte sowie hierarchische Sichtweise bei der Modellierung. Die *ProC/B* Notation hat sowohl Konzepte des Prozessketten-Paradigmas von Kuhn [72] adoptiert, wodurch Anforderungen und Bedarfe der Logistik berücksichtigt sind, als auch Konzepte des Modellierungs- und Analyse-Tools HIT von Beilner et al [16], wodurch eine hinreichend formale Beschreibung und eindeutige Interpretation der Modelle als Voraussetzung für automatisierte Übersetzungen in formale analytische Modelle erreicht wird. Damit sind bekannte Technologien der modellbasierten *DEDS*-Analyse via *ProC/B*-Schnittstelle anwendungsfreundlich verfügbar.
Der zweite Schritt der Systematik ist, die Methodik der modellbasierten *DEDS*-Analyse auf ihre Eignung hin zu prüfen und (zum Teil voraussehbare) Defizite zu beheben. Wie in Abb. 1.1 dargestellt, integriert die *ProC/B*-Schnittstelle neben Markov-Ketten-Modellen auch algebraische (Warteschlangen)-Modelle und simulative Modelle [11].
In der vorliegenden Arbeit wird die Anbindung und Anwendung des Markov-Ketten-Instrumentariums unter Zuhilfenahme von Petri-Netzen [13] als interne Zwischennotation betrachtet (linker Strang in Abb. 1.1). Dabei werden Markov-Ketten-Lösungsverfahren experimentell untersucht, die auf asynchronen und hierarchischen, sowie auf verteilt ausgeführten linearen Iterationen beruhen.

1.1 Motivation

Die interdisziplinäre Ausrichtung des SFB 559 "Modellierung großer Netze in der Logistik" spiegelt sich in der Vernetzung von Methoden- und Anwendungsprojekten wider. Der Beitrag der vorliegenden Arbeit gliedert sich ebenso in einen Informatik-Beitrag (Methodik) und in einen Logistik-Beitrag (Anwendung). Der Informatik-Beitrag ist, die Methodik der modellbasierten *DEDS*-Analyse im Bereich der Markov-Ketten-Analyse zu verbessern. Der Logistik-Beitrag ist, die Methodik im Allgemeinen und Speziellen (Markov-Ketten) auf seine Nützlichkeit und Anwendbarkeit im Kontext logistischer Systeme hin zu eruieren. Beide Beiträge werden nachfolgend erläutert.

Logistik-Beitrag Im Management komplexer betrieblicher Systeme hat sich ein prozessorientierter Ansatz etabliert, der Prozesse bzw. Wertschöpfungsketten bereichsübergreifend betrachtet [72, 89]. Solch ein Prozess-Management heißt branchenspezifisch Lieferketten-Management (Supply Chain Management:SCM) oder Geschäftsprozess-Management (Business Process Management:BPM), Logistiknetz-Management [15] etc. Das Prozess-Management hat 4 Funktionen: 1. Prozess-Planung, 2. Prozess-Implementierung (geplante Prozesse in betriebliche Anwendungssoftware überführen), 3. Prozess-Ausführung (operative Unterstützung zur Prozessabwicklung) und 4. Prozess-Controlling (Speicherung, Integration und Aufbereitung entscheidungsrelevanter Daten). Das Ziel von Enterprise Resource Planning (ERP) Tools ist, alle oben aufgezählten Funktionen zu unterstützen. Beispielweise gehören zum SAP-Portfolio für SCM [8] die Stammdatenerfassung im R/3-Basissystem (datenorientierte Modellierung), SAP-APO (Adavanced Planner & Optimizer) für die Planung, SAP-NetWeaver für die Implementierung, SAP-LES (Logistics Execution System) für die Ausführung und SAP-BW (Business Information Warehouse) für das Controlling.
Prozess-Planung beinhaltet die Planung neuer Prozesse oder die Reorganisation bestehender Prozesse unter gegebenen Rahmenbedingungen (existierende, unveränderliche Strukturen, gesetzliche Richtlinien, Investitionsrahmen). Die Vorgehensweise hierbei ist, Prozesse in geeigneter Form zu beschreiben/modellieren, entscheidungsrelevante Performance-Indikatoren zu definieren und qualitative/quantitative Nutzenpotentiale und Engpässe zu identifizieren und zu bewerten. Die Erstellung eines aussagekräftigen und realitätsnahen Prozess-Modells erfordert einen gewissen Aufwand, bietet aber den prinzipiellen Vorteil, teure Messungen an prototypisch implementierten Prozessen oder riskante Eingriffe an im Betrieb befindlichen Prozessen zu vermeiden. So können Prozessabläufe im Modell simuliert und "What-If" Szenarien durch Experimentserien am Modell getestet werden.
Das Modell kann unter zwei Aspekten bewertet werden. Der erste und in der Praxis weit verbreitete Aspekt ist der, dass eine objekt- und prozessorientierte Modellierung Einblicke in betriebliche Abläufe gibt, die es z.B. einem Berater bzw. Analysten ermöglicht, mit Intuition, Erfahrung und Grobabschätzungen Verbesserungsmöglichkeiten zu identifizieren. "Erfahrung" ist oft in praxisgetesteten "Best-Practices" dokumentiert. Für das SCM sind im "Supply Chain Reference Modell" (SCOR) [41] diverse Referenzprozesse beschrieben. Eine menschliche Bewertung des Modells stößt an Grenzen, wenn viele Planungsszenarien zu berücksichtigen sind. Gewöhnlich treten bei der Dimensionierung und Disposition von Ressourcen viele Planungsszenarien auf. Eine menschliche Bewertung stößt ebenfalls an Grenzen, wenn modellierte Prozessabläufe komplex und/oder dynamisch sind. In diesem Fall ist die Beziehung zwischen steuernden Modellparametern und den sie verursachten Wirkungen implizit. Der Ausweg ist eine automati-

sierte (rechnergestützte) Bewertung mit formalen bzw. mathematischen Analysetechniken. Eine Performance Analyse mit formalen Methoden setzt ein ausführbares und hinreichend formales (=eindeutig interpretierbares) Modell voraus.
Ein Beitrag dieser Arbeit ist, dass Leistungs-Indikatoren wie Laufzeiten von Prozessen und Auslastungen eingesetzter Betriebsmittel mit Markov-Ketten ermittelt werden können. Des Weiteren werden Leistungsindikatoren in Kombination mit Zuverlässigkeitsmodellen eingesetzter Betriebsmittel betrachtet.

Informatik Sicht Markov-Ketten sind stochastische Prozesse, die eine spezifische Klasse von *DEDS* beschreiben und im Bereich der quantitativen Performance-Modellierung technischer Systeme weit verbreitet sind [3, 20, 36, 91, 49, 50]. Die stationäre Analyse von Markov-Ketten führt zu einem linearen Gleichungssystem - ein Berechnungsproblem, für das prinzipiell viele erprobte numerische Lösungsverfahren existieren [91]. Logistische Systeme induzieren große Zustandsräume. Ein damit einhergehender hoher Platz- und Rechenaufwand limitiert die Anwendbarkeit des Markov-Ketten-Instrumentariums und erzwingt Weiterentwicklungen sowohl im Bereich mathematischer Modelle numerischer Verfahren als auch bei deren algorithmischer Umsetzung [49, 50]. In dieser Arbeit wird eine algorithmische Umsetzung vorgestellt.
Implementierungen der Lösungsverfahren zielen auf eine effiziente (Platz und Zeit) Ausführbarkeit ab. Lineare Fixpunkt-Iterationen (Jacobi, Gauss-Seidel) mit Erweiterungen sind als numerische Lösungsverfahren für Markov-Ketten praxisbewährt und können mit einer hierarchischen Kronecker-Datenstruktur zur platzeffizienten Speicherung der Markov-Kette angewandt werden [28, 31, 32, 22, 25, 23, 26, 27, 70, 33]. Von großem Interesse ist eine verteilte Implementierung linearer Fixpunkt-Iterationen, die leistungsfähige Rechen- und Platzkapazitäten erschließt. Ein Ergebnis dieser Arbeit ist eine robuste und performante verteilte Implementierung hierarchischer und asynchroner linearer Fixpunkt-Iterationen: *ASYNC*. Die Abkürzung *ASYNC* verweist nachfolgend auf das implementierte Lösungsverfahren und hebt seine besondere Eigenschaft - die Asynchronität in Berechnung und Kommunikation - hervor.
Das theoretische Konvergenzverhalten hierarchischer und asynchroner Erweiterungen linearer Fixpunkt-Iterationen ist in der Mathematik untersucht worden [7, 9, 14, 19, 21, 38, 63, 62, 64, 73, 76, 82, 95, 92, 93, 94]. Es ist zunächst nicht offensichtlich, welche der bekannten theoretischen Ergebnisse im Kontext der stationären Analyse von Markov-Ketten tatsächlich anwendbar sind und es ermöglichen, den Einfluss von Hierarchie und Asynchronität auf die Konvergenzrate zu bewerten. Das Ergebnis der Recherche ist, dass Konvergenz gesichert ist, aber Aussagen zum Einfluss von Hierarchie und Asynchronität auf die Konvergenzrate nicht bekannt sind bzw. sehr schwach sind. Dies rechtfertigt den experimentellen Ansatz dieser Arbeit.
Reale verteilte Implementierungen asynchroner linearer Fixpunkt-Iterationen sind wenig verbreitet und der spezielle Einsatz für die Lösung großer Markov-Ketten mit einer Ausnahme [45] neu. Ein Grund für die geringe Verbreitung könnte der hohe Entwicklungs- und Testaufwand sein. Verteilte asynchrone Systeme erfordern einen hohen Implementierungsaufwand bei der Instanziierung und Verwaltung von (sicheren) Prozessen, bei der Realisation von Kommunikationsroutinen mit asynchroner Semantik und funktional-korrekten Kommunikationsprotokollen sowie beim Testen (Source-Code wird probabilistisch ausgeführt). In bekannten experimentellen Studien über verteilte und asynchrone Iterationen wird die Beschleunigung und Effizienz gemessen, aber Ursachen für gute oder schlechte Performance nicht weiter analysiert. In dieser Arbeit wird der inhärent-probabilistische Charakter asynchroner Berechnungen und Kommunikation

ex-post basierend auf Trace-Informationen analysiert. Eine Visualisierung der zur Laufzeit gesammelten Trace-Informationen gibt Einblick in und das Verständnis für das probabilistische Laufzeitverhalten. Die vorhandene Experimentierumgebung ermöglicht, zwischen verschiedenen *ASYNC*-Implementierungsvarianten zu wählen, um beispielsweise den Grad der Asynchronität einzustellen. Experimente haben gezeigt, dass schwach-asynchrone Iterationen oft die Performance total-asynchroner Iterationen erreichen oder sogar verbessern, wenn die Last gut balanziert ist. Des Weiteren wird in dieser Arbeit für verschiedene Rechen- und Kommunikations-System-Konfigurationen experimentell untersucht, wo *ASYNC* Engpässe induziert. Engpässe im Kommunikationsmedium konnten durch eine Steuerung der Kommunikation behoben werden.

Asynchronität in Berechnung und Kommunikation bietet eine gewisse Flexibilität in der Implementierung, die offen läßt, wer, wann, was berechnet und kommuniziert. Aus "theoretischer Sicht" wird dies als Vorteil angesehen, andererseits stellt sich aus praktischer Sicht die Frage, ob eine vollkommen ungesteuerte Ausführung der Berechnung und Kommunikation sinnvoll ist. Die Asynchronität und Hierarchisierung der Iteration in *ASYNC* bieten zahlreiche Parameter, die das stochastische Laufzeitverhalten und die Konvergenz der numerischen Berechnung beeinflussen und als Steuerparameter zur Erzielung einer verbesserten Konvergenz und Laufzeit dienen können. Allerdings hat es sich als schwierig herausgestellt, den Einfluss zu bewerten. Die Aussagekraft beobachtbarer Wirkzusammenhänge aus einzelnen Experimenten ist gering, weil sie bedingt durch die inhärente Stochastik des Laufzeitverhaltens nicht reproduzierbar und somit nicht allgemeingültig sind.

Eine modellbasierte Performance-Bewertung synchroner und asynchroner Iterationen umfasst die zeitliche Bewertung asynchroner Iterationen im Zusammenspiel mit theoretischen Abschätzungen der Konvergenzrate [19, 35, 84]. In Ergänzung dieser Arbeiten wäre eine genauere systemtheoretische Interpretation hierarchischer und asynchroner Iterationen als Voraussetzung für eine stochastische Performance-Modellierung hierarchischer und asynchroner Iterationen sinnvoll. In dieser Arbeit werden stochastische Modelle für die zeitliche Bewertung hierarchischer und schwach-asynchrone Iterationen vorgestellt und die schwierige analytische Handhabbarkeit diskutiert.

Aufgabenbeschreibung *ASYNC* zielt auf die Beherrschbarkeit großer Markov-Ketten ab. Es soll gezeigt werden, dass *ASYNC* die Performance (=Lösungszeit) und die behandelbare Zustandsraumgröße bekannter numerischer Lösungsverfahren effizient verbessert. Die Performance wird anhand von Messungen evaluiert. Darüber hinaus soll die Möglichkeit einer modellbasierten Performance-Bewertung eruiert werden. Es soll bewertet werden, inwieweit das *ASYNC*-Instrumentarium zur quantitativen Analyse logistischer Systeme anwendbar und nützlich ist und benutzerfreundlich verfügbar gemacht werden kann.

1.2 Organisation

Im Kapitel 2 wird die Klasse der Diskreten Ereignis-gesteuerten Dynamischen Systeme (*DEDS*) definiert und eine Übersicht zu Modellierungsnotationen und Analyseverfahren gegeben. Die Interpretation logistischer Systeme als *DEDS* ist der Ausgangspunkt für ihre modellbasierte quantitative Analyse.
Das Kapitel 3 führt eine spezielle *DEDS*-Klasse - Markov-Ketten - ein. Eine hierarchische, Kronecker-Algebra-basierte Darstellung der Markov-Kette, die als platzeffiziente Datenstruktur zur Speicherung der Markov-Kette dient, wird informal mit Verweisen auf die Originalarbeiten eingeführt. Des Weiteren werden etablierte numerische Lösungsverfahren kurz vorgestellt.
Im Kapitel 4 werden hierarchische und asynchrone Erweiterungen von Fixpunkt-Iterationen anhand mathematischer Modelle definiert und erklärt.
Im Kapitel 5 wird das hier entwickelte numerische Lösungsverfahren zur stationären Analyse von Markov-Ketten - nachfolgend mit *ASYNC* abgekürzt - vorgestellt. Dies beinhaltet: Synergieeffekte zentraler *ASYNC*-Eigenschaften, die Bewertung bekannter Konvergenzeigenschaften hinsichtlich der Anwendbarkeit und Nützlichkeit (Konvergenzraten) im *ASYNC*-Kontext, Besonderheiten und Alleinstellungsmerkmale der *ASYNC*-Implementierung, die Berechnung der Lastverteilung und weitere Implementierungsaspekte wie die Anbindung externer Software und die Realisierung asynchroner Kommunikation.
Das Kapitel 6 fokussiert auf die Messung und Modellierung der *ASYNC*-Performance. In Experimenten wird der Einfluss der Asynchronität auf *ASYNC*-Lösungszeiten und die Effizienz der verteilten Berechnung beobachtet. Das dynamische Laufzeitverhalten von *ASYNC* wird anhand visualisierter Traces ex-post analysiert. Des Weiteren wird eine systemtheoretische Interpretation von *ASYNC* entwickelt, die als Grundlage für eine stochastische Performance-Modellbildung dient.
Im Kapitel 7 wird die Einbindung von *ASYNC* in bestehende Software-Werkzeuge vorgestellt. Die Einbindung unterstützt die benutzerfreundliche Verfügbarmachung und Anbindung an die Logistik-Domäne.
Die Nützlichkeit des *ASYNC*-Instrumentariums und der begleitenden Modellierungs-Software für die Logistik-Domäne wird im Kapitel 8 demonstriert. Hierfür wird das *ASYNC*-Instrumentarium und die Modellierungs-Software aus der Perspektive von Logistik-Anforderungen bewertet und die Performance konkreter logistischer Systeme mit *ASYNC* analysiert.

Kapitel 2

Diskrete ereignis-gesteuerte dynamische Systeme

Die Dynamik vieler technischer Systeme (Computer, Produktion, Logistik) ist durch diskrete Zustände gekennzeichnet, wobei Zustandswechsel über der Zeit durch diskrete und asynchron eintretende Ereignisse (Aktionen) ausgelöst werden [20, 36]. Dynamische Systeme mit diesen Eigenschaften gehören zur Klasse Diskreter Ereignis-gesteuerter Dynamischer Systeme (*DEDS*) [36]. Der Interpretation eines technischen Systems als *DEDS* liegt somit die Annahme zugrunde, dass ein Mechanismus im System Ereignisse/Aktionen generiert. Die Reproduktion bekannter Aspekte dieses Mechanismus in einem Modell, um daraus a priori unbekannte, implizite oder am realen System nicht messbare Eigenschaften zu ermitteln, ist Gegenstand und Ziel einer modellbasierten *DEDS*-Analyse. Eigenschaften beziehen sich auf funktional-logisches Systemverhalten (Auftreten spezifischer Zustandsfolgen, Erreichbarkeit von Zuständen), oder, wenn das Auftreten von Ereignissen explizit zeitbehaftet ist, auf quantitative Systemeigenschaften (Aufenthaltsdauer in Zuständen). Viele Systeme beinhalten stochastisch eintretende Ereignisse, wobei die Zeitdauer bis zum Ereignis-Eintritt stochastisch quantifiziert sein kann. Die modellbasierte *DEDS*-Analyse ist methodisch so angelegt, dass die manuelle Modellbildung durch eine hochsprachliche Notation unterstützt ist. Die Notation muss hinreichend ausdrucksmächtig sein, um den Ereignis-Mechanismus adäquat zu erfassen. Des Weiteren muss die Notation hinreichend formal und eindeutig sein, um die automatisierte Erzeugung eines konsistenten analytischen Modells zu unterstützen. Für *DEDS* sind zahlreiche Modellierungsnotationen und korrespondierende Analyseverfahren bekannt [13, 20, 36, 42, 65, 74].

Der Abschnitt 2.1 führt in die *DEDS*-Modellbildung ein. Hierfür werden zunächst stochastisch-zeitbehaftete Petri-Netze definiert (Abschnitt 2.1.1) und erläutert, wie der unterliegende probabilistische Automat (Abschnitt 2.1.2) und der unterliegende stochastische Prozess (Abschnitt 2.1.3) ableitbar sind. Ansätze zur modellbasierten *DEDS*-Analyse werden im Abschnitt 2.2 rekapituliert.

2.1 *DEDS* Modellbildung

Für die *DEDS* Modellbildung sind zahlreiche höhersprachliche Notationen bekannt, unter ihnen stochastische, zeitbehaftete Petri-Netze (*PN*)[1, 2, 10, 13, 20, 36], Warteschlangen und deren Netzwerke (*QN*) [20, 36], Kombinationen aus *PN*s und *QN*s [10], sowie eine Vielzahl, auf spezielle Sichtweisen und Anforderungen von Systemen angepasste Notationen, wie zum Beispiel Ereignisgesteuerte Prozessketten [89], die *ProC/B* Notation [11, 72] und diverse Simulator-Sprachen [74]. Die Notationen dienen dem Zweck, bekannte Aspekte des *DEDS*-Ereignis-Mechanismus formal oder semi-formal zu beschreiben.
Für die Bewertung hochsprachlicher Notationen können folgende Kriterien dienen: die Notation muss in ihrer Syntax hinreichend mächtig sein, um Eigenschaften des Systems adäquat zu beschreiben; sie muss die Beschreibung des Mechanismus auf ein notwendiges Mindestmaß reduzieren, indem sie zum Beispiel Redundanzen erfasst; sie muss an Denkweisen und spezifischen Anforderungen des jeweiligen Anwendungsgebiets adaptieren, um Akzeptanz zu finden; sie muss Strukturen (Teilsysteme, Hierarchien), Symmetrien und sonstige Informationen erfassen und zugänglich machen, aus denen die Analyse Effizienzsteigerungen erzielen kann; sie muss in ihrer Semantik hinreichend formal und eindeutig interpretierbar sein, um eine automatisierte Analyse zu ermöglichen und sie muss Restriktionen respektieren, die ggf. durch die ausgewählte Analysetechnik vorgegeben sind.
Zustandsraumbasierte Analyseverfahren können gewöhnlich nicht direkt auf hochsprachlichen *DEDS*-Modellen agieren, weil notwendige Zustandsraum-Informationen dort nur implizit verfügbar sind. Deswegen ist es notwendig, aus dem hochsprachlichen Modell ein analytisches Modell abzuleiten, das Zustandsraum-Informationen explizit macht und so als direkte Eingabe für das zustandsraumbasierte Analyseverfahren fungiert. Hierbei spielen probabilistische Zustandsautomaten und stochastische Prozesse eine zentrale Rolle. Nachfolgend werden stochastisch-zeitbehaftete Petri-Netze als eine hochsprachliche Notation detailliert betrachtet, weil sie im Kontext dieser Arbeit bedeutsam sind (vgl. Abschnitt 7.2). Danach wird erläutert, wie aus stochastisch-zeitbehafteten Petri-Netzen probabilistische Zustandsautomaten (Abschnitt 2.1.2) und stochastische Prozesse (Abschnitt 2.1.3) ableitbar sind. Dies schafft den Übergang von *DEDS* zu einer speziellen Klasse stochastischer Prozesse - der Markov-Ketten - die im nachfolgenden Kapitel eingeführt werden.

2.1.1 Stochastische Petri-Netze

Petri-Netze (*PN*) sind ein verbreiteter Formalismus zur Modellierung von *DEDS* [1, 2, 10, 13, 20, 36]. Wesentliche *PN*-Konstrukte sind *Stellen* (symbolisiert durch Kreise), *Transitionen* (Rechtecke) und *Token* (kleine Punkte). Eine Stelle repräsentiert eine diskrete Variable, deren Zustand durch die *Markierung* (=Anzahl der Token) kodiert wird. Eine Stelle modelliert ein atomares Teilsystem. Die Markierungen über alle Stellen repräsentieren Zustände des Gesamtsystems. Ereignisse im *DEDS* werden durch *PN*-Transitionen modelliert, die Markierungen ineinander transformieren und somit Zustandswechsel auslösen. Für jede *PN*-Transition wird eine logische Bedingung auf der Menge der Markierungen definiert, die angibt, wann eine *PN*-Transition *aktiviert* ist. Vorab definierte Regeln priorisieren aktivierte *PN*-Transitionen und wählen eine *PN*-Transition aus (Konfliktauflösung), die *feuert*, d.h. die aktuelle Markierung in eine Nachfolgemarkierung überführt. Die Transformation basiert auf der Addition und/oder Subtraktion von Token auf/von spezifischen Stellen. Es ist vorteilhaft, wenn die Aktivierungsbedingung von

der Markierung weniger Stellen abhängt und der Zustandswechsel durch Modifikation (Addition/Subtraktion) weniger Stellen beschreibbar ist, d.h. wenn im System Voraussetzungen für und Auswirkungen von Ereignissen lokal eingrenzbar sind. Anderenfalls ist die Spezifikation des *PN*s aufwendig und die Lesbarkeit der graphischen *PN*-Darstellung schlecht.
Ursprünglich wurden *PN*s zur funktionalen Analyse (Lebendigkeit, Zustandserreichbarkeitsgraph, Invarianten etc) verteilter Systeme eingesetzt. Erweiterungen der ursprünglichen *PN*-Notation führen zur vereinfachten Spezifikation von Zustandswechseln im *DEDS* (Gewichtsfunktion für Transitionen, Inhibitor-Kanten) und ertüchtigen quantitative Leistungsanalysen durch eine zeitliche Bewertung von *PN*-Transitionen. Die zeitliche Bewertung durch stochastische Verteilungen führt zu stochastischen *PN*s, die mit den zuvor erwähnten Erweiterungen zur Klasse generalisierter, stochastischer *PN*s [1] führen, vgl. Def. 2.1.

Definition 2.1 *Ein **generalisiertes stochastisches Petri-Netz** (GSPN) [1] wird formal durch ein 8-Tupel* $(P, T, \psi, I, O, H, W, M_0)$ *beschrieben. P ist die Menge der Stellen, T ist die Menge der Transitionen,* $\psi : T \to \{0, 1\}$ *ist eine Prioritätsfunktion,* $I, O : T \to \mathbb{N}^{|P|\times|T|}$ *sind Eingabe- und Ausgabefunktion der Transitionen,* $W : T \to \mathbb{R}^{>0}$ *ist eine Gewichtsfunktion für Transitionen und* $M_0 : P \to \mathbb{N}$ *ist eine Funktion zur Festlegung der initialen Markierung der Stellen. Daraus ableitbar ist* $TT = \{t \in T \mid \psi(t) = 0\}$*, die Menge zeitbehafteter Transitionen.*

$W(t)$ erfasst für $t \in TT$ den/die Parameter der stochastischen Verteilung, welche die Lebenszeit der Transition t (Zeit zwischen Aktivierung und feuern) spezifizieren. $W(t)$ ist zum Beispiel die Rate einer Exponentialverteilung. Für zeitlose Transitionen t ($\psi(t) = 1$) ist $W(t)$ ein relatives Gewicht. Zeitlose Transitionen haben stets Priorität gegenüber zeitbehafteten Transitionen.
Die *GSPN*-Erreichbarkeitsmenge $\mathcal{RS}$ ist definiert als die kleinste Menge von Markierungen, für die $M_0 \in \mathcal{RS}$ gilt und die abgeschlossen ist bezüglich aller Transitionen aus T. Es sei $\mathcal{TRS}$ die Menge aller Markierungen, in denen nur zeitbehaftete Transitionen aktiviert sind (tangible reachable set).
Mehrere *GSPN*s können konstruktiv zu einem "superposed *GSPN*" (*SGSPN*) [46, 70] verknüpft werden, vgl. Def. 2.2. Die einzelnen *GSPN*s, nachfolgend als Komponenten des *SGSPN* bezeichnet, repräsentieren Teilsysteme. Die Strukturinformation kann zur Effizienzsteigerung in zustandsraumbasierten Analyseverfahren benutzt werden, vgl. Abschnitt 3.2.

Definition 2.2 *Ein **zusammengesetztes (superposed) GSPN** (SGSPN) [46] wird formal durch ein 9-Tupel* $(P, T, \psi, I, O, H, W, M_0, \phi)$ *beschrieben, wobei* $(P, T, \psi, I, O, H, W, M_0)$ *ein GSPN ist, vgl. Def. 2.1.* $\phi = \{P^1, \dots P^J\}$ *definiert eine Partition von P mit J **Komponenten*** $(P^j, T^j, \psi^j, I^j, O^j, H^j, W^j, M_0^j)$*, wobei* $T^j = {}^\bullet P^j \cup P^j{}^\bullet$ *ist (alle Transitionen die im Vor- oder Nachbereich mit Stellen aus* P^j *verbunden sind) und* $\psi^j, I^j, O^j, H^j, W^j, M_0^j$ *sind die Restriktionen von* ψ, I, O, H, W, M_0 *auf* P^j *bzw.* T^j*. Die Partition* ϕ *klassifiziert die Transitionen wie folgt:* $TL^j = \{t \in T^j \mid {}^\bullet t \cup t^\bullet \subseteq P^j\}$ *ist die Menge **lokaler Transitionen**,* $TS = T \backslash (\cup_{j=1}^{J} TL^j)$ *ist die komplementäre Menge **synchronisierender Transitionen** und* $TT^j = \{t \in T^j \mid \psi(t) = 0\}$ *ist die Menge **zeitbehafteter Transitionen**, die im Vor- und Nachbereich mit Stellen aus* P^j *verbunden sind.*

Nachfolgend wird ein vielzitiertes *PN*-Modell aus dem Bereich der Warteschlangensysteme vorgestellt, das Analogien zur Stückgutumschlaghalle - einem logistischen System - besitzt, vgl. Abschnitt 8.2.1

Beispiel 2.3 (Multi-Server Multi-Queue (MSMQ) Anwendung [2]) .
Ein Multi-Server Multi-Queue (MSMQ) Bediensystem besteht aus $J > 1$ Warteschlangen-Stationen, die von S Bedienern zirkulierend besucht werden, vgl. Abbildung 2.1 links. Jede Station j ($1 \leq j \leq J$) hat einen Puffer mit endlicher Kapazität C_j. Die Last (Kunden etc) wird durch stationsspezifische Poisson-Prozesse mit Rate λ_j generiert. Bei einem Puffer-Überlauf gehen Kunden verloren. Die Kunden haben eine exponentielle Bedienwunschverteilung mit der mittleren Bedienzeit μ_j^{-1}. Der Übergang eines Bedieners von der Station j zur Station $(j \bmod J) + 1$ dauert im Mittel ω_j^{-1} Zeiteinheiten. Wenn bei Ankunft oder nach Beendigung der Bedienung die Warteschlange leer ist, geht der Bediener zur nächsten Warteschlange. Wenn nach der Bedienung die Warteschlange nicht leer ist, geht der Bediener entweder zur nächsten Station oder er bleibt in der aktuellen Station (Modellvarianten). An einer Station können mehrere Bedienvorgänge simultan stattfinden, sofern $S > 1$ ist. Die Abbildung 2.1 rechts zeigt das GSPN-Modell einer einzelnen Station. Das SGSPN-Modell für die gesamte MSMQ Anwendung besteht aus J GSPN-Modellen (=Komponenten) einzelner Stationen, die über zeitbehaftete Transitionen t_Sw_j und t_Sw_j+1 kommunizieren. Die zeitbehafteten Transitionen t_Buf und t_Proc beschreiben die Ankunft und die Bearbeitung von Kunden, die Transitionen t_Sw_j und t_Sw_j+1 beschreiben die Ankunft und den Weggang eines Bedieners an Station j. Die Zuordnung eines verfüfgbaren Bedieners (Token in Stelle p_4) zum wartenden Kunden (Token in Stelle p_2) wird über die zeitlose Transition t_4 modelliert.

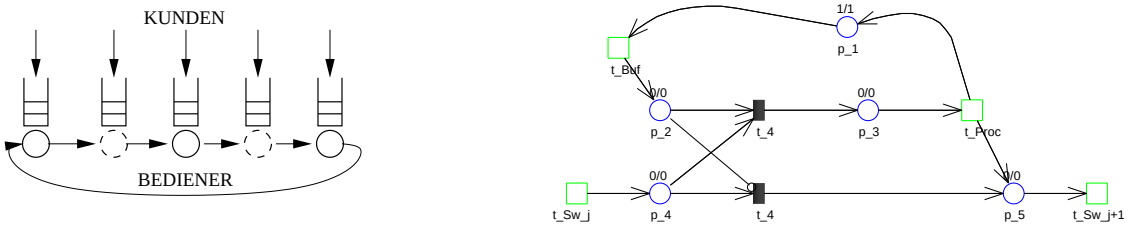

Abbildung 2.1: Links: MSMQ Bedienstation mit $J = 5$ Warteschlangen und $S = 3$ Bedienern; Rechts: *GSPN*-Modell einer einzelnen Bedienstation (Komponente j) mit polling-Bedienung

2.1.2 Probabilistische Automaten

Probabilistisch-zeitbehaftete Automaten sind aus *GSPN*s bzw. *SGSPN*s ableitbar. Zuvor sollen probabilistisch-zeitbehaftete Automaten definiert und in ihrer Wirkungsweise erläutert werden.

Definition 2.4 (Probabilistisch-zeitbehafter Automat [36]) .
Ein stochastisch-zeitbehafteter Automat ist ein 6-Tupel der Form

$$(\mathcal{E}, \mathcal{S}, \Gamma, p, p_0, G),$$

*mit der **Ereignismenge** $\mathcal{E} = \{1, \ldots, m\}$ und dem **Zustandsraum** $\mathcal{S}$. Dabei ist $\Gamma(x)$ die Menge von Ereignissen, die im Zustand $x \in \mathcal{S}$ aktiviert sind und potentiell eintreten. Wenn ein Ereignis*

i im vorgehenden Zustand deaktiviert war und im aktuellen Zustand aktiviert wird, erhält es eine ***Lebenszeit*** y_i *durch die Dichtefunktion* $\{G_i \,|\, i \in \mathcal{E}\}$ *"zugewiesen" (Notation:* $y_i \sim G_i$*). Die Transitions-Funktion* $p : \mathcal{S} \times \mathcal{E} \to \mathcal{S}$ *bestimmt (hier deterministisch) den Nachfolgezustand basierend auf dem aktuellen Zustand und dem eingetretenen Ereignis. Die Funktion* p_0 *bestimmt die Startverteilung des initialen Zustands.*

Die Wirkungsweise des probabilistisch-zeitbehafteten Automaten soll nachfolgend genauer erläutert werden. Die Wirkungsweise entspricht der eines diskreten ereignis-gesteuerten Simulators [36, 74]: Ausgehend vom aktuellen Zustand x wird das nächste Ereignis e' und der Nachfolgezustand x' bestimmt und dabei der Zeitfortschritt protokolliert.

1. Ereignis-Auswahl e':
   Jedes Ereignis $i \in \mathcal{E}$ ist abhängig vom aktuellen Zustand x entweder aktiviert $i \in \Gamma(x)$ oder deaktiviert. Es können nur aktivierte Ereignisse eintreten. Als nächstes Ereignis wird das mit der geringsten Lebenszeit ausgewählt, d.h.

   $$e' = \arg \min_{i \in \Gamma(x)} y_i.$$

   Die Auswahl ist bei zeitlosen Ereignissen nicht eindeutig. In diesem Fall müssen Ereignisse zusätzlich priorisiert werden. Es ist

   $$y^* \triangleq \min_{i \in \Gamma(x)} y_i.$$

   die Zeitdauer zwischen dem zuletzt beobachteten Ereignis und dem nächsten Ereignis.

2. Nachfolgezustand-Auswahl x': $\quad x' = p(x, e')$

3. Lebenszeit-Aktualisierung y_i':
   Wird im Zustand x' ein Ereignis i aktiviert (vorher deaktiviert) oder ist i gerade eingetreten $(i = e')$, so bekommt es "seine" Lebensdauer y_i durch die Verteilung G_i "zugewiesen". Anderenfalls, war i bereits im Zustand x aktiviert, ist aber nicht eingetreten, so wird die Lebenszeit um y^* vermindert, sofern i im Zustand x' aktiviert bleibt. D.h.

   $$\forall i \in \Gamma(x') \;\; y_i' \triangleq \begin{cases} \sim G_i & \text{falls } i = e' \text{ oder } i \notin \Gamma(x) \\ y_i - y^* & \text{falls } i \neq e' \text{ und } i \in \Gamma(x) \end{cases}$$

   mit

   $$y^* \triangleq \min_{i \in \Gamma(x)} y_i.$$

Aus der *GSPN*-Spezifikation $(P, T, \psi, I, O, W, M_0)$ ist die Automat-Spezifikation $(\mathcal{E}, \mathcal{S}, \Gamma, p, p_0, G)$ ableitbar.

- Ereignismenge $\mathcal{E}$: $\mathcal{E}$ ist isomorph zur Menge der Transitionen T.
- Zustandsraum $\mathcal{S}$: Im *GSPN* werden Zustände durch Markierungen kodiert. Die *GSPN*-Erreichbarkeitsmenge $\mathcal{RS}$ ist im *GSPN* implizit spezifiziert und muss durch eine Erreichbarkeitsanalyse ermittelt werden. Der Zustandsraum $\mathcal{S}$ ist isomorph zur Erreichbarkeitsmenge $\mathcal{RS}$.

- Zustandsabhängige Menge aktivierter Ereignisse Γ: Für eine Transition $t \in T$ bestimmt die Eingabe-Inzidenzmatrix $I(t) \in \mathbb{N}^{|P|\times|T|}$, für welche Markierungen/Zustände die Transition t aktiviert ist. Umgekehrt kann somit für jede Markierung/Zustand festgelegt werden, welche Transitionen aktiviert sind.

- Transitionsfunktion p: Für jede Transition $t \in T$ bestimmen die Eingabe- und Ausgabe-Inzidenzmatrizen $I(t)$ bzw. $O(t)$ die Relation zwischen aktueller Markierung (aktueller Zustand) und Nachfolge-Markierung. Die Transitionsfunktion p ist somit aus I und O ableitbar.

- Initiale Zustands-Verteilung p_0: Die initiale Zustands-Verteilung p_0 entspricht der initialen Markierungs-Verteilung M_0.

- Verteilungen für Ereignis-Lebenszeiten G: Im *GSPN* werden die Verteilungen der zeitbehafteten Transitionen durch die Gewichtsfunktion W spezifiziert (oft Rate der Exponentialverteilung). Daraus ist G direkt ableitbar. Zeitlose Transitionen führen zu Ereignissen mit Lebenszeit 0.

2.1.3 Stochastische Prozesse

Stochastische Prozesse sind das analytische Modell (= Eingabe) für zustandsraumbasierte *DEDS*-Analysemethoden.

Definition 2.5 *Ein **stochastischer** Prozess ist eine Familie von Zufallsvariablen*

$$\{X(t) : t \in \mathcal{T}\} \triangleq [X(t_0), X(t_1), \ldots]$$

*in der jede Zufallsvariable $X : \mathcal{S} \to \mathbb{R}$ mit einem Zeitparameter $t \in \mathcal{T}$ indiziert ist. Der Definitionsbereich $\mathcal{S}$ von X ist der **Zustandsraum** des stochastischen Prozesses. Ein **zeit-diskreter** stochastischer Prozess (= stochastische Folge) liegt vor, wenn $\mathcal{T}$ ein diskrete Teilmenge von $\mathbb{N}$ ist. Für $\mathcal{T} = \mathbb{R}_+$ ist der Prozess **zeit-kontinuierlich**. Eine **stochastische Kette** liegt vor, wenn der Zustandsraum $\mathcal{S}$ diskret ist.*

Transitionen im *GSPN* bzw. Ereignisse im probabilistischen Automaten lösen ausgehend vom Startzustand eine stochastische Zustandsfolge (=Trajektorie) aus. Wenn zeitlose Transitionen bzw. Ereignisse eliminierbar sind, ist die Realisierung des *GSPN* bzw. des Automaten ein stochastischer Prozess, genauer ein Generalisierter Semi-Markov-Prozess [36]. "Semi-Markov" bezieht sich auf die Eigenschaft, dass in die Berechnung der Dichtefunktion von $X_{t_{k+1}}$ nur der Zustand x_k und nicht frühere Zustände $x_{k-1}, \ldots, x_0$ zu berücksichtigen sind. "Generalisiert" bezieht sich auf die Eigenschaft, dass die Zeitdifferenz zwischen Ereignissen, also $t_1 - t_0, t_2 - t_1, \ldots$ durch eine beliebige Verteilung (implizit) spezifiziert ist.

2.2 *DEDS* Analyse

Das verbreiteste *DEDS* Analyseverfahren ist die **Diskrete Ereignis-gesteuerte Simulation** [74], weil *DEDS* Modelle ohne Restriktionen einer Simulation zugänglich sind. Der Nachteil

der Simulation liegt in der langen Ausführungszeit für Modelle, bei denen quantitative Kennzahlen mit statistisch abschätzbarer hoher Genauigkeit erforderlich sind. Erschwert wird die simulative Analyse, wenn Ereignisse, die zur gesuchten Kennzahl beitragen, nur niederfrequent auftreten (Zeitskalen-Differenzen), weil dann die Aussagekraft der statistischen Ergebnis-Schätzer sinkt.
Demgegenüber erlauben **analytisch-algebraische Verfahren** [20] eine sehr zeiteffiziente Analyse von *DEDS*-Modellen und sind deswegen sehr gut für Experiment-Reihen parametrisierter Modelle ("what-if" Szenarien) geeignet, wie sie zum Beispiel bei Optimierungsaufgaben auftreten. Allerdings sind *DEDS*-Modelle - um analytisch-algebraisch exakt lösbar zu sein - zahlreichen Restriktionen unterworfen. Approximative Verfahren, die entweder ein approximatives *DEDS*-Modell analytisch-algebraisch exakt oder das originäre *DEDS*-Modell unter Verwendung approximativer Teillösungen lösen, sind übliche Lösungsansätze.
Analytisch-numerische, Zustandsraum-basierte Verfahren [20, 36, 49] beschränken die Restriktionen im *DEDS*-Modell auf das zeitliche Zustandsübergangsverhalten (Markov, Gedächtnislosigkeit) und erlauben die Abbildung von Synchronisationsmechanismen. Die Anwendbarkeit numerischer Lösungsverfahren ist hauptsächlich durch die Größe des Zustandsraums limitiert. Deswegen zielen die Weiterentwicklungen der Verfahren auf die Behandelbarkeit großer Zustandsräume. Im Abschnitt 3.3 werden bekannte Ansätze vorgestellt. Der Nachteil der Verfahren liegt - trotz sehr leistungsfähiger Rechen- und Platzkapazitäten sind - in langen Lösungszeiten.
Simulative, analytisch-algebraische und analytisch-numerische Verfahren können sinnvoll miteinander kombiniert werden, um ihre jeweiligen Vorzüge zu nutzen. Eine Möglichkeit ist die Voranalyse dedizierter Teilmodelle mit numerischen Verfahren für die Spezifikation vereinfachter Ersatzmodelle (Aggregate) und einer anschließenden Analyse des Gesamtsystems (inkl. Aggregat) mit algebraischen Verfahren oder Simulation. Eine andere Möglichkeit ist die Kopplung von analytisch-numerischen Verfahren und Simulation im Bereich der Performability-Modellierung [67]. Beide Ansätze werden im Kontext der Modellierung und Analyse logistischer Systeme in den Abschnitten 8.2.2 und 8.2.3 demonstriert.

Kapitel 3

Markov-Ketten

Sofern die *DEDS*-Modellbildung kein Selbstzweck ist und nicht nur dem Explizieren bekannter Systemeigenschaften, sondern der Ermittlung der im Modell implizit erfassten Performance-Eigenschaften dient, unterliegt die *DEDS*-Modellbildung dem Zielkonflikt zwischen guter analytischer Behandelbarkeit und Beschreibungsmächtigkeit der Modellierungsnotation. Ein Kompromiss (neben anderen) ist, bestimmte - sogenannte Markovsche - Annahmen bezüglich des zeitlichen Verhaltens im *DEDS* zu treffen, die dazu führen, dass das analytische Modell eine Markov-Kette ist. Durch die sogenannte Gedächtnislosigkeit, einer essentiellen Eigenschaft der Markov-Kette, wird die analytische Behandelbarkeit des Modells erheblich erleichtert.

Markov-Ketten sind eine Klasse stochastischer Prozesse, die im Bereich der Modellierung und quantitativen Analyse technischer Systeme weit verbreitet sind [3, 20, 36, 49, 50, 91]. Spezielle Ausprägungen von Markov-Ketten sind (geschlossen) analytisch-algebraisch lösbar, anderenfalls ist eine analytisch-numerische Lösung möglich.

Der Abschnitt 3.1 führt kurz in die Theorie der Markov-Ketten-Modellierung und der stationären Analyse ein. Detaillierter wird darauf eingegangen, wie aus probabilistisch-zeitbehafteten Automaten (zeitkontinuierliche) Markov-Ketten ableitbar sind. Markov-Ketten werden durch sogenannte Generatormatrizen spezifiziert. Für Generatormatrizen ist unter bestimmten Umständen eine Hierarchie und kompositionelle Darstellung generierbar, vgl. Abschnitt 3.2, von der Analyseverfahren profitieren können. Für die stationäre Analyse von Markov-Ketten ist ein großes Portfolio numerischer Verfahren bekannt und erprobt, vgl. Abschnitt 3.3. Im Abschnitt 3.4 wird die Markov-Ketten Modellbildung für zwei Systeme exemplifiziert.

3.1 Grundlagen

Stochastische Ketten (vgl. Def. 2.5) können hinsichtlich statistischer Eigenschaften involvierter Zufallsvariablen klassifiziert werden. Im Zielkonflikt zwischen analytischer Handhabbarkeit und Ausdrucksmächtigkeit involvierter Zufallsvariablen stellen **Markov-Ketten** einen guten Kompromiss dar. Zum Vergleich: In **unabhängigen** stochastischen Ketten sind alle Zufallsvariablen $X(t_0), X(t_1), \ldots$ unabhängig. Diese Annahme ist sehr restriktiv bzgl. des beschreibbaren Systemverhaltens, resultiert aber in einer einfachen Analyse unabhängiger Zufallsvariablen. Das andere Extrem tritt ein, wenn alle Zufallsvariablen $X(t_0), X(t_1), \ldots$ voneinander abhängen, hier kann die Berechnung einer gemeinsamen Verteilungsfunktion über alle Zufallsvariablen extrem kompliziert sein. Demgegenüber hängt in Markov-Ketten die Vorhersage des Nachfolgezustands nur vom aktuellen Zustand ab. Konzeptionell "erfaßt" der aktuelle Zustand alle relevanten Aspekte der gesamten Historie. Deswegen ist die Berechnung der Verteilungsfunktion involvierter Zufallsvariablen gut behandelbar (s.u.). Die Def. 3.1 beinhaltet eine formale Definition für zeitkontinuierliche Markov-Ketten.

Definition 3.1 (Zeitkontinuierliche Markov-Ketten) .
Eine zeitkontinuierliche stochastische Kette $X(t)$ ist eine zeitkontinuierliche Markov-Kette, wenn für die zu beliebigen Zeitpunkten $t_0 \leq t_1 \leq \ldots \leq t_{k+1}$ beobachteten Zustände $x_0, x_1, \ldots, x_{k-1}$
$P[X(t_{k+1}) = x_{k+1} \mid X(t_k) = x_k, X(t_{k-1}) = x_{k-1}, \ldots, X(t_0) = x_0] =$
$P[X(t_{k+1}) = x_{k+1} \mid X(t_k) = x_k]$
gilt, d.h. die stochastische Prognose des Zustands zum Zeitpunkt t_{k+1} ist

P1: unabhängig von der historischen Zustandsfolge $x_1, \ldots, x_{k-1}$,

P2: unabhängig von der Verweildauer im aktuellen Zustand x_k.

*Die Eigenschaften P1 und P2 drücken die Markov-Ketten **Gedächtnislosigkeit** aus.*

Im Abschnitt 2.1.1 und 2.1.2 wurden generalisierte stochastische Petri-Netze (*GSPN*) und probabilistische Automaten als Notationen zur *DEDS*-Modellbildung eingeführt. Wenn zeitlose Transitionen (in *GSPN*) bzw. zeitlose Aktivitäten (in Automat) eliminierbar sind, ist die unterliegende stochastische Kette eine Generalisierte Semi-Markov Kette [36]. Wenn zusätzlich gilt, dass die Verteilung der Lebenszeit der *GSPN*-Transitionen (in *GSPN*) bzw. die Verteilung der Lebenszeit der Ereignisse (in Automat) durch Exponentialverteilungen bestimmt ist, dann ist die einem *GSPN* bzw. Automaten unterliegende stochastische Kette eine Markov-Kette. Dieses Resultat ist wohlbekannt und resultiert aus der sogenannten Gedächtnislosigkeit der Exponentialverteilung. Die Gedächtnislosigkeit bewirkt, dass die Lebenszeit-Verteilung V eines Ereignisses mit Rate λ nach einem beliebigen Alter z (Ereignis ist nach z Zeiteinheiten trotz Aktivierung noch nicht eingetreten, d.h. $V > z$) stets mit der Verteilung der Lebenszeit zu Beginn der Aktivierung übereinstimmt, d.h. $P[V - z \leq t | y > z] = 1 - \exp(-\lambda_i\, t)$. Deswegen kann man bei der Ereignis-Lebenszeit Bestimmung zu jedem beliebigen Zeitpunkt auf die Exponentialverteilungen zurückgreifen. Ein Mitführen des Alters aktivierter Ereignisse, was eine analytische Handhabbarkeit verschlechtert [75], ist nicht notwendig. Deswegen kann die Ereignis-Verteilung und damit auch die Verteilung des Nachfolgezustands leicht bestimmt werden. Für die zustandsabhängige Ereignis-Verteilung E gilt [36]:

$$P[E = i \mid X(t_k) = x_k] = \frac{\lambda_i}{\Lambda(x_k)} \qquad \text{mit} \qquad \Lambda(x_k) \triangleq \sum_{i \in \Gamma(x_k)} \lambda_i \tag{3.1}$$

und

$$P[X(t_{k+1}) = x_{k+1}] = \sum_{\substack{i \in \Gamma(x_k) \\ p(x_k,i)=x_{k+1}}} \frac{\lambda_i}{\Lambda(x_k)}. \tag{3.2}$$

Die Gl. 3.2 zeigt, dass die Prognose des Nachfolgezustands lediglich vom aktuellen Zustand x_k abhängt. Alle anderen Werte in Gl. 3.2 sind entweder aus x_k ableitbar (Menge aktivierter Ereignisse $\Gamma(x_k)$, Gesamtrate aktivierter Ereignisse $\Lambda(x_k)$) und/oder sind expliziter Bestandteil der Automaten/*GSPN*-Spezifikation. Des Weiteren verdeutlicht Gl. 3.2, dass die Prognose unabhängig von der Verweildauer im aktuellen Zustand x_{k+1} und damit auch vom Alter aktivierter Ereignisse ist. Daher sind die Bedingungen P1 und P2 in der Def. 3.1 der Markov-Ketten erfüllt. Die Restriktion der Zeitverbräuche auf Exponentialverteilungen vereinfacht nicht nur die analytische Handhabbarkeit, sondern vereinfacht im Vergleich zu Automaten auch die Spezifikation zeitkontinuierlicher Markov-Ketten.

Definition 3.2 (Homogene, Zeitkontinuierliche Markov-Ketten) .
Eine homogene, zeitkontinuierliche Markov-Kette ist ein 3-Tupel

$$(\mathcal{S}, \mathbf{Q}, p_0)$$

mit diskretem Zustandsraum $\mathcal{S}$, Generatormatrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ (n ist Kardinalität von $\mathcal{S}$) und initialer Zustandsverteilung p_0.

Im Vergleich zum probabilistischen Automaten sind Ereignisse, ihre Lebenszeiten und Ihre Aktivierungsbedingung in Def. 3.2 nicht angegeben. Diese Bestandteile der Automaten-Spezifikation sind in den Einträgen der Generatormatrix $\mathbf{Q}$ verdichtet. In der Beschreibung des Zustandsübergangsverhalten einer Markov-Kette wird von auslösenden Ereignissen abstrahiert (Informationsverlust). Die Einträge der Generatormatrix $\mathbf{Q}$ können wie folgt definiert bzw. interpretiert werden:

Definition 3.3 (Generatormatrix Q) .
Die Einträge der Generatormatrix

$$\mathbf{Q} = \begin{pmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} & \cdots & q_{nn} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

sind durch das Chapman-Kolmogorov-Differentialgleichungssystem

$$\frac{d\,\mathbf{P}(t)}{d\,t} = \mathbf{P}(t) \cdot \mathbf{Q} \text{ mit Transitionsmatrix } \mathbf{P}(t) = \begin{pmatrix} p_{11}(t) & \cdots & p_{1n}(t) \\ \vdots & \ddots & \vdots \\ p_{n1}(t) & \cdots & p_{nn}(t) \end{pmatrix} \tag{3.3}$$

charakterisiert [20, 36, 91]. Die Einträge der Transitionsmatrix sind $p_{ij}(t) \triangleq P[X(s+t) = j | X(s) = i]$ $(\forall s \in \mathbb{R}^{>0})$. Dabei ist $p_{ij}(t)$ die Wahrscheinlichkeit, dass innerhalb von t Zeiteinheiten (unabhängig vom absoluten Zeitpunkt s, weil homogen) ein Zustandsübergang von i nach j stattfindet. Die Randbedingung für $\mathbf{P}(t)$ in $t = 0$ ist:

$$p_{ij}(0) \triangleq \begin{cases} 1 & j = i \\ 0 & j \neq i \text{ (keine Transition in Nullzeit).} \end{cases} \tag{3.4}$$

Die skalare Form der Chapman-Kolmogorov-Systems in Gl. 3.3 und die Bedingung 3.4 ergeben $q_{ij} = \dot{p}_{ij}(0)$, d.h. der Eintrag q_{ij} ist die Rate für $t = 0$, mit der ein Zustandsübergang von i nach j stattfindet bzw. für $i = j$ die Kette im Zustand i verharrt. Eine andere Interpretation basiert auf involvierten Ereignissen. Man kann zeigen [36], dass q_{ij} $(i \neq j)$ die Summe aller Raten von Ereignissen ist, die einen Zustandswechsel von i nach j bewirken. q_{ii} ist die negative Summe aller Raten von Ereignissen, die einen Zustandswechsel von i in einen beliebigen anderen Zustand $\neq i$ bewirken.

Die **transiente Analyse** von Markov-Ketten fokussiert auf die Berechnung der Wahrscheinlichkeitsverteilung $\boldsymbol{\pi}(t) = (\pi_1(t), \ldots, \pi_n(t))$ mit $\pi_i(t) \triangleq P[X(t) = i]$ der Zustände $1, \ldots, n$ zum Zeitpunkt t. Unter Ausnutzung der Transitionsmatrix $\mathbf{P}(t)$ kann $\boldsymbol{\pi}(t)$ wie folgt berechnet werden [20, 36, 91]:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot \mathbf{P}(t) \text{ mit } \boldsymbol{\pi}(0) = p_0.$$

Die Chapman-Kolmogorov Gl. 3.3 liefert eine geschlossene Darstellung von $\mathbf{P}(t)$ gemäß $\mathbf{P}(t) = \exp(\mathbf{Q} \cdot t)$, so dass $\boldsymbol{\pi}(t)$ geschlossen darstellbar durch:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot \exp(\mathbf{Q} \cdot t). \tag{3.5}$$

Allerdings liegt gewöhnlich keine explizite Darstellung des Ausdrucks $\exp(\mathbf{Q} \cdot t)$ in Gl. 3.5 vor. Die Ableitung von Gl. 3.5 nach t ergibt

$$\frac{d\,\boldsymbol{\pi}(t)}{d\,t} = \boldsymbol{\pi}(t) \cdot \mathbf{Q}. \tag{3.6}$$

Die **stationäre Analyse** von Markov-Ketten untersucht das Langzeit-Systemverhalten. Viele Systeme verhalten sich nach einer hinreichend langen Betriebszeit stationär, d.h. die Wahrscheinlichkeit für das Auftreten bestimmter Zustände ist invariant und unabhängig vom Startzustand. Die stationäre Analyse untersucht den Grenzwertprozess

$$\lim_{t \to \infty} \boldsymbol{\pi}(t).$$

Sofern ein Grenzvektor $\boldsymbol{\pi}$ existiert, gilt, dass für $t \to \infty$ die Ableitung $d\,\boldsymbol{\pi}(t)/d\,t$ gegen 0 konvergiert, so dass sich Gl. 3.6 im Grenzwertprozess $t \to \infty$ zu $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ reduziert. Bedingungen für die Existenz und Konsistenz $(\sum_{i=1\ldots n} \pi_i = 1)$ des Grenzvektors $\boldsymbol{\pi}$ und der Berechnungsansatz sind in der Prop. 3.4 zusammengefasst. Die Bedingungen für Existenz und Konsistenz sind anhand von Klassifikationen der Zustände der Markov-Kette formulierbar. Dies betrifft insbesondere die Konzepte der Irreduzibilität, positiver Rekurrenz, Aperiodizität (nur zeitdiskrete Markov-Kette) und Ergodizität (= positive Rekurrenz und Aperiodizität), vgl. Lehrbücher [20, 36, 91].

Proposition 3.4 (Stationäre Verteilung: Existenz, Konsistenz und Berechnung) .
Für eine irreduzible, zeitkontinuierliche Markov-Kette bestehend aus positiv-rekurrenten Zuständen existiert eine eindeutige und konsistente stationäre Wahrscheinlichkeitsverteilung der Zustände $\boldsymbol{\pi}$ mit $\pi_i > 0$ und $\lim_{t\to\infty} \boldsymbol{\pi}(t) = \boldsymbol{\pi}$, die unabhängig vom Initialvektor $\boldsymbol{\pi}(0)$ ist.
Die stationäre Wahrscheinlichkeitsverteilung $\boldsymbol{\pi}$ ist die Lösung des homogenen, linearen Gleichungssystems

$$\mathbf{x} \cdot \mathbf{Q} = \mathbf{0} \tag{3.7}$$

ergänzt durch die Nebenbedingung $\|\mathbf{x}\|_1 = 1$.

Die stationäre Analyse einer zeitkontinuierlichen Markov-Kette erfordert also die Lösung eines wohldefinierten linearen Gleichungssystems. Damit ist die quantitative, stationäre und auf Markov-Ketten basierende Systemanalyse auf ein grundlegendes mathematisches Berechnungsproblem zurückgeführt.
Im Bsp. 3.5 wird das Markov-Ketten-Modell einer Bedienstation mit endlichem Puffer für ankommende Kunden vorgestellt.

Beispiel 3.5 (Markov-Kette einer Bedienstation mit endlichem Puffer) .
Das Ankunftsverhalten der Kunden sei durch einen Poisson-Ankunfsprozess mit der Rate $\lambda > 0$ *erfasst, die Bedienstation habe einen Puffer mit Kapazität 3 (darüber hinaus ankommende Kunden werden abgewiesen) und die Bedienzeit der Kunden sei durch eine Exponentialverteilung mit der Rate* $\mu > 0$ *approximierbar. Dann hat die Generatormatrix der zeitkontinuierlichen Markov-Kette folgendes Aussehen:*

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 & 0 \\ \mu & -(\lambda+\mu) & \lambda & 0 \\ 0 & \mu & -(\lambda+\mu) & \lambda \\ 0 & 0 & \mu & -\mu \end{pmatrix} \tag{3.8}$$

Die spezielle Struktur der Generatormatrix beschreibt einen homogenen, endlichen Geburts- und Todesprozess. Der Lösungsraum $\mathcal{L}$ *des linearen Gleichungssystems* $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ *ist in diesem speziellen Fall geschlossen darstellbar durch*

$$\mathcal{L} = \left\{ r \left(1, \frac{\lambda}{\mu}, \frac{\lambda^2}{\mu^2}, \frac{\lambda^3}{\mu^3} \right) \mid \textit{mit } \lambda, \mu > 0 \textit{ und } r \in \mathbb{R} \right\}.$$

Der normierte Vektor aus $\mathcal{L}$ *mit*

$$r = \begin{cases} \frac{1-\frac{\lambda}{\mu}}{1-(\frac{\lambda}{\mu})^4} & \frac{\lambda}{\mu} \neq 1 \\ 0.25 & \textit{sonst} \end{cases}$$

ist die stationäre Verteilung.

3.2 Hierarchiebildung und Komposition

Die Generatormatrix $\mathbf{Q}$ einer Markov-Kette kann durch eine Blockstruktur ergänzt werden. Das unstrukturierte Gleichungssystem in Gl. 3.7 nimmt dann die Form

$$(\boldsymbol{\pi}_{[1]}, \ldots \boldsymbol{\pi}_{[N]}) \begin{pmatrix} \mathbf{Q}[1,1] & \ldots & \mathbf{Q}[1,N] \\ \vdots & \ddots & \vdots \\ \mathbf{Q}[N,1] & \ldots & \mathbf{Q}[N,N] \end{pmatrix} = \mathbf{0} \text{ und } \sum_{x=1}^{N} \|\boldsymbol{\pi}_{[x]}\|_1 = 1 \tag{3.9}$$

mit Blockgrößen $n(X) \triangleq \dim \mathbf{Q}[X,X]$ an. Eckige Klammern ("[ ]") in Gl. 3.9 verweisen auf bestimmte Teile eines Vektors bzw. auf Blöcke innerhalb der Matrix. In einem Block zusammengefasste Zustände sind per Definition ein *Makrozustand*, im Beispiel der Gl. 3.9 gibt es N solcher Makrozustände. Dadurch entsteht eine zweistufige Hierarchie, die zwischen Intra-Makrozustand

Transitionen (Blöcke auf Hauptdiagonalen) und Inter-Makrozustand Transitionen (Blöcke neben der Hauptdiagonalen) unterscheidet.
Die Strukturierung der Generatormatrix zielt auf Effizienzsteigerungen im numerischen Lösungsverfahren ab. In dieser Arbeit werden die Zeilen und Spalten der Matrix derart permutiert und in Blöcke gruppiert, dass eine platzeffiziente Datenstruktur zur Speicherung der **Q**-Matrix anwendbar ist und verteilte/parallele Rechen- und Platzressourcen erschlossen werden. Der zweite Aspekt wird im Abschnitt 5.5 betrachtet. Nachfolgend wird auf die Anwendung platzeffizienter Datenstrukturen fokussiert. Bei der nachfolgenden Einführung in diese Thematik werden nur die wichtigsten Resultate vorgestellt und mit Verweisen auf die Original-Literatur versehen. Ein konkretes Beispiel für die Anwendung platzeffizienter Datenstrukturen wird im Abschnitt 7.3 vorgestellt.
Gewöhnlich werden Markov-Ketten nicht direkt modelliert, sondern aus einem höhersprachlichen Modell abgeleitet. Die grundlegende Idee hierbei ist, die Struktur in **Q** unter Ausnutzung im höhersprachlichen Modell hinterlegter und/oder durch Voranalysen ermittelter Struktur-Informationen zu erzeugen. In den Ansätzen [23, 31] werden explizite und/oder implizite Struktureigenschaften eines höhersprachlichen Modells (hier *GSPN* bzw. *SGSPN*) für die Hierarchisierung der Markov-Kette verwertet. In [31] werden semi-automatisch hinterlegte Informationen (Regionenkonzept), ergänzt durch die Ergebnisse einer Invariantenanalyse für eine Hierarchiebildung, bereits auf der Ebene des Petri-Netzes benutzt, ein Zustandsraum erzeugt und die zuvor ermittelte Hierarchie übertragen. In [23] wird ausgehend von einem komponierten *GSPN* (= *SGSPN*) eine modulare Kronecker-Darstellung (s.u.) des Zustandsraums erzeugt und dann auf dem erzeugten Zustandsübergangsmodell eine spezifische Erreichbarkeitsanalyse durchgeführt, die zur Äquivalenzklassenbildung bzw. Hierarchisierung auf der Ebene des Zustandsraums benutzt wird. Der Ansatz aus [31] ist weniger aufwendig, weil kein Zustandsraum exploriert werden muss. Allerdings wird - mit Ausnahme einiger, spezieller Klassen von Petri-Netzen - eine Generatormatrix mit nicht erreichbaren Zuständen erzeugt, die in der nachfolgenden numerischen Analyse gesondert behandelt werden müssen. In dieser Arbeit wird der Ansatz aus [23] verwendet.
Technische Systeme sind aus organisatorischen Gründen oft modular (in Teilsysteme) strukturiert. Ein Teilsystem kann sowohl in Interaktion mit seiner Umgebung agieren als auch unabhängig von dieser. Datenstrukturen zur Speicherung der Generatormatrix können platzeffizienter gestaltet werden, wenn man die Eigenschaften der modularen Struktur ausnutzt. Die grundlegende Idee hierbei ist, das Verhalten einzelner Teilsysteme explizit zu beschreiben und daraus eine generische Beschreibung des gesamten Systems abzuleiten. Viele *DEDS*-Notationen ermöglichen es, Strukturinformationen über Teilsysteme zu hinterlegen und schaffen damit die Voraussetzung für eine platzeffiziente Speicherung der Generatormatrix. Beispiele sind Stochastic Automata Networks *SAN*s [91], *SGSPN*s [46] und die *ProC/B* [11, 59] Notation. Plateau hat erkannt, dass die Dynamik von *SAN*s durch eine Generatormatrix erfassbar ist, die generisch durch kleinere Matrizen (erfassen Dynamik einzelner Automaten bzw. synchronisierender Transitionen) verknüpft mit den Operatoren der Kronecker-Algebra $\oplus$ und $\otimes$ darstellbar ist, vgl. [91], Kapitel 9.

Definition 3.6 (Kronecker Operatoren $\otimes$ und $\oplus$ [66]) . *Das Kronecker-Produkt von Ma-*

trizen $\mathbf{A} \in \mathbb{R}^{r_A \times c_A}$ *und* $\mathbf{B} \in \mathbb{R}^{r_B \times c_B}$ *geschrieben als* $\mathbf{A} \otimes \mathbf{B}$ *ist definiert als*

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{1,1} \cdot \mathbf{B} & a_{2,1} \cdot \mathbf{B} & \dots & a_{1,c_A} \cdot \mathbf{B} \\ a_{2,1} \cdot \mathbf{B} & a_{2,2} \cdot \mathbf{B} & \dots & a_{2,c_A} \cdot \mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_A,1} \cdot \mathbf{B} & a_{r_A,2} \cdot \mathbf{B} & \dots & a_{r_A,c_A} \cdot \mathbf{B} \end{pmatrix} \in \mathbb{R}^{r_A \cdot r_B \times c_A \cdot c_B}$$

Die Kronecker-Summe von quadratischen Matrizen basiert auf dem Kronecker-Produkt und der gewöhnlichen Summation von Matrizen. Es ist

$$\mathbf{A} \oplus \mathbf{B} \triangleq \mathbf{A} \otimes \mathbf{I}_{r_B} + \mathbf{I}_{r_A} \otimes \mathbf{B} \in \mathbb{R}^{r_A \cdot r_B \times r_A \cdot r_B}$$

mit $\mathbf{I}_x$ *als Einheitsmatrix der Dimension* x*.*

Dies führt zu **modularen Kronecker-Darstellungen** der Generatormatrix, die oft sehr platzeffizient sind. Beispielsweise wird ein *SAN* bestehend aus zwei unabhängigen Automaten modelliert durch zeitkontinuierliche Markov-Ketten mit Generatormatrix $\mathbf{Q}_1$ bzw. $\mathbf{Q}_2$ durch die Generatormatrix $\mathbf{Q}_1 \oplus \mathbf{Q}_2$ erfasst. Der Umgang mit derartigen Datenstrukturen stellt in zweierlei Hinsicht eine Herausforderung dar: Die kompositionelle Darstellung der Generatormatrix beinhaltet alle potentiellen Zustände des Systems, u.U. auch Zustände, die im Ausgangssystem bedingt durch Synchronisationen zwischen Teilsystemen nicht erreichbar sind. Des Weiteren müssen restriktive Strukturierungsmöglichkeiten (Interaktion zwischen Teilsystemen), die durch die Art der angestrebten Komposition (Kronecker) auferlegt sind, akzeptiert werden.
Buchholz hat aufgezeigt, wie Hierachiebildung in der Markov-Kette (s.o.) die Nutzung einer modularen Kronecker-Darstellung unter Vermeidung nicht-erreichbarer Zustände ermöglicht (Referenzen siehe unten). Eine **hierarchische Kronecker-Darstellung** der Generatormatrix basiert auf einer Zerlegung des Zustandsraums $\mathcal{S}$ in Makrozustände $\mathcal{S}[1], \dots, \mathcal{S}[N]$ mit $\mathcal{S} = \mathcal{S}[1] \dot{\cup} \dots \dot{\cup} \mathcal{S}[N]$. Hier kann die Partition so gewählt werden, dass einzelne Blöcke $\mathbf{Q}[\cdot,\cdot]$ der Generatormatrix durch eine modulare Kronecker-Darstellung erfassbar sind. Eine derartige Zerlegung des Zustandsraums ist nur in wenigen Fällen direkt aus dem hochsprachlichen Modell ableitbar, siehe Anmerkungen zur Hierarchisierung oben. Das algorithmische Vorgehen für *SAN*s ist, die Zustandsräume $\mathcal{S}^1, \dots, \mathcal{S}^J$ involvierter Automaten (Strukturinformation im hochsprachlichen Modell) gemäß einer Äquivalenzrelation (basiert auf Erreichbarkeitskriterium) zu partitionieren (erzeugt lokale Makrozustände), die dann partiell über Kreuzproduktbildung zu (globalen) Makrozuständen "expandieren":

$$\mathcal{S} = \bigcup_{X \in \mathcal{Z}^0} \bigtimes_{j=1}^{J} \mathcal{S}^j[X^j]. \tag{3.10}$$

In Gl. 3.10 ist $\mathcal{Z}^0$ die Indexmenge der Makrozustände, "[ ]" referenziert Partitionen und X^j bildet den Index X eines Makrozustands auf den eines lokalen Makrozustands in Komponente j ab. Es gilt weiterhin $\prod_{j=1}^{J} |\mathcal{S}^j[X^j]| = \dim |\mathcal{S}[X]| \triangleq n(X)$. Die Methodik der hierarchischen Kronecker-Darstellung wurde durch Buchholz und Kemper konzipiert [31, 25, 23, 70, 33] ([33] ist Übersichtsarbeit), implementiert [32] und im Zusammenspiel mit Basisoperationen auf der Datenstruktur [28] und diversen numerischen Analyserverfahren [29, 22, 24, 26, 27] getestet und bewertet.
Plateau hat gezeigt, dass die Kronecker-Darstellung mit *SAN*s verträglich ist, wenn die Automaten über synchronisierte, zeitbehaftete Transitionen kommunizieren. "Synchron" bedeutet hier,

dass mindestens 2 Automaten einvernehmlich eine Zustandstransition auslösen. Erweiterungen der Strukturierungsmöglichkeiten betreffen zeitbehaftete, asynchrone Transitionen und zeitlose Transitionen [47].

3.3 Numerische Verfahren zur stationären Analyse

Numerische Verfahren zur stationären Analyse von Markov-Ketten sind im Bereich der Performance-Modellierung etabliert [91]. Die Herausforderung bei der Lösung von Markov-Ketten-Modellen ist die Größe des Zustandsraums, trotz vieler Fortschritte bei der Weiterentwicklung mathematischer Modelle numerischer Verfahren und deren Umsetzung in algorithmische Modelle [49, 50]. Reale Systeme induzieren große Zustandsräume. Ein damit einhergehender hoher Rechen- und Platzaufwand limitiert die Anwendbarkeit des gesamten Markov-Ketten-Instrumentariums. Algorithmische Modelle, die hohe Rechen- und Platzkapazitäten erschließen (Rechnernetzwerke), sind deswegen sehr interessant.
Mathematische Modelle zur stationären Analyse von Markov-Ketten resultieren aus einem breiten "Fundus" konventioneller Verfahren zur Lösung linearer Gleichungssysteme, die (teilweise) mit problemspezifischen Anpassungen und Erweiterungen ausgestattet sind. Man unterscheidet grob

- direkte Verfahren: *Gauss*-Elimination und LU-Faktorisierung;
- Fixpunkt-Iterationen: Jacobi (JAC), Gauss-Seidel (GS), Jacobi mit Relaxation (JOR) und Gauss-Seidel mit Relaxation (SOR) [18, 98];
- blockorientierte Fixpunkt-Iterationen: $BJAC$, BGS, $BJOR$ und $BSOR$ [18, 91];
- blockorientierte Fixpunkt-Iterationen mit Aggregierung- und Disaggregierungsschritten [22, 26, 44, 77, 78, 88, 90]
- sowie projektive Verfahren (*CGS, GMRES, Arnoldi*) [24, 88, 91].

Direkte Verfahren sind für kleine Zustandsräume sehr gut anwendbar, aber sonst wegen des implizit hohen Platzaufwands inpraktikabel. Projektive Verfahren berechnen sukzessive eine erweiterte Basis des Lösungsraums und generieren aus ihr eine Approximation der Lösung. Die Effizienz hängt von der Anzahl notwendiger Schritte ab (GMRES), weil einzelne Schritte zeitlich teuer sind und die Anzahl der Basisvektoren schrittweise zunimmt und damit auch der Platzaufwand steigt.
Numerische Lösungsverfahren können eine **Blockstruktur**, wie sie in Gl. 3.9 angezeigt ist, auf unterschiedliche Weise nutzen. Blockorientierte Fixpunkt-Iterationen lehnen sich direkt an einer Blockstruktur an. Verteilt ablaufende Verfahren können die Blockstruktur als Problemzerlegung nutzen [29, 55, 45]. Hierarchische Fixpunkt-Iterationen (Abschnitt 4.3) agieren unterschiedlich auf Intra-Block-Ebene und Inter-Block-Ebene und betten in Schritte des iterativen Verfahrens auf Inter-Block-Ebene iterative Verfahren zur Lösung einzelner Blöcke ein (=Intra-Block-Ebene). Fixpunkt-Iterationen mit Aggregierungsschritten alternieren Iterationsschritte im Original-Lösungsraum und in einem niederdimensionalen (aggregierten) Lösungsraum (=Dimension durch Makro-Block-Ebene vorgegeben). Wenn die Blockstruktur eine NCD-Struktur der Generatormatrix [42] widergibt, separieren iterative Verfahren mit Aggregierungsschritten die

Behandlung unterschiedlicher Kopplungsgrade in Generatormatrizen, indem in Ergänzung zu starken Zusammenhangskomponenten auf Intra-Block-Ebene auch schwache Zusammenhangskomponenten auf Inter-Block-Ebene durch Aggregierungsschritte "behandelt" werden. Dadurch wird die Konvergenz zur Lösung beschleunigt (Heuristik).
Umsetzungen mathematischer Modelle in algorithmische Modelle zielen auf eine möglichst effiziente (Platz und Zeit) Ausführbarkeit im Rechner ab. Hierfür sind platzeffiziente Datenstrukturen und die Nutzbarmachung leistungsfähiger Rechen- und Platzkapazitäten bedeutsam. Die hierarchische Kronecker-Darstellung der Generatormatrix (vgl. Abschnitt 3.2) hat hierzu einen substantiellen Beitrag geleistet, weil der Engpass bei der Speicherung der Generatormatrix in vielen Anwendungen aufgehoben wird und eine Blockstruktur (s.o.) erzeugt wird. Die hierarchische Kronecker-Darstellung hat ihre Eignung in Kombination mit (sequentiellen) Fixpunkt-Iterationen [25], verteilt implementierten Fixpunkt-Iterationen [29, 55], projektiven Verfahren [24] und iterativen Aggregierungs-, Disaggregierungsverfahren [22, 26] bewiesen.
Parallele Rechnerarchitekturen bieten hohe Rechen- und Platzkapazitäten. Die parallele Programmierung wird durch vorhandene Bibliotheken für Kommunikationsroutinen (*PVM* [40, 51], *MPI* [40, 48]) erleichtert. Insbesondere projektive Verfahren [17, 71] und auf *BJAC* basierende Fixpunkt-Iterationen [17, 55, 71, 83, 29] sind für verteilte Realisierungen geeignet, weil sie gut skalierbar sind. In den zitierten Arbeiten ist die Generatormatrix als dünbesetzte Matrix im Arbeitsspeicher [71, 83] oder ausgelagert auf externe Medien [17, 71], oder - wie in dieser Arbeit realisiert - mit hierarchischer Kronecker-Darstellung im Arbeitsspeicher [29, 55] gespeichert.
In der vorliegenden Arbeit werden hierarchische und asynchrone Erweiterungen blockorientierter Fixpunkt-Iterationen zur stationären Analyse von Markov-Ketten experimentell untersucht. Insbesondere asynchrone Iterationen sind auf ihre praktische Eignung hin wenig - im Kontext der stationären Analyse von Markov-Ketten noch gar nicht - untersucht worden. Das Grundgerüst bildet eine *BJAC*-Iteration, die asynchron und verteilt auf parallelen Rechenressourcen mit verteiltem Speicher ausgeführt wird. In die Schritte der *BJAC*-Iteration werden Intra-Block-Iterationen eingebettet (hierarchische Iteration). Als Basis-Datenstruktur zur Speicherung der Generatormatrix wird die bereits verfügbare hierarchische Kronecker-Darstellung benutzt.

3.4 Beispiele Markov-Ketten Modellbildung

In diesem Abschnitt werden Zustandsraumgrößen und Blockstrukturen zweier parametrisierter Markov-Ketten vorgestellt, auf die nachfolgend bei der Berechnung von Lastverteilungen und Laufzeitexperimenten mehrfach zurückgegriffen wird.
Das "Flexible Manufacturing System" (`FMS`) von Ciardo und Trivedi [39] wird in der Literatur oft als Referenzsystem für die Bewertung von Markov-Ketten Analyseverfahren verwendet, vgl. [17, 27]. Das `FMS`-Modell beschreibt eine Produktionslinie bestehend aus 3 Maschinen, an denen typisierte Bauteile nach einem festen Schema verarbeitet werden. Die Anzahl der Bauteile im System ist durch die Anzahl der Paletten, die für den Transport der Bauteile zwischen den Maschinen benötigt werden, beschränkt. Nachfolgend sei die Anzahl der Bauteile jedes Typs durch den Modellparameter k festgelegt. Die Tab. 3.1 zeigt die Größe des Zustandsraums n (= Größe Generatormatrix), die Anzahl der Makrozustände N (= Anzahl Hauptdiagonalblöcke in Generatormatrix) und die Anzahl NZ der Makrozustand-Transitionen (= Anzahl Nebendiagonalblöcke $\neq \mathbf{0}$ in Generatormatrix) jeweils in Abhängigkeit von k.
In Stückgut-Umschlaghallen werden ankommende LKWs an mehreren Terminal abgefertigt.

k	n	N	NZ
5	152 712	6	20
10	25 397 658	11	65
15	724 284 864	16	135

Tabelle 3.1: Eigenschaften der `FMS` Markov-Kette

Jedes Terminal hat eine individuelle Anzahl Rampen, die von den LKWs für deren Abfertigung angefahren werden (Rampen beschränken die LKW-Anzahl im Terminal). Sobald ein LKW an einer von ihm belegten Rampe wartet, wird ein Gabelstapler angefordert. Ein Gabelstapler ist entweder bereits am Terminal oder er fährt von einem zentralen Gabelstapler-Pool zum Terminal. Die Anzahl der Gabelstapler k ist ein Modellparameter. Im Abschnitt 8.2.1 wird das System und die Modellbildung eingehend erläutert. Im Vorgriff darauf zeigt die Tab. 3.2 kennzeichnende Größen zweier Modellvarianten in Abhängigkeit von k. Es wird sich später zeigen, dass das `SUH1`-Modell mit $k = 7$ und 1 313 059 781 Zuständen behandelbare Modellgrößen überschreitet, vgl. Abschnitt 6.1.6. Weil das `SUH1`-Modell mit $k = 6$ deutlich kleiner (und behandelbar) ist, wird noch ein zweites Modell `SUH2` betrachtet, in dem das Verhalten der Gabelstapler leicht modifiziert ist und die Zustandsräume im Vergleich zum `SUH1`-Modell etwas kleiner sind. Das `SUH2`-Modell hat 588 759 872 Zustände für $k = 7$.

	`SUH1`			`SUH2`		
k	n	N	NZ	n	N	NZ
4	30 690 750	126	434	19 106 256	126	434
5	122 189 237	252	1008	67 920 112	252	1008
6	423 807 404	462	2058	211 098 912	462	2058
7	1 313 059 781	792	3828	588 759 872	792	3828

Tabelle 3.2: Eigenschaften der `SUH1`- und `SUH2`-Markov-Kette

Kapitel 4

Hierarchische und asynchrone lineare Fixpunkt-Iterationen

In diesem Kapitel werden hierarchische und asynchrone Erweiterungen konventioneller linearer Fixpunkt-Iterationen definiert und bekannte Konvergenzresultate rekapituliert.
Große lineare Gleichungssysteme werden oft mit Fixpunkt-Iterationen numerisch gelöst [98]. Eine Fixpunkt-Iteration berechnet, ausgehend von einer initialen Approximation der Lösung, sukzessive Approximationen bzw. eine Folge von Iterationsvektoren. Eine neue Approximation entsteht dadurch, dass eine lineare Abbildung auf bekannte Approximationen angewandt wird. Fixpunkt-Iterationen unterscheiden sich im Auswahlmechanismus bekannter Approximationen und anzuwendender Iterations-Operatoren, vgl. Abschnitt 4.1. Mathematische Modelle beschreiben die Struktur der Iterations-Operatoren in einer für Konvergenzuntersuchungen zugänglichen Form.
Das Kapitel ist wie folgt strukturiert: Der Abschnitt 4.1 stellt Klassen von Fixpunkt-Iterationen vor und führt in die grundlegenden Konzepte der Konvergenzanalyse ein. Die nachfolgenden Abschnitte 4.2 bis 4.4 betrachten "Stationäre Iterationen", "Nicht-stationäre hierarchische Iterationen" und "Stochastische asynchrone Iterationen".

4.1 Einleitung

Lineare Fixpunkt-Iterationen können zur Lösung linearer Gleichungssysteme verwendet werden. Nachfolgend werden nur homogene Systeme der Form

$$\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$$

mit Koeffizientenmatrix $\mathbf{Q}$ betrachtet. Eine iterative Methode zur Lösung des Systems ist durch lineare Funktionen

$$\mathrm{f}^1(\boldsymbol{\pi}(0) \,|\, \mathbf{Q}),\ \mathrm{f}^2(\boldsymbol{\pi}(0), \boldsymbol{\pi}(1) \,|\, \mathbf{Q}),\ \ldots, \mathrm{f}^t(\boldsymbol{\pi}(0), \ldots, \boldsymbol{\pi}(t-1) \,|\, \mathbf{Q}), \ldots \in \mathbb{R}^{n \times n}$$

definiert, die gemäß dem **Iterationsschema**

$$\boldsymbol{\pi}(t) = \mathrm{f}^t(\boldsymbol{\pi}(0), \ldots, \boldsymbol{\pi}(t-1) \,|\, \mathbf{Q}) \tag{4.1}$$

ausgehend von einer initialen Approximation $\boldsymbol{\pi}(0)$ sukzessive Approximationen $\boldsymbol{\pi}(1), \boldsymbol{\pi}(2), \ldots$ für die Lösungsmenge $\mathcal{L}(\mathbf{Q})$ des Systems $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ generieren. Die Iteration in Gl. 4.1 heißt **stationär**, wenn f^t unabhängig von t ist, anderenfalls nicht-stationär. Im stationären Fall ist $\boldsymbol{\pi}(t) = \mathrm{f}(\boldsymbol{\pi}(0), \ldots, \boldsymbol{\pi}(t-1) \,|\, \mathbf{Q})$. Die Iteration in Gl. 4.1 ist vom **Grad** B, wenn in die Berechnung von $\boldsymbol{\pi}(t)$ maximal die letzten B Approximationen einfließen, d.h. für alle $t - B > 0$ gilt:

$$\boldsymbol{\pi}(t) = \mathrm{f}^t(\boldsymbol{\pi}(t-B), \ldots, \boldsymbol{\pi}(t-1) \,|\, \mathbf{Q}). \tag{4.2}$$

Im Fall nicht-stationärer Iterationen ist gewöhnlich ein **Pool linearer Abbildungen** $\mathcal{F} = \{f^1, \ldots f^M\}$ a priori bestimmbar, aus dem die in der Iteration angewandten linearen Funktionen "entnommen" werden. Die Funktion $j : \mathbb{N} \to [1, \ldots, M]$ "modelliert" die Auswahl, so dass Gl. 4.1 in

$$\boldsymbol{\pi}(t) = \mathrm{f}^{j(t)}(\boldsymbol{\pi}(0), \ldots, \boldsymbol{\pi}(t-1) \,|\, \mathbf{Q}). \tag{4.3}$$

übergeht. Die Gl. 4.3 beschreibt eine **deterministische, nicht-stationäre Iteration**, wenn j eine gewöhnliche nicht konstante deterministische Funktion ist. Die Gl. 4.3 beschreibt eine **stochastische nicht-stationäre Iteration**, wenn j eine Zufallsvariable ist.

Die funktionale Sicht auf die Iteration in den Gleichungen 4.1 bis 4.3 erlaubt, Iterationen hinsichtlich der Auswahl bekannter Approximationen und der Auswahl des Iterations-Operators f^* zu klassifizieren. In Jacobi-Iterationen fließt nur die aktuelle Approximation ein, in Gauss-Seidel-Iterationen fließen bereits aktualisierte Teile und (noch) unveränderte Teile der aktuellen Approximation gemischt ein und in Semi-iterativen Verfahren werden historische Approximationen gezielt zur Verbesserung der Konvergenzgeschwindigkeit gemischt [98]. Asynchrone Iterationen (Abschnitt 4.4) mischen ähnlich wie Semi-iterative Verfahren bekannte Approximationen, allerdings in einer unkontrollierten, stochastischen Weise. Des Weiteren werden in asynchronen Iterationen die Komponenten des Iterationsvektors asynchron aktualisiert.

Die funktionale Sicht der Iteration in den Gleichungen 4.1 bis 4.3 hebt die grundlegende Struktur der Iteration hervor, abstrahiert aber von konkreten Ausprägungen der f^t-Funktionen, die Gegenstand der Konvergenzanalyse sind. Aus der linearen Algebra ist bekannt, dass die Menge möglicher Ausprägungen isomorph zur Menge der Matrizen $= \mathbb{R}^{n \times n}$ ist. Der Iterations-Operator kann explizit durch einen algebraischen Ausdruck angegeben werden, der das "**mathematische Modell**" für die Konvergenzanalyse darstellt. Mathematische Modelle resultieren aus einem **Splitting** der Koeffizientenmatrix $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ mit nicht-singulärer Matrix $\mathbf{M}$, das das System $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ mit algebraischen Umformungen in eine Fixpunktgleichung $\mathbf{x} = \mathbf{x} \cdot \mathbf{N} \cdot \mathbf{M}^{-1}$ überführt. Aus der Fixpunktgleichung ist die Iterationsvorschrift

$$\begin{aligned} \boldsymbol{\pi}(t+1) &= \boldsymbol{\pi}(t) \cdot \underbrace{\mathbf{N} \cdot \mathbf{M}^{-1}}_{\mathbf{T}} \\ &= \boldsymbol{\pi}(t) - \underbrace{\boldsymbol{\pi}(t) \cdot \mathbf{Q}}_{\mathbf{r}(t)} \cdot \mathbf{M}^{-1} \end{aligned}$$

mit **Iterationsvektor** $\boldsymbol{\pi}(t)$, **Iterations-Operator** $\mathbf{T} \triangleq \mathbf{N}\mathbf{M}^{-1}$, und **Residualvektor** $\mathbf{r}(t)$ direkt ablesbar.

Nachfolgend werden Begrifflichkeiten der Konvergenzanalyse definiert, die in den anschließenden Abschnitten bedeutsam sind.

Definition 4.1 (Grundlegende Definitionen zur Konvergenzanalyse) .

a) *Es sei* $\sigma(\mathbf{T}) \triangleq \{\lambda_1, \ldots, \lambda_n\}$ *das* ***Spektrum*** *der* ***Eigenwerte*** *von* $\mathbf{T} \in \mathbb{R}^{n \times n}$ *mit o.B.d.A* $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$ *und zugehörigen Eigenvektoren* $\mathbf{v}_1, \ldots, \mathbf{v}_n$ *(*$\mathbf{v}_i \cdot \mathbf{T} = \lambda_i \cdot \mathbf{v}_i$ *für* $i = 1, \ldots, n$*),* $\rho(\mathbf{T}) = \max\{|\lambda| \,|\, \lambda \in \sigma(\mathbf{T})\}$ *ist der* ***Spektralradius*** *von* $\mathbf{T}$*, d.h. der betragmäßig größte Eigenwert von* $\mathbf{T}$*. Des Weiteren sei* $\gamma(\mathbf{T}) = \max\{|\lambda| \,|\, \lambda \in \sigma(\mathbf{T}),\ |\lambda| \neq 1\}$ *die* ***Magnitude*** *von* $\mathbf{T}$*. d.h. der betragmäßig größte Eigenwert von* $\mathbf{T}$ *ungleich 1.*

b) *Eine Matrix* $\mathbf{T}$ *heißt* ***konvergent*** *bzw.* ***0-konvergent****, wenn der Grenzwert* $\lim_{t\to\infty} \mathbf{T}^t$ *existiert bzw. gleich 0 ist. Resultiert* $\mathbf{T}$ *aus einem Matrix-Splitting, dann spricht man von einem* ***konvergenten*** *bzw.* ***0-konvergenten Splitting****.*

c) *Eine Matrix* $\mathbf{A}$ *heißt* ***M-Matrix****, wenn sie in der Form* $\mathbf{A} = s\mathbf{I} - \mathbf{P}$ *mit* $s \geq \rho(\mathbf{P}) > 0$ *und* $\mathbf{P} \geq \mathbf{0}$ *ausgedrückt werden kann.*

d) *Eine M-Matrix* $\mathbf{A}$ *mit* $\mathbf{A} = s\mathbf{I} - \mathbf{P}$ *hat die* ***Eigenschaft C****, wenn* $s^{-1}\mathbf{P}$ *konvergent ist (weitere Charakterisierungen in* [18, 85] [1] *).*

e) *Ein* ***Splitting*** $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ *heißt* ***[schwach] regulär****, wenn* $\mathbf{M}^{-1} \geq \mathbf{0}$ *und* $\mathbf{N} \geq \mathbf{0}$ *[*$\mathbf{N}\mathbf{M}^{-1} \geq \mathbf{0}$*] gilt.*

Die **Konvergenzrate** ist ein Maß für die Geschwindigkeit, mit der sich der Abstand der Approximationen $\boldsymbol{\pi}(t)$ von der exakten Lösung $\boldsymbol{\pi}$ verringert. Der Abstand als Vektor wird mit **Vektornormen** $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ skalar quantifiziert. Für $\mathbf{x} \in \mathbb{R}^n$ und einen positiven Vektor $\boldsymbol{\pi} \in \mathbb{R}^n > \mathbf{0}$ (elementweise) ist die Summen-Norm $\|\mathbf{x}\|_1$, die Maximum-Norm $\|\mathbf{x}\|_\infty$ und die projektive Parallelepiped-Norm $\|\mathbf{x}\|_{\boldsymbol{\pi}}$ definiert durch

$$\|\mathbf{x}\|_1 \triangleq \sum_{i=1}^{n} |\mathbf{x}_i| \tag{4.4}$$

$$\|\mathbf{x}\|_\infty \triangleq \max_{i=1\ldots n} |\mathbf{x}_i| \tag{4.5}$$

$$\|\mathbf{x}\|_{\boldsymbol{\pi}} \triangleq \max_{i=1\ldots n} \frac{|\mathbf{x}_i|}{\boldsymbol{\pi}_i} = \min\{\delta > 0 \,|\, -\delta\boldsymbol{\pi} \leq \mathbf{x} \leq \delta\boldsymbol{\pi}\}. \tag{4.6}$$

Eine **Distanzfunktion** $\mathrm{d} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ bewertet den Abstand zwischen Vektoren basierend auf Vektornormen, z.B. mit der Maximum-Norm oder der projektiven Parallelepiped-Norm

$$\mathrm{d}(\mathbf{x}, \boldsymbol{\pi}) \triangleq \begin{cases} \|\mathbf{x} - \boldsymbol{\pi}\|_\infty = \max\limits_{i=1\ldots n} |\mathbf{x}_i - \boldsymbol{\pi}_i| \\ \|\mathbf{x}\|_{\boldsymbol{\pi}} = \max\limits_{i=1\ldots n} \frac{|\mathbf{x}_i|}{\boldsymbol{\pi}_i}. \end{cases} \tag{4.7}$$

Die Distanzfunktion kann auch für kompakte Mengen Π von Vektoren definiert werden. Hier sei

$$\mathrm{d}(\mathbf{x}, \Pi) \triangleq \inf_{\boldsymbol{\pi} \in \Pi} \mathrm{d}(\mathbf{x}, \boldsymbol{\pi}). \tag{4.8}$$

In Iterationen wird der Abstand des aktuellen Iterationsvektors $\boldsymbol{\pi}(t)$ von Lösungen aus $\mathcal{L}(\mathbf{Q})$ durch $\mathrm{d}(\boldsymbol{\pi}(t), \mathcal{L}(\mathbf{Q}))$ bewertet. Eine in der Praxis oft verwendete Distanzfunktion ist die Maximum-Norm des t-ten **Fehlervektors** $\|\boldsymbol{\epsilon}(t)\|_\infty$ oder des t-ten normierten Fehlervektors $\|\bar{\boldsymbol{\epsilon}}(t)\|_\infty$:

[1] [18]: Lemma 6.4.11, Theorem 6.4.12 und Lemma 7.6.18 (singular) und [85]: Lemma 1

$$\boldsymbol{\epsilon}(t) \triangleq \boldsymbol{\pi}(t) - \boldsymbol{\pi} \tag{4.9}$$

$$\bar{\boldsymbol{\epsilon}}(t) \triangleq \bar{\boldsymbol{\pi}}(t) - \boldsymbol{\pi} \quad \text{mit } \bar{\boldsymbol{\pi}}(t) \triangleq \frac{1}{\|\boldsymbol{\pi}(t)\|_1}\boldsymbol{\pi}(t). \tag{4.10}$$

Dabei ist $\boldsymbol{\pi}$ ein Element aus $\mathcal{L}(\mathbf{Q})$. Für $\boldsymbol{\epsilon}(t)$ gilt

$$\boldsymbol{\epsilon}(t) \triangleq \boldsymbol{\pi}(t) - \boldsymbol{\pi} = \boldsymbol{\epsilon}(t-1) \cdot \mathbf{T} = \ldots = \boldsymbol{\epsilon}(0) \cdot \mathbf{T}^t.$$

und für eine beliebige Norm $\|\cdot\|$ ist $\|\boldsymbol{\epsilon}(t)\| = \|\boldsymbol{\epsilon}(0) \cdot \mathbf{T}^t\| \leq \|\boldsymbol{\epsilon}(0)\| \cdot \|\mathbf{T}^t\|$. Die **mittlere theoretische Konvergenzrate** ist definiert durch [98]

$$R_t(\mathbf{T}) \triangleq -t^{-1} \cdot \log \|\mathbf{T}^t\|. \tag{4.11}$$

In praktischen Situationen ist $R_t(\mathbf{T})$ oft nicht bekannt und man geht zur **asymptotischen theoretischen Konvergenzrate** definiert durch [98]

$$R_\infty(\mathbf{T}) \triangleq \lim_{t\to\infty} R_t(\mathbf{T}) = -\log \lim_{t\to\infty} (\|\mathbf{T}^t\|)^{1/t} = \begin{cases} -\log \rho(\mathbf{T}) & \text{falls } \rho(\mathbf{T}) < 1 \\ -\log \gamma(\mathbf{T}) & \text{falls } \rho(\mathbf{T}) = 1 \end{cases} \tag{4.12}$$

über. Die Anzahl der Iterationen, die notwendig ist, um den Fehler $\boldsymbol{\epsilon}(t)$ relativ zum Startfehler $\boldsymbol{\epsilon}(0)$ um einen Faktor ϵ zu vermindern, ist wegen $\boldsymbol{\epsilon}(t) = \boldsymbol{\epsilon}(0) \cdot \mathbf{T}^t$ durch

$$t > -\frac{\log \epsilon}{R_t(\mathbf{T})} \approx -\frac{\log \epsilon}{R_\infty(\mathbf{T})} \text{ mit } \epsilon = \frac{\|\boldsymbol{\epsilon}(t)\|}{\|\boldsymbol{\epsilon}(0)\|}$$

abschätzbar.

Zur Charakterisierung des Konvergenzverhaltens einer Folge von Iterationsvektoren $\boldsymbol{\pi}(0), \boldsymbol{\pi}(1), \ldots$ werden oft sogenannte **Folgen geschachtelter Level-Mengen** $\{X(k) : X(k) \subset \mathbb{R}^n\}$ mit $X(k+1) \subset X(k)$ für $k \geq 0$ betrachtet. Hierbei ist $X(k)$ eine Menge von Punkten, die bzgl. einer Distanzfunktion einen durch k parametrisierten Abstand unterschreiten. Beispielsweise beschreiben für $\rho < 1$ und $C > 0$ die Mengen $X(k) \triangleq \{\mathbf{x} : d(\mathbf{x}, \boldsymbol{\pi}) \leq C \cdot \rho^k\}$ für $k = 0, 1, \ldots$ eine Folge geschachtelter Level-Mengen bzgl. einer Lösung $\boldsymbol{\pi}$ und einer Distanzfunktion d. Des Weiteren sei $\phi : \mathbb{N} \to \mathbb{N}$ eine schwach steigende Funktion ($\phi(k+1) \geq \phi(k)$ für $k \geq 0$) mit $\lim_{k\to\infty} \phi(k) = \infty$, die aus einer beliebigen Folge von Iterationsvektoren $\boldsymbol{\pi}(0), \boldsymbol{\pi}(1), \ldots$ eine Teilfolge derart konstruiert, so dass für geschachtelte Level-Mengen $X(0), X(1), \ldots$

$$\forall t \geq \phi(k) \,:\, \boldsymbol{\pi}(t) \in X(k) \tag{4.13}$$

gilt. Konstruktive Konvergenzbeweise werden (grob) nach folgendendem Schema geführt: zuerst wird eine geschachtelte Folge von Level-Mengen und danach eine Teilfolge gemäß Bedingung 4.13 konstruiert. In den Level-Mengen werden Vektoren mit aufsteigender Approximationsgüte gruppiert. Die Teilfolge markiert untere Schranken für die Anzahl von Iterationsschritten, nach denen der Iterationsprozess einen "messbaren" Fortschritt hin zur Lösung erzielt, also von einer Level-Menge $X(k)$ in die "bessere" Level-Menge $X(k+1)$ übergeht. Die theoretische Konvergenzrate hängt sowohl von der Konstruktion der Level-Mengen ab (wie "groß" ist der Fortschritt bei Übergang von $X(k)$ nach $X(k+1)$) als auch vom Anstieg der durch ϕ beschriebenen Teilfolge.

4.2 Stationäre Iterationen

Ein Splitting $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ mit nicht-singulärer Matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ überführt das System $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ in eine Fixpunktgleichung bzw. Fixpunkt-Iteration $\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t) \cdot \mathbf{T}$. Das Ziel der Konvergenzanalyse ist es, (möglichst schwache) Bedingungen für die Koeffizientenmatrix $\mathbf{Q}$ und das Splitting zu finden, die hinreichend (und notwendig) für Konvergenz sind.
Für jede Koeffizientenmatrix $\mathbf{Q}$, die eine nicht-singuläre M-Matrix ist, existiert ein (schwach) reguläres Splitting und alle (schwach) regulären Splittings sind stets 0-konvergent [18][2]. Der Banachsche Fixpunktsatz besagt, dass die 0-Konvergenz (Kontraktion) des Iterationsoperators hinreichend, im Fall nicht-singulärer Systeme auch notwendig für Konvergenz der Iteration ist [18, 43][3]. Dabei wird 0-Konvergenz einer Matrix $\mathbf{T}$ oft durch die äquivalente Aussage $\rho(\mathbf{T}) < 1$ ersetzt, siehe [43][4].
Für den Fall eines **singulären Systems $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$** sind die Zusammenhänge komplizierter. Eine wichtige Beobachtung ist, dass jedes Splitting zu einem Iterationsoperator $\mathbf{T}$ mit einem Spektralradius 1 führt [79, 80, 94], vgl. Lemma 4.2.

Lemma 4.2 (Spektralradius der Iterationsmatrix für singuläres System) .
***Voraussetzung**: Es sei $\mathbf{T} = \mathbf{N} \cdot \mathbf{M}^{-1}$ der Iterationsoperator resultierend aus dem Splitting $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ ($\mathbf{M}^{-1}$ existiert) des singulären Systems $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$.*
***Behauptung**: $\mathbf{Q}$ singulär $\Leftrightarrow \rho(\mathbf{T}) = 1$.*
***Beweis**: "$\Rightarrow$": Nach Voraussetzung existiert ein Vektor $\mathbf{v} \neq \mathbf{0}$ mit $\mathbf{v} \cdot \mathbf{Q} = \mathbf{0}$. Mit $\mathbf{Q} = (\mathbf{I} - \mathbf{T}) \cdot \mathbf{M}$ und der Existenz von $\mathbf{M}^{-1}$ ist dieser Ausdruck algebraisch äquivalent zu $\mathbf{v} \cdot \mathbf{T} = \mathbf{v}$.*
"$\Leftarrow$": Es ist $\mathbf{v} \cdot \mathbf{Q} = \mathbf{v} \cdot (\mathbf{I} - \mathbf{T}) \cdot \mathbf{M} = \mathbf{v} \cdot \mathbf{M} - \underbrace{\mathbf{v} \cdot \mathbf{T}}_{=\mathbf{v}} \cdot \mathbf{M} = \mathbf{0}$ für ein $\mathbf{v} \neq \mathbf{0}$ und damit ist $\mathbf{Q}$ singulär.

Dies gilt insbesondere für Splittings von asynchronen und hierarchischen Iterationen [94]. Iterationsoperatoren mit dieser Eigenschaft sind niemals 0-konvergent (kontrahierend), sondern nur konvergent oder divergent. Das Theorem 4.3 beinhaltet bekannte Bedingungen, die notwendig und hinreichend für die Konvergenz eines Iterationsoperators sind. Für die Formulierung der Bedingung 4.3 c) sind mathematische Begrifflichkeiten erforderlich, die dem Bereich der Triagonalisierbarkeit linearer Operatoren bzw. der dafür im charakteristischen Polynom enthaltenen Strukturinformationen [43] und der verallgemeinerten Inversen von singulären linearen Operatoren [34] entstammen.

Theorem 4.3 (Charakterisierung konvergenter Iterationsoperatoren) [85]
Eine Matrix $\mathbf{T}$ ist genau dann konvergent, wenn

a) $\rho(\mathbf{T}) = 1$ *und*

b) $\gamma(\mathbf{T}) < 1$

sowie c) eine der folgenden äquivalenten Eigenschaften:

c') der Elementarteiler [43] *zum Eigenwert 1 im charakteristischen Polynom von $\mathbf{T}$ ist linear*

[2] Theorem 6.2.3, Bedingungen N45, N46, O47 und P48
[3] [18]:Theorem 7.3.6 und [43]:Theorem 9.2
[4] Theorem 8.2

c") $\operatorname{rang}(\mathbf{I}-\mathbf{T})^2 = \operatorname{rang}(\mathbf{I}-\mathbf{T})$ *(bzw.* $\operatorname{ind}(\mathbf{I}-\mathbf{T}) = 1$*)*

c"') $(\mathbf{I}-\mathbf{T})^{\#}$ *als Drazin-Inverse* [34] *existiert und* $\lim_{t\to\infty} \mathbf{T}^t = \mathbf{I} - (\mathbf{I}-\mathbf{T})(\mathbf{I}-\mathbf{T})^{\#}$ [34][5], [18][6] *gilt.*

4.3 Nicht-stationäre hierarchische Iterationen

Nicht-stationäre Iterationen haben die Eigenschaft, dass Iterations-Operatoren bzw. lineare Abbildungsvorschriften vom aktuellen Iterationsschritt abhängig sind. Es gibt zahlreiche Ursachen für variierende Iterations-Operatoren: die variable Reihenfolge, in der skalare Einträge oder Blöcke in Gauss-Seidel Iterationen aktualisiert werden; ein variabler Relaxationsparameter; die variable Auswahl "historischer" Iterationsvektoren in Semi-Iterativen Verfahren und die variable Anzahl innerer Iterationsschritte in hierarchischen Iterationen. Hierarchische Iterationen betten in jeden äußeren Iterationsschritt eine innere Iteration ein. Da der Typ und die Anzahl der sogenannten inneren Iterationsschritten variabel ist, verursacht sie einen nicht-stationären Iterations-Operator.
Ein **äußeres Splitting** $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ mit nicht-singulärer Matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ und ein **inneres Splitting** $\mathbf{M} = \mathbf{F} - \mathbf{G}$ mit nicht-singulärer Matrix $\mathbf{F} \in \mathbb{R}^{n\times n}$ überführen das Gleichungssystem $\boldsymbol{\pi}\cdot\mathbf{Q} = \mathbf{0}$ in das innere Iterationsschema

$$\tilde{\boldsymbol{\pi}}(\tilde{t}+1) = \boldsymbol{\pi}(t)\cdot\mathbf{N}\cdot\mathbf{F}^{-1} + \tilde{\boldsymbol{\pi}}(\tilde{t})\cdot\underbrace{\mathbf{G}\cdot\mathbf{F}^{-1}}_{\mathbf{H}}. \tag{4.14}$$

Hier ist $\mathbf{H}$ der **innere Iterations-Operator** und $\tilde{\boldsymbol{\pi}}$ der **innere Iterationsvektor**. Das innere Iterationsschema wird in ein äußeres Iterationsschema eingebettet, indem der Startvektor der inneren Iteration $\tilde{\boldsymbol{\pi}}(0)$ als aktueller Iterationsvektor der äußeren Iteration $\boldsymbol{\pi}(t)$ gewählt wird und nach q inneren Iterationsschritten der äußere Iterationsvektor durch $\boldsymbol{\pi}(t+1) \leftarrow \tilde{\boldsymbol{\pi}}(q)$ aktualisiert wird. Der **äußere Iterations-Operator** $\mathbf{T}_q$ ist explizit und kompakt darstellbar durch

$$\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t)\cdot\underbrace{\left(\mathbf{N}\cdot\mathbf{F}^{-1}\sum_{j=0}^{q-1}\mathbf{H}^j + \mathbf{H}^q\right)}_{\mathbf{T}_q}, \tag{4.15}$$

vgl. Lemma 4.4. Die Ausprägung von $\mathbf{T}_q$ hängt von der Anzahl innerer Iterationen ab. Deswegen ist der Iterations-Operator $\mathbf{T}_q$ durch q indiziert.

Lemma 4.4 (Geschlossene Darstellung des hierarchischen Iterations-Operators $\mathbf{T}_q$)
Behauptung: $\mathbf{T}_q = \mathbf{N}\cdot\mathbf{F}^{-1}\sum_{j=0}^{q-1}\mathbf{H}^j + \mathbf{H}^q$.
Beweis: *(Induktion über q):*
Es ist $T_1 = \mathbf{N}\cdot\mathbf{F}^{-1} + \mathbf{H}$ *und die Gl. 4.14 zur Beschreibung eines inneren Schritts zeigt, dass* T_1 *korrekt ist. Der Induktionsschritt ist ebenfalls unter Anwendung von Gl. 4.14 durchführbar,*

[5] Theorem 8.2.2
[6] Lemma 7.6.11

denn es gilt:

$$\begin{aligned}\tilde{\boldsymbol{\pi}}(q+1) &= \tilde{\boldsymbol{\pi}}(0)\cdot\mathbf{N}\cdot\mathbf{F}^{-1} + \tilde{\boldsymbol{\pi}}(q)\cdot\mathbf{H} \\ &= \tilde{\boldsymbol{\pi}}(0)\cdot\mathbf{N}\cdot\mathbf{F}^{-1} + \tilde{\boldsymbol{\pi}}(0)\cdot(\mathbf{N}\cdot\mathbf{F}^{-1}\sum_{j=0}^{q-1}\mathbf{H}^j + \mathbf{H}^q)\cdot\mathbf{H} \\ &= \tilde{\boldsymbol{\pi}}(0)\cdot\left(\mathbf{N}\cdot\mathbf{F}^{-1}\sum_{j=0}^{q}\mathbf{H}^j + \mathbf{H}^{q+1}\right)\end{aligned}$$

Äußeres und inneres Splitting der Koeffizientenmatrix können explizit durch ein Splitting dargestellt werden. Die Variabilität der inneren Iteration in Form unterschiedlicher Ausprägungen von $\mathbf{T}_q$ führt zu **Multi-Splittings** $\mathbf{Q} = \mathbf{M}_q - \mathbf{N}_q$ mit $\mathbf{T}_q = \mathbf{N}_q \cdot \mathbf{M}_q^{-1}$. Jeder Iterations-Operator $\mathbf{T}_q$ induziert ein (nicht eindeutiges) Splitting $\mathbf{Q} = \mathbf{M}_q - \mathbf{N}_q$, vgl. Lemma 4.5. Dies verdeutlicht, dass eine hierarchische Iteration mit fester Anzahl innerer Schritte wieder zu einer stationären Iteration mit singulärem Splitting $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ degeneriert. Für $q = 1$ kann das induzierte Splitting direkt angegeben werden, es ist $\mathbf{M}_1 = \mathbf{F}$, $\mathbf{N}_1 = \mathbf{N} + \mathbf{G}$ und $\mathbf{T}_1 = (\mathbf{N} + \mathbf{G}) \cdot \mathbf{F}^{-1}$, vgl. Lemma 4.5.

Lemma 4.5 (Induktion eines Splittings aus Iteration-Operator $\mathbf{T}_q$ [73]) .
***Behauptung**: Es existieren Matrizen $\mathbf{M}_q$ und $\mathbf{N}_q$ mit $\mathbf{T}_q = \mathbf{N}_q \cdot \mathbf{M}_q^{-1}$ und $\mathbf{M}_q - \mathbf{N}_q = \mathbf{Q}$.*
***Beweis**: Nach Voraussetzung ist $\mathbf{M} = (\mathbf{I} - \mathbf{H}) \cdot \mathbf{F}$ und $\mathbf{M}^{-1}$ existiert. Deswegen ist auch $(\mathbf{I} - \mathbf{H})$ nicht-singulär und die Teleskopsummengleichung $(\mathbf{I} - \mathbf{H}) \sum_{j=0}^{q-1} \mathbf{H}^j = \mathbf{I} - \mathbf{H}^q$ kann überführt werden in*

$$\sum_{j=0}^{q-1}\mathbf{H}^j = (\mathbf{I} - \mathbf{H})^{-1} \cdot (\mathbf{I} - \mathbf{H}^q).$$

Damit kann der Iterations-Operator $\mathbf{T}_q$ wie folgt transformiert werden:

$$\begin{aligned}\mathbf{T}_q = \mathbf{N}\cdot\mathbf{F}^{-1}\sum_{j=0}^{q-1}\mathbf{H}^j + \mathbf{H}^q &= \mathbf{N}\cdot\underbrace{\mathbf{F}^{-1}\cdot(\mathbf{I}-\mathbf{H})^{-1}}_{\mathbf{M}^{-1}}\cdot(\mathbf{I}-\mathbf{H}^q) + \mathbf{H}^q \\ &= \mathbf{N}\cdot\mathbf{M}^{-1}\cdot(\mathbf{I}-\mathbf{H}^q) + \mathbf{H}^q \\ &= \underbrace{\left[\mathbf{N} + \mathbf{H}^q\cdot(\mathbf{I}-\mathbf{H}^q)^{-1}\cdot\mathbf{M}\right]}_{\mathbf{N}_q}\cdot\underbrace{\left[(\mathbf{I}-\mathbf{H}^q)^{-1}\cdot\mathbf{M}\right]^{-1}}_{\mathbf{M}_q^{-1}}\end{aligned}$$

Damit hat man für $\mathbf{Q}$ ein einfaches Splitting aus dem Iterations-Operator $\mathbf{T}_q$ induziert bzw. konstruiert, denn für die Matrizen $\mathbf{M}_q$ und $\mathbf{N}_q$ gilt:

$$\begin{aligned}\mathbf{M}_q - \mathbf{N}_q &= (\mathbf{I}-\mathbf{H}^q)^{-1}\cdot\mathbf{M} - \mathbf{H}^q\cdot(\mathbf{I}-\mathbf{H}^q)^{-1}\cdot\mathbf{M} - \mathbf{N} \\ &= (\mathbf{I}-\mathbf{H}^q)\cdot(\mathbf{I}-\mathbf{H}^q)^{-1}\cdot\mathbf{M} - \mathbf{N} \\ &= \mathbf{M} - \mathbf{N} = \mathbf{Q}\end{aligned}$$

Die Matrizen $\mathbf{M}_q$ und $\mathbf{N}_q$ sind nach [73][7] eindeutig. Für den Spezialfall $q = 1$ ist $\mathbf{M}_1 = \mathbf{F}$, $\mathbf{N}_1 = \mathbf{N} + \mathbf{G}$ und damit $\mathbf{T}_1 = (\mathbf{N} + \mathbf{G}) \cdot \mathbf{F}^{-1}$.

[7] Lemma 2.3

Dem durch die Gl. 4.14 und 4.15 beschriebenen mathematischen Modell einer hierarchischen Iteration liegen globale Splittings zugrunde, die den gesamten Iterationsvektor gleichermaßen erfassen. Dieses Modell kann für blockorientierte Varianten hierarchischer Iterationen verfeinert werden, in denen für jeden Block eine quasi eigenständige lokale innere Iteration definierbar ist. Nachfolgend wird stets ein *BJAC*-Splitting als äußeres Splitting vorausgesetzt, d.h. es ist $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ und

$$\mathbf{M} = \mathbf{D}\left(\mathbf{Q}[1,1], \ldots, \mathbf{Q}[N,N]\right).$$

Für $X = 1, \ldots, N$ sind

$$\begin{aligned} \mathbf{F}_X &= \mathbf{D}(\mathbf{I}, \ldots, \mathbf{B}_X, \ldots, \mathbf{I}) \in \mathbb{R}^{n \times n} \\ \mathbf{G}_X &= \mathbf{D}(\mathbf{I} - \mathbf{Q}[1,1], \ldots, \mathbf{C}_X, \ldots, \mathbf{I} - \mathbf{Q}[N,N]) \in \mathbb{R}^{n \times n} \end{aligned}$$

blockabhängige *innere, lokale Splittings* mit $\mathbf{Q}[X,X] = \mathbf{B}_X - \mathbf{C}_X$ ($\mathbf{B}_X, \mathbf{C}_X \in \mathbb{R}^{n(X) \times n(X)}$). Des Weiteren sei

$$\mathcal{Q}_t \triangleq \{q(t,1), \ldots, q(t,N)\} \in \mathbb{N}^N$$

die Menge der blockabhängigen inneren Iterationsschritte im äußeren Schritt t, $\mathbf{H}_X \triangleq \mathbf{G}_X \cdot \mathbf{F}_X^{-1} \in \mathbb{R}^{n \times n}$ der blockabhängige innere Iterationsoperator und $\mathbf{E}_X = \mathbf{D}(\mathbf{0}, \ldots, 0, \mathbf{I}, \mathbf{0}, \ldots, 0)$ ein Maskierungsoperator, in der $\mathbf{I}$ genau der X-te Hauptdiagonalblock ist. Es ist bekannt, dass der äußere Iterationsoperator $\mathbf{T}_{\mathcal{Q}_t}$ explizit und kompakt darstellbar ist durch

$$\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t) \cdot \underbrace{\left(\sum_{X \in \mathcal{Z}^0} \mathbf{E}_X \left[\mathbf{N} \cdot \mathbf{F}_X^{-1} \sum_{j=0}^{q(t,X)-1} \mathbf{H}_X^j \; + \; \mathbf{H}_X^{q(t,X)} \right] \right)}_{\mathbf{T}_{\mathcal{Q}_t}}. \tag{4.16}$$

Diese Darstellung kann in eine weniger kompakte Form überführt werden, die mehr strukturelle Informationen preisgibt, siehe Lemma 4.6. Es sei

$$\mathbf{Q}_X \triangleq \mathbf{Q}[X,X], \; \mathbf{H}_X \triangleq \mathbf{C}_X \cdot \mathbf{B}_X^{-1} \in \mathbb{R}^{n(X) \times n(X)},$$

$$\boldsymbol{\mathcal{K}}(\mathcal{Q}_t) \triangleq \begin{pmatrix} \mathbf{Q}_1^{-1} \cdot (\mathbf{I} - \mathbf{H}_1^{q(t,1)}) & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{-1} \cdot (\mathbf{I} - \mathbf{H}_2^{q(t,2)}) & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{Q}_N^{-1} \cdot (\mathbf{I} - \mathbf{H}_N^{q(t,N)}) \end{pmatrix}$$

und

$$\boldsymbol{\mathcal{L}}(\mathcal{Q}_t) = \begin{pmatrix} \mathbf{H}_1^{q(t,1)} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2^{q(t,2)} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{H}_N^{q(N,2)} \end{pmatrix}.$$

Dann gilt in kompakter Schreibweise

$$\mathbf{T}_{\mathcal{Q}_t} = \mathbf{N} \cdot \boldsymbol{\mathcal{K}}(\mathcal{Q}_t) + \boldsymbol{\mathcal{L}}(\mathcal{Q}_t). \tag{4.17}$$

Lemma 4.6 (Induktion eines Splittings aus Iterationsoperator $\mathbf{T}_{\mathcal{Q}_t}$ [73]) .
***Behauptung**: Es existieren Matrizen $\mathcal{K}(\mathcal{Q}_t)$ und $\mathcal{L}(\mathcal{Q}_t)$, so dass der Iterations-Operator $\mathbf{T}_{\mathcal{Q}_t}$ darstellbar ist als*

$$\mathbf{T}_{\mathcal{Q}_t} = \mathbf{N} \cdot \mathcal{K}(\mathcal{Q}_t) \; + \; \mathcal{L}(\mathcal{Q}_t).$$

***Beweis**: Der Iterations-Operator*

$$\mathbf{T}_q = \mathbf{N} \cdot \mathbf{M}^{-1} \cdot (\mathbf{I} - \mathbf{H}^q) \; + \; \mathbf{H}^q \tag{4.18}$$

aus Lemma 4.5 beschreibt den Spezialfall, dass im Schritt t für alle Blöcke die gleiche Anzahl $q = q(t,1) = \ldots = q(t,N)$ innerer Schritte ausgeführt wird. Die Darstellung von $\mathbf{T}_{\mathcal{Q}_t}$ ist aus der Struktur von $\mathbf{T}_q$ als Verallgemeinerung ableitbar. Da $\mathbf{M}$ eine Blockdiagonalmatrix ist, gilt dies auch für $\mathbf{M}^{-1}$, $\mathbf{F}$, $\mathbf{G}$ und damit auch für $\mathbf{H}$, $\mathbf{H}^q$ und schließlich auch für $\mathbf{M}^{-1} \cdot (\mathbf{I} - \mathbf{H}^q)$. Der X-te Block dieser Matrix ist $\mathbf{Q}_X^{-1} \cdot (\mathbf{I} - \mathbf{H}_X^q)$. Variiert man nun q blockweise, so gilt

$$\mathbf{T}_{\mathcal{Q}_t} = \mathbf{N} \cdot \underbrace{\mathbf{D}(\mathbf{Q}_1^{-1} \cdot (\mathbf{I} - \mathbf{H}_1^{q(t,1)}), \ldots, \mathbf{Q}_N^{-1} \cdot (\mathbf{I} - \mathbf{H}_N^{q(t,N)}))}_{\triangleq \mathcal{K}(\mathcal{Q}_t)} + \underbrace{\mathbf{D}(\mathbf{H}_1^{q(t,1)}, \ldots, \mathbf{H}_N^{q(t,N)})}_{\triangleq \mathcal{L}(\mathcal{Q}_t)},$$

vgl. auch [62]*, Gleichungen 4-8. Ebenso gilt*

$$\mathcal{K}(\mathcal{Q}_t) = \mathbf{D}(\mathbf{B}_1^{-1} \cdot \sum_{j=0}^{t(t,1)-1} \mathbf{H}_1^j, \ldots, \mathbf{B}_N^{-1} \cdot \sum_{j=0}^{t(t,N)-1} \mathbf{H}_N^j),$$

wenn man die zur Gl. 4.18 äquivalente Darstellung

$$\mathbf{T}_q = \mathbf{N} \cdot \mathbf{F}^{-1} \sum_{j=0}^{q-1} \mathbf{H}^j \; + \; \mathbf{H}^q$$

von $\mathbf{T}_q$ verwendet.

Zusammenfassung: Der Einfluss blockunabhängiger und blockabhängiger innerer Iterationen auf die Struktur des Iterations-Operators ist analytisch erfassbar. Der jeweilige Iterations-Operator korrespondiert zu einem Matrix-Splitting der Koeffizientenmatrix $\mathbf{Q}$.

4.4 Stochastische asynchrone Iterationen

Mathematische Modelle asynchroner Fixpunkt-Iterationen sind oft untersucht worden [9, 19, 14, 38, 63, 64, 76, 92]. Asynchronität in Fixpunkt-Iterationen hat zwei Eigenschaften: die Komponenten des Iterationsvektors können "entkoppelt" - mit individueller Rate - iteriert werden und Komponenten, die in die Aktualisierung anderer Komponenten einfließen, können unterschiedlichen "historischen" Iterationsschritten entstammen (Iteration vom Grad > 1, vgl. Abschnitt 4.1). Asynchrone Iterationen sind wie auch hierarchische Iterationen nicht-stationäre Iterationen, in denen der Iterations-Operator variiert. In hierarchischen Iterationen verursacht die variable Anzahl innerer Iterationen nicht-stationäre Iterations-Operatoren. In asynchronen Iterationen verursacht die variable Auswahl zu iterierender Komponenten und die variable Auswahl "historischer" Iterationsvektoren einen nicht-stationären Iterations-Operator.

Mathematische Modelle asynchroner Iterationen generalisieren das mathematische Modell einer gewöhnlichen stationären, synchronen Iteration der Form $\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t) \cdot \mathbf{T}$. Der stochastische Rückgriff auf historische Iterationsvektoren wird formal durch Vektoren $\boldsymbol{\pi}^1(t), \ldots, \boldsymbol{\pi}^N(t)$ mit

$$\boldsymbol{\pi}^r(t) \triangleq \left(\boldsymbol{\pi}_{[1]}(\tau_1^r(t)), \ldots, \boldsymbol{\pi}_{[N]}(\tau_N^r(t))\right) \in \mathbb{R}^n, \tag{4.19}$$

beschrieben. $\boldsymbol{\pi}^r(t)$ "modelliert" eine **asynchrone Rekombination** der Komponenten des Iterationsvektors $\boldsymbol{\pi}_{[1]}, \ldots, \boldsymbol{\pi}_{[N]}$. Per Definition sei $\boldsymbol{\pi}^r(t)$ die "lokale Sicht" auf den globalen Iterationsvektor, die im Schritt t in die Aktualisierung von $\boldsymbol{\pi}_{[r]}$ einfließt. Die asynchrone Rekombination beitragender Komponenten ist abhängig von einer Familie zeitdiskreter stochastischer Ketten $\tau_1^1(t), \ldots, \tau_N^N(t)$. Die Kette $\tau_s^r(t)$ "modelliert" den Rückgriff auf die beitragende Komponente $\boldsymbol{\pi}_{[s]}$ ("s"=**s**ender) zur Aktualisierung von $\boldsymbol{\pi}_{[r]}$ ("r"=**r**eceiver). Des Weiteren sei $\mathcal{I}^t \subseteq \{1, \ldots, N\} \triangleq \mathcal{Z}^0$ eine **Aktualisierungsmenge** von Komponentenindizes, die im Schritt t iteriert bzw. aktualisiert werden. Der synchrone Iterations-Operator $\mathbf{T} \in \mathbb{R}^{n \times n}$ sei blockstrukturiert und die Blockstruktur sei konsistent zur Blockstruktur von $\mathbf{Q}$ (vgl. Abschnitt 3.2 zur Notation der Blockstruktur) und $\mathbf{T}_{[r]} \in \mathbb{R}^{n \times n(r)}$ sei die Submatrix der r-ten Block-Spalte. Damit erhält man das allgemeine Modell einer asynchronen Iteration nach Bertsekas und Tsitsiklis [19][8]

$$\boldsymbol{\pi}_{[r]}(t+1) = \begin{cases} \boldsymbol{\pi}_{[r]}(t) & \text{falls } r \notin \mathcal{I}^t \\ \boldsymbol{\pi}^r(t) \cdot \mathbf{T}_{[r]} & \text{falls } r \in \mathcal{I}^t. \end{cases} \tag{4.20}$$

Im Spezialfall einer **partiell-asynchronen** Iteration ist der Grad B a priori bekannt. In diesem Fall kann die Dynamik der $\boldsymbol{\pi}^r(t)$ Vektoren in die Dynamik des Iterations-Operators "verschoben" und dadurch die Dynamik der asynchronen Iteration insgesamt eingegrenzt werden. Hierfür definiert man einen expandierten Historienvektor $\tilde{\boldsymbol{\pi}}(t)$ (Def. s.u.) mit statischer Zusammensetzung, der alle zulässigen lokalen Sichten (endlich viele) integriert. Durch Variation des Iterations-Operators wird die jeweilige Sicht für den aktuellen Iterationsschritt extrahiert. Die formale Beschreibung des expandierten **Historienvektors** ist

$$\tilde{\boldsymbol{\pi}}(t) \triangleq (\boldsymbol{\pi}(t), \boldsymbol{\pi}(t-1), \ldots, \boldsymbol{\pi}(t-B)) \in \mathbb{R}^{n \cdot (B+1)}. \tag{4.21}$$

Eine Alternative zur Gl. 4.20 ist

$$\tilde{\boldsymbol{\pi}}(t+1) = \tilde{\boldsymbol{\pi}}(t) \cdot \mathbf{T}_{\mathcal{I}^t, \mathcal{D}^t} \tag{4.22}$$

Der nicht-stationäre und ebenfalls expandierte Iterations-Operator $\mathbf{T}_{\mathcal{I}^t, \mathcal{D}^t} \in \mathbb{R}^{n \cdot (B+1) \times n \cdot (B+1)}$ verschiebt ($\searrow$) "historische" Komponenten

$$\begin{array}{rccccccccc} \tilde{\boldsymbol{\pi}}(t) = & (& \boldsymbol{\pi}(t) & & \boldsymbol{\pi}(t-1) & \ldots & \boldsymbol{\pi}(t-B+1) & & \boldsymbol{\pi}(t-B) &) \\ & & & \searrow & & & & \searrow & & \\ \tilde{\boldsymbol{\pi}}(t+1) = & (& \underbrace{\boldsymbol{\pi}(t+1)}_{aktualisiert} & & \boldsymbol{\pi}(t) & \ldots & \boldsymbol{\pi}(t-B+2) & & \boldsymbol{\pi}(t-B+1) &) \end{array}$$

[8] Bertsekas und Tsitsiklis betrachten asynchrone iterative Verfahren in einem breiteren Kontext, insbesondere auch für nicht-lineare Funktionen. Deswegen verwenden ihre Modelle eine funktionale Notation mit $\boldsymbol{\pi}_{[r]}(t+1) = \boldsymbol{\pi}_{[r]}(t)$ bzw. $\boldsymbol{\pi}_{[r]}(t+1) = f_r(\boldsymbol{\pi}^r(t))$, die von konkreten Eigenschaften von f (linear, nicht-linear etc.) zunächst abstrahiert.

und aktualisiert $\boldsymbol{\pi}_{[r]}(t)$ in $\boldsymbol{\pi}(t+1)$ für alle $r \in \mathcal{I}^t$.
Im Hinblick auf die Konvergenzanalyse asynchroner Iterationen ist es notwendig, die Menge prinzipiell möglicher Ausführungen asynchroner Iterationen durch gewisse Anforderungen einzuschränken. Nachfolgend ist ein Katalog typischer (parametrisierter) Anforderungen angegeben. Diese Anforderungen können in Implementierungen asynchroner Iterationen erfüllt werden, vgl. Abschnitte 6.1.3 und 6.1.4.

Bedingung 4.7 (Anforderungen an asynchrone Iterationen) .

"FIFO": *Daten werden nach dem "First-in First-out" Prinzip (keine Überholungen) kommuniziert, d.h.*

$$\forall r, s \,:\, \tau_s^r(0) \leq \tau_s^r(1) \leq \tau_s^r(2) \ldots \tag{4.23}$$

"progress of inputs" ("admissible delays"): *Es wird nicht auf beliebig alte Komponenten historischer Iterationsvektoren zurückgegriffen bzw. es werden keine Komponenten beliebig oft benutzt, d.h.*

$$\forall\, r, s \,:\, \lim_{t \to \infty} \tau_s^r(t) = \infty \tag{4.24}$$

"partial asynchronism" ("regulated delays"): *Die asynchrone Iteration ist vom Grad D, d.h. die Differenz zwischen dem aktuellen Iterationsschritt und dem beitragender, historischer Komponenten des Iterationsvektors ist nach oben durch D beschränkt:*

$$\exists D \in \mathbb{N} \;:\forall\, r, s, t : t - \tau_s^r(t) \leq D \tag{4.25}$$

Die Bedingung 4.25 ist hinreichend für Bedingung 4.24.

"progress of updates" ("admissible updating sets"): *Alle Komponenten des Iterationsvektors werden unendlich oft aktualisiert , d.h.*

$$\forall\, s \,:\, |\{t | s \in \mathcal{I}^t\}| = \infty \tag{4.26}$$

"partial asynchronism" ("regulated updating sets"): *Jede Komponente des Iterationsvektors wird innerhalb von $S+1$ Iterationsschritten mindestens einmal aktualisiert, d.h.*

$$\exists S \in \mathbb{N} \;:\forall\, i : \bigcup_{t=i}^{i+S} \mathcal{I}^t \;=\; \{1, \ldots, N\} \tag{4.27}$$

Die Bedingung 4.27 ist hinreichend für Bedingung 4.26.

"data locality": *Für die Aktualisierung einer Komponenten steht immer die aktuellste Version derselben zur Verfügung:*

$$\forall\, r : r \in \mathcal{I}^t \Rightarrow \tau_r^r(t) = t \tag{4.28}$$

Das Szenario einer asynchronen Iteration heißt **zulässig**, wenn die Bedingungen 4.24 und 4.26 erfüllt sind. Ein Szenario heißt **reguliert**, wenn die Bedingungen 4.25, 4.27 und 4.28 erfüllt sind. $B = \max(D, S)$ ist die **Asynchronitätskonstante** [19]. In den Abschnitten 6.1.3 und 6.1.4 wird der Einfluss der Parameter D und S auf die Performance asynchroner Iterationen experimentell untersucht.

Kapitel 5

Verteilte sowie hierarchische und asynchrone Iterationen zur stationären Analyse von Markov-Ketten - *ASYNC*

Das Kapitel beschreibt ein neues numerisches Verfahren (*ASYNC*) zur stationären Analyse von Markov-Ketten. *ASYNC* basiert auf hierarchischen und asynchronen Erweiterungen einer blockorientierten Jacobi-Iteration (*BJAC*). Das Grundgerüst ist eine *BJAC*-Iteration, die asynchron und verteilt auf parallelen Rechenressourcen mit verteiltem Speicher ausgeführt wird. In die asynchronen Schritte der *BJAC*-Iteration werden Intra-Block-Iterationen eingebettet (hierarchische Iteration). Der Bezeichner *ASYNC* verweist auf die zentrale Eigenschaft der Implementierung: die Asynchronität in Berechnung und Kommunikation.
Das Kapitel ist wie folgt strukturiert: im Abschnitt 5.1 "Einleitung und Übersicht" werden die zentralen Eigenschaften von *ASYNC* bzgl. des mathematischen Modells, des algorithmischen Modells und der Rechnerarchitektur genannt und ihre Synergieeffekte als Motivation für deren Integration erläutert. Der Abschnitt 5.2 "Konvergenzverhalten" ist eine kritische Bestandsaufnahme bekannter Konvergenzresultate aus der Mathematik hinsichtlich ihrer Anwendbarkeit und praktischen Nützlichkeit im Kontext dieser Arbeit. Die eingesetzte parallele Rechnerarchitektur und das damit verbundene parallele Programmmodell werden im Abschnitt 5.3 erläutert. Anschließend werden im Abschnitt 5.4 "Algorithmisches Modell" die Besonderheiten des numerischen Lösungsverfahrens *ASYNC* und seiner Implementierung angesprochen. Die Voraussetzung für eine verteilte Berechnung - die Identifikation von Teilproblemen und die Berechnung einer Lastverteilung - wird im Abschnitt 5.5 "Lastverteilung und Kommunikation" behandelt und mit Beispielen unterlegt. Schließlich werden im Abschnitt 5.6 ausgewählte Implementierungsaspekte betrachtet.

5.1 Einleitung und Übersicht

Die essentiellen Eigenschaften von *ASYNC* sind:

1. **Mathematisches Modell**: Das Grundgerüst von *ASYNC* ist eine asynchrone und hierarchische *BJAC*-Iteration. In die asynchronen *BJAC*-Schritte sind Intra-Block-Iterationen (Hierarchie) und "on-the-fly" Aggregierungsschritte eingebettet.

2. **Algorithmisches Modell**: *ASYNC* nutzt als Basis-Datenstruktur zur Speicherung der Generatormatrix eine hierarchische Kronecker-Darstellung, fokussiert auf die verteilte Ausführung von asynchronen *BJAC*-Iterationsschritten im verteilten Speicher und nutzt Kommunikationsroutinen mit asynchroner Sende- und Empfangssemantik.

3. **Rechnerarchitektur**: *ASYNC* zielt auf parallele Rechenarchitekturen mit verteiltem Speicher ab (Rechner-Netzwerke und -Cluster).

Die Integration dieser Aspekte ist neu und durch zahlreiche Synergieeffekte motiviert:

1. Die Kronecker-Darstellung der Generatormatrix profitiert von einer parallelen Rechnerarchitektur, weil leicht verteuerte Rechenoperationen auf der Generatormatrix (bedingt durch generische Darstellung) bei einer verteilten Ausführung zusätzliche Rechenressourcen erschließen. Zudem wird in einem Rechnernetzwerk ein großer, wenn auch verteilter Arbeitsspeicher verfügbar gemacht. Eine Verteilung des Iterationsvektors ist attraktiv, weil der Platz-Flaschenhals (= Speicherung des Iterationsvektors) behoben wird.

2. Die Kronecker-Darstellung liefert eine Blockstruktur der Generatormatrix, an der blockorientierte Verfahren adaptieren können. Eine *BJAC*-Iteration liefert eine Dekomposition des Berechnungsproblems in Teilproblme, die auf der gegebenen Blockstruktur skalierbar ist und eine parallele Ausführung ermöglicht. Zudem reflektiert die Blockstruktur etwaige NCD-Charakteristika [42] im System, so dass Teilprobleme numerisch gering gekoppelt sind, ein Umstand, von dem parallele Ausführungen profitieren und der Aggregierungsschritte motiviert.

3. Eine verteilte Ausführung der *BJAC*-Iteration profitiert von der Kronecker-Darstellung, weil in jedem verteilten Arbeitsspeicher die vollständige Generatormatrix als Duplikat gehalten werden kann, was Datenabhängigkeiten verringert und gleichzeitig die Kommunikation flexibilisiert, sowie Lastumverteilungen erleichtert.

4. Verteilte, asynchrone Iterationen sind unabhängig von der Verfügbarkeit "aktueller" kommunizierter Daten und deswegen geeignet für Kommunikationsmedien mit geringer bzw. lastabhängig-schwankender Bandbreite. Das Kommunikationsmedium profitiert von verteilten, asynchronen Iterationen, die zu einer gleichmäßigeren Belastung der Kommunikationsressourcen führen, weil Rechenphasen (geringe Kommunikation) und Kommunikationsphasen nicht streng alternierend, sondern zeitlich überlappend ablaufen.

5. Asynchrone und hierarchische Iterationen sind flexibel ausführbar und können an Performance-Charakteristika des Kommunikationsmediums (adaptiv) angepasst werden (Stabilität und Robustheit). Dies beinhaltet die Steuerung von Kommunikationsraten und die Möglichkeit, lokale Iterationen in Abhängigkeit von den verfügbaren kommunizierten Daten selektiv auszuführen.

5.2 Konvergenzverhalten

Die grundsätzliche Zielsetzung dieser Arbeit ist die stationäre Analyse einer Markov-Kette. Dies erfordert die Lösung eines linearen Gleichungssystems $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ mit der Generatormatrix $\mathbf{Q}$, vgl. Abschnitt 3.1, insbesondere Proposition 3.4. Die Datenstruktur der Generatormatrix basiert hier auf einer hierarchischen Kronecker-Darstellung, vgl. Abschnitt 3.2. Diese Darstellung der Generatormatrix induziert eine Blockstruktur auf der Generatormatrix und das System $\mathbf{x} \cdot \mathbf{Q} = \mathbf{0}$ nimmt die Form

$$(\boldsymbol{\pi}_{[1]}, \ldots, \boldsymbol{\pi}_{[N]}) \begin{pmatrix} \mathbf{Q}[1,1] & \ldots & \mathbf{Q}[1,N] \\ \vdots & \ddots & \vdots \\ \mathbf{Q}[N,1] & \ldots & \mathbf{Q}[N,N] \end{pmatrix} = \mathbf{0}$$

an. Zur Lösung dieses strukturierten Gleichungssystems wird in *ASYNC* eine Block-Iteration angewendet. Die Basis-Iteration in *ASYNC* ist eine *BJAC*-Iteration. Ein *BJAC*-Iterationsschritt erfordert die Lösung N nicht-homogener Gleichungssysteme

$$\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]} \qquad \text{mit} \qquad \mathbf{b}_{[c]} \triangleq -\sum_{r \neq c} \boldsymbol{\pi}_{[r]} \cdot \mathbf{Q}[r,c] \quad \text{für alle} \quad c \in \{1, \ldots, N\} \triangleq \mathcal{Z}^0. \tag{5.1}$$

Die rechten Seiten $\mathbf{b}_{[c]}$ repräsentieren eine aggregierte Sicht auf die aktuelle Approximation von $\boldsymbol{\pi}$. Für Blöcke $\mathbf{Q}[r,c] \neq \mathbf{0}$ beinhalten die rechten Seiten Datenabhängigkeiten. Die Systeme in Gl. 5.1 werden approximativ durch eine eingebettete Iteration mit q inneren Schritten gelöst. Dieser Ansatz wird als hierarchische Iteration bezeichnet. Wenn q fest ist, liegt eine gewöhnliche stationäre Iteration vor (Abschnitt 4.2), anderenfalls eine nicht-stationäre hierarchische Iteration (Abschnitt 4.3). Des Weiteren werden die Systeme in Gl. 5.1 asynchron gelöst, vgl. Abschnitt 4.4 "Stochastische asynchrone Iterationen".

Stationäre und nicht-stationäre hierarchische Iterationen sowie stochastische asynchrone Iterationen sind in der Mathematik eingehend untersucht worden. In dieser Arbeit soll festgestellt werden, welche Aussagen zu Konvergenz und Konvergenzraten auf den hier betrachteten Fall einer Generatormatrix als Koeffizientenmatrix des zu lösenden Gleichungssystems übertragbar sind und so der Bewertung von *ASYNC* dienen können. Hierfür war es notwendig, den Stand der Theorie über die Konvergenz hierarchischer und asynchroner Iterationen zu sichten und im Kontext dieser Arbeit auf seine Relevanz hin zu prüfen. Im Kapitel 4 sind die mit hierarchischen und asynchronen Iterationen korrespondierenden mathematischen Modelle angegeben. Nachfolgend werden für diese Modelle Konvergenzresultate rekapituliert und einige Experimente zum Konvergenzverhalten dokumentiert.

5.2.1 Stationäres Szenario

Stationäre hierarchische Iterations-Operatoren werden durch *ASYNC* ausgeführt, wenn die Systeme in Gl. 5.1 iterativ mit einer festen Anzahl innerer Iterationen gelöst werden. Das Theorem 4.3 (Seite 30) charakterisiert konvergente stationäre Iterations-Operatoren. Jede Generatormatrix hat die Eigenschaft eine singuläre M-Matrix mit "Eigenschaft C" zu sein. Dies garantiert Konvergenz, vgl. Prop. 5.1.

Proposition 5.1 (Konvergenz stationäre Iteration) .
Jede Generatormatrix $-\mathbf{Q}$ ist eine singuläre M-Matrix mit der 'Eigenschaft C' [18][1], weil für

[1] Theorem 8.4.2

$s = 1$ und $\mathbf{P} = \Delta t\mathbf{Q} + \mathbf{I}$ mit $\Delta t = \max\{|q| : q$ ist Diagonaleintrag in $\mathbf{Q}\}$ gilt, dass $-\mathbf{Q} = s\mathbf{I} - \mathbf{P}$, $s \geq \rho(\mathbf{P}) = 1 > 0$ und $\mathbf{P} \geq \mathbf{0}$ (zeilenstochastisch) ist. Nach [18][2] bzw. [85][3] ist eine Matrix genau dann eine singuläre M-Matrix mit der 'Eigenschaft C', wenn ein reguläres Splitting zu einem Iterationsoperator $\mathbf{T}$ führt, für den die Bedingungen a) und c) aus Theorem 4.3 erfüllt sind. Wenn die Matrix zusätzlich irreduzibel ist, erfüllt bereits ein schwach reguläres Splitting die Bedingungen a) und c), siehe [91][4]. Insbesondere ist das BJAC-Splitting einer M-Matrix stets regulär [85]. Eine Eigenwert-Transformation gemäß $\mathbf{T}_\alpha = (1-\alpha)\mathbf{I} + \alpha\,\mathbf{T}$ bewirkt schließlich, dass die Magnitude von $\mathbf{T}_\alpha$ echt kleiner als 1 ist [91][5], also die Bedingung b) aus Theorem 4.3 erfüllt ist, siehe auch [85][6] und [18][7].

Für singuläre Systeme ($\rho(\mathbf{T}) = 1$) wird die asymptotische Konvergenzrate durch die Magnitude $\gamma(\mathbf{T})$ des Iterationsoperators $\mathbf{T}$ bestimmt [18, 79, 80] [8]. Theoretische asymptotische Konvergenzraten können gute Abschätzungen für praktische Konvergenzraten liefern, wie das nachfolgende Beispiel einer stationären hierarchischen Iteration verdeutlicht.

Beispiel 5.2 (Vergleich theoretische und praktische Konvergenzrate) .

Eine stationäre Iteration sei durch $\boldsymbol{\pi}(0) = (0.25, 0.25, 0.25, 0.25)$ und

$$\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t) \cdot \mathbf{T}(\lambda) \qquad \text{mit } \mathbf{T}(\lambda) \triangleq \frac{1}{1+\lambda} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \lambda & \lambda^2 \\ \frac{1}{\lambda} & 1 & \lambda & 0 \\ 0 & 0 & 0 & \lambda \end{pmatrix} \tag{5.2}$$

$(0 < \lambda \leq 1)$ definiert. Der Iterationsoperator $\mathbf{T}(\lambda)$ korrespondiert zu einer hierarchischen BJAC-JAC-Iteration mit 2 inneren JAC-Iterationen je 2×2 Block (vgl. Matrix $\mathbf{T}_2$ in Beispiel 5.6 im nächsten Abschnitt) angewendet auf die Generatormatrix $\mathbf{Q}$ aus Bsp. 3.5 mit $\mu = 1$. Man kann nachrechnen, dass $\rho(\mathbf{T}(\lambda)) = 1$ (für alle $0 < \lambda \leq 1$) und $\gamma(\mathbf{T}(\lambda)) = (\lambda+1)^{-1}$ ist. Die letzte Gleichung zeigt, dass mit der Zunahme von λ die Magnitude γ sinkt und wegen $R_\infty(\mathbf{T}(\lambda)) \triangleq -\log\gamma(\mathbf{T}(\lambda))$ die asymptotische Konvergenzrate steigt.
Die stationäre Lösung ist $\boldsymbol{\pi} = (0.25, 0.25, 0.25, 0.25)$ $(\lambda = 1)$ bzw. $(1-\lambda)\cdot(1-\lambda^4)\cdot(1, \lambda, \lambda^2, \lambda^3)$ $(0 \leq \lambda < 1)$, vgl. Bsp. 3.5 mit $\mu = 1$. Die Güte der gleichverteilten Startverteilung $\boldsymbol{\pi}(0)$ ist somit von λ abhängig und wird zunehmend besser (schlechter) für $\lambda \to 1$ $(\lambda \to 0)$. Die Kurven in Abb. 5.2 links (rechts) zeigen $\log_{10}\|\boldsymbol{\epsilon}(t)\|_\infty$ für $t = 0, 1, 3, 5, 9, 15$ $(\log_{10}\|\boldsymbol{\epsilon}(t)\|_\infty / \log_{10}\|\boldsymbol{\epsilon}(0)\|_\infty$ für $t = 1, 3, 5, 9, 15)$ jeweils in Abhängigkeit von λ (logarithmische Skala zur besseren Darstellung). Die Kurven im linken Diagramm zeigen, dass mit Ausnahme des ersten Schritts eine Zunahme von λ (erhöht asymptotische Konvergenzrate) die Effektivtät der Iterationsschritte verbessert (vertikaler Abstand zwischen Kurven wird größer). Der für kleine λ Werte erzielte

[2] Theorem 6.4.12 Bedingung F13 oder Theorem 7.6.20
[3] Theorem 1
[4] Theorem 3.18
[5] Theorem 3.18
[6] Theorem 2
[7] Theorem 7.6.19
[8] in [18] siehe Abschnitt 7.6

Vorteil wird nach wenigen Schritten kompensiert (Kurven fallen zunehmend stark ab). Die Kurven im rechten Diagramm zeigen den Fehler im Verhältnis zum Startfehler. Mit Zunahme von λ sinkt der relative Fehler schneller (vertikaler Abstand zwischen Kurven wird größer), wobei für kleine λ-Werte der relative Fehler in den ersten Schritten zunächst besser ist.

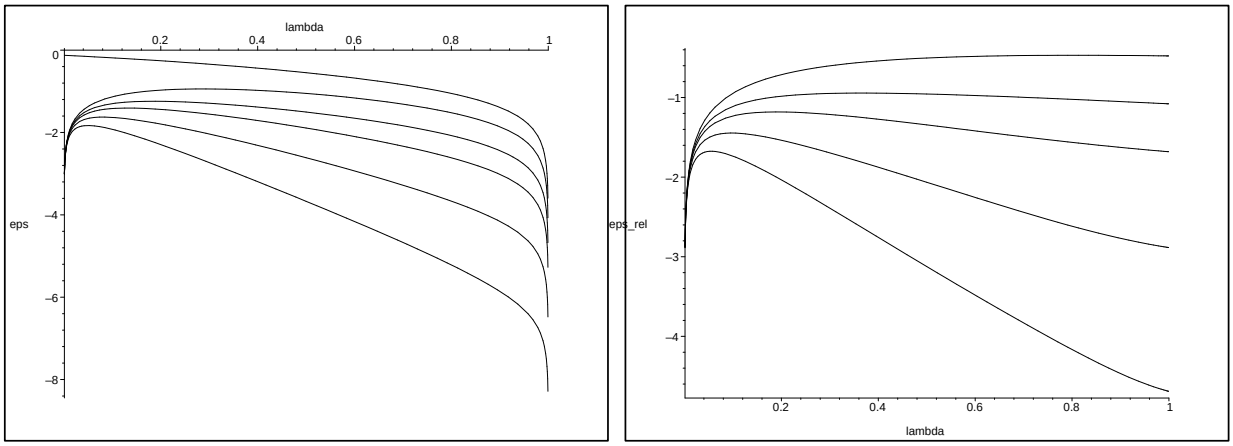

Abbildung 5.1: Links: Logarithmische Norm des Fehlervektors $\log_{10} \|\boldsymbol{\epsilon}(t)\|_\infty$ für $t = 0, 1, 3, 5, 9, 15$ (von oben nach unten) in Abhängigkeit von λ; Rechts: Logarithmische relative Norm des Fehlervektors $\log_{10} \|\boldsymbol{\epsilon}(t)\|_\infty / \log_{10} \|\boldsymbol{\epsilon}(0)\|_\infty$ für $t = 1, 3, 5, 9, 15$ (von oben nach unten) in Abhängigkeit von λ

Die theoretische (geschätzte) und tatsächliche Anzahl von Iterationen, die notwendig ist, um einen relativen Fehler $\epsilon = 0.01$ zu erreichen, sind in Tab. 5.2 für ausgewählte λ Werte aufgezählt. Der geschätzte Wert basiert auf der asymptotischen Konvergenzrate $R_\infty(\mathbf{T}(\lambda))$. Die theoretische Abschätzung ist - von Rundungsfehlern abgesehen - sogar exakt.

λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$-\frac{\log \epsilon}{R_\infty(\mathbf{T}(\lambda))}$	48	25	18	14	11	10	9	8	7
real t	49	26	18	14	12	10	9	8	8

Tabelle 5.1: Geschätze und reale Anzahl von Schritten t für relativen Fehler $\boldsymbol{\epsilon}(t)/\boldsymbol{\epsilon}(0) \triangleq \epsilon < 0.01$

5.2.2 Nicht-stationäres hierarchisches Szenario

Zur Strukturierung der Konvergenzresultate ist es hilfreich, Szenarien nicht-stationärer, hierarchischer Iterationen zu definieren. Später wird noch bei der *ASYNC*-Performance-Modellierung auf den Szenario-Begriff zurückgegriffen.

Definition 5.3 (Szenario hierarchische Iteration) .
*Die Sequenz $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$ mit $\mathcal{Q}_t \triangleq \{q(t,1), \ldots, q(t,N)\} \in \mathbb{N}^N$ spezifiziert zusammen mit dem Startvektor $\boldsymbol{\pi}(0)$ eindeutig die Folge der Iterationsvektoren $\boldsymbol{\pi}(1), \boldsymbol{\pi}(2), \ldots$ bzw. das **Szenario** einer hierarchischen BJAC-Iteration. Hierbei ist $q(t,c)$ die Anzahl innerer Schritte zur Lösung der Systems mit Index c in Gl. 5.1 im äußeren BJAC-Iterationsschritt t.*

Eine Konvergenzanalyse des mathematischen Modells erfordert es, zwischen folgenden Szenarien zu unterscheiden.

Beispiel 5.4 (Spezielle Szenarien einer hierarchischen Iteration) .

Aus der Menge möglicher Szenarien (Def. 5.3) sind Szenarien mit folgenden Restriktionen R1, R2 und R3 ableitbar:

R1: $\forall t, X$	$: q(t,X) = const$	$\Rightarrow$	*stationäres Szenario, q ist konstant*
R2: $\forall t, X, Y$	$: q(t,X) = q(t,Y)$	$\Rightarrow$	*nicht-stationär, aber q ist block<u>un</u>abhängig*
R3: $\forall t, u, X$	$: q(t,X) = q(u,X)$	$\Rightarrow$	*stationäres Szenario: q ist blockabhängig*

Bekannte Ergebnisse zu Konvergenz und Konvergenzraten sind in der nachfolgenden Proposition 5.5 zusammengetragen. Die Ergebnisse sind folgenden Abschnitten zugeordnet: Konvergenz (a und c), Konvergenzrate (b und d), nicht-singuläre Systeme (a und b), singuläre Systeme (c und d), stationäre Iterationen (R1 und R3) und nicht-stationäre Iterationen (R2 und $\bar{R}$).

Proposition 5.5 (Konvergenz hierarchische *BJAC*-Iteration) .
Für hierarchische BJAC-Iterationen (Gl. 4.16 und 4.17) sind folgende Konvergenzaussagen bekannt:

a: ***Konvergenz*** *ist garantiert* [73][9]*, wenn* $\mathbf{Q}$ ***nicht-singulär*** *ist,* $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ *ein* ***reguläres Splitting*** *ist und wenn abhängig von der Erfüllung der Restriktionen R1, R2 und R3 ($\bar{R}$ = keine Restriktion, vgl. Bsp. 5.4) gilt:*

R1,R3: $\forall X : \mathbf{M} = \mathbf{F}_X - \mathbf{G}_X$ *ist* ***schwach reguläres Splitting***
R2, $\bar{R}$*:* $\forall X : \mathbf{M} = \mathbf{F}_X - \mathbf{G}_X$ *ist* ***reguläres Splitting****.*

b: Die relative ***Konvergenzrate*** *ist abschätzbar* [73][10]*, wenn a) erfüllt ist. Dann gilt:*

R1:	$\rho(\mathbf{T}_q) \leq \rho(\mathbf{T}_p)$	$q \geq p$
R2:	$\rho(\Upsilon_r) \leq [\rho(\mathbf{T}_1)]^r$	*mit* $\Upsilon_r = \mathbf{T}_{\mathcal{Q}_r} \cdot \ldots \cdot \mathbf{T}_{\mathcal{Q}_1}$
R3:	$\rho(\mathbf{T}_{\mathcal{Q}}) \leq \rho(\mathbf{T}_{\hat{\mathcal{Q}}})$	*mit* $\mathcal{Q} = \{q(1), \ldots, q(N)\}$, $\hat{\mathcal{Q}} = \{\hat{q}(1), \ldots, \hat{q}(N)\}$, $\hat{q}(X) \leq q(X)$
$\bar{R}$*:*	$\rho(\mathbf{T}_{\mathcal{Q}_t}) \leq \rho(\mathbf{T}_1)$	*mit* $\mathcal{Q}_t := \{q(t,1), \ldots, q(t,N)\}$ *und* $q(t,X) \geq 1$

c: ***Konvergenz*** *ist garantiert, wenn* $\mathbf{Q}$ *eine* ***singuläre M-Matrix mit Eigenschaft C*** *ist,* $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ *ein* ***reguläres Splitting*** *und alle* $\mathbf{M} = \mathbf{F}_X - \mathbf{G}_X$ ***schwach reguläre Splittings*** *sind und*

R1,R3: $\forall X : \mathbf{G}_X \cdot \mathbf{F}_X^{-1}$ *hat positive* (> 0) *Diagonaleinträge* [82][11]*,*
R2,$\bar{R}$*:* *es existiert Norm* $\|\cdot\|$ *mit* $\|\mathbf{T}_{\mathcal{Q}_t}(\mathbf{I} - \mathbf{T}_{\mathcal{Q}_t})(\mathbf{I} - \mathbf{T}_{\mathcal{Q}_t})^{\#}\| < 1$*, siehe* [82][12]*.*

d: Im Gegensatz zum nicht-singulären Fall ist der Einfluss der Anzahl innerer Iterationen auf die Magnitude des Iterationsoperators bzw. auf die Konvergenzrate bereits für stationäre Iterationen unbekannt[13]*.*

Die zentrale Botschaft der Proposition 5.5 lautet, dass Konvergenzbedingungen für hierarchische Iterationen zur Lösungen singulärer Systeme formulierbar und auch in der Praxis erfüllbar sind

[9] Theorem 4.2 und Korollar 4.3, Theorem 6.4, Theorem 7.1, Theorem 7.3
[10] Korollar 5.2, Theorem 6.4, Theorem 7.2, Theorem 7.3
[13] Anfrage bei D.B. Szyld

(Teil c), aber leider kein analytischer Zusammenhang zwischen der Anzahl innerer Iterationen und der Konvergenzrate bekannt ist (Teil d). Für nicht-singuläre Systeme gilt, dass die theoretische Konvergenzrate mit der Zunahme innerer Iterationen q nicht sinkt. Selbst diese - aus praktischer Sicht wenig verwertbare Aussage - ist nicht auf den singulären Fall übertragbar, wie das Gegenbeispiel 5.7 zeigt. Zuvor wird im Bsp. 5.6 das Gegenbeispiel konstruiert.

Beispiel 5.6 (Konstruktion einer hierarchischen *BJAC-JAC* Iteration) .
Ausgehend von der Generatormatrix

$$-\mathbf{Q} = \left(\begin{array}{cc|cc} \lambda & -\lambda & 0 & 0 \\ -\mu & \lambda+\mu & -\lambda & \\ \hline 0 & -\mu & \lambda+\mu & -\lambda \\ 0 & 0 & -\mu & \mu \end{array}\right),$$

vgl. Bsp. 3.5, einem äußeren BJAC-Splitting $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ *mit*

$$\mathbf{M} = \left(\begin{array}{cc|cc} \lambda & -\lambda & 0 & 0 \\ -\mu & \lambda+\mu & 0 & 0 \\ \hline 0 & 0 & \lambda+\mu & -\lambda \\ 0 & 0 & -\mu & \mu \end{array}\right), \mathbf{N} = \left(\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ \hline 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right)$$

und einem inneren JAC-Splitting $\mathbf{M} = \mathbf{F} - \mathbf{G}$ *mit*

$$\mathbf{F} = \left(\begin{array}{cc|cc} \lambda & 0 & 0 & 0 \\ 0 & \lambda+\mu & 0 & 0 \\ \hline 0 & 0 & \lambda+\mu & 0 \\ 0 & 0 & 0 & \mu \end{array}\right), \mathbf{G} = \left(\begin{array}{cc|cc} 0 & \lambda & 0 & 0 \\ \mu & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \lambda \\ 0 & 0 & \mu & 0 \end{array}\right), \underbrace{\mathbf{G}\cdot\mathbf{F}^{-1}}_{\triangleq \mathbf{H}} = \left(\begin{array}{cc|cc} 0 & \frac{\lambda}{\lambda+\mu} & 0 & 0 \\ \frac{\mu}{\lambda} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{\lambda}{\mu} \\ 0 & 0 & \frac{\mu}{\lambda+\mu} & 0 \end{array}\right).$$

liefert Gl. 4.15 die Iterations-Operatoren $\mathbf{T}_1, \mathbf{T}_2$ *(1 bzw. 2 innere Iterationsschritte) mit*

$$\mathbf{T}_1 = \left(\begin{array}{cc|cc} 0 & \frac{\lambda}{\lambda+\mu} & 0 & 0 \\ \frac{\mu}{\lambda} & 0 & \frac{\lambda}{\lambda+\mu} & 0 \\ \hline 0 & \frac{\mu}{\lambda+\mu} & 0 & \frac{\lambda}{\mu} \\ 0 & 0 & \frac{\mu}{\lambda+\mu} & 0 \end{array}\right), \mathbf{T}_2 \left(\begin{array}{cc|cc} \frac{\mu}{\lambda+\mu} & 0 & 0 & 0 \\ 0 & \frac{\mu}{\lambda+\mu} & \frac{\lambda}{\lambda+\mu} & \frac{\lambda^2}{(\lambda+\mu)\,\mu} \\ \hline \frac{\mu^2}{(\lambda+\mu)\,\lambda} & \frac{\mu}{\lambda+\mu} & \frac{\lambda}{\lambda+\mu} & 0 \\ 0 & 0 & 0 & \frac{\lambda}{\lambda+\mu} \end{array}\right).$$

Es ist leicht zu verifizieren, dass $\mathbf{M}^{-1} \geq \mathbf{0}$, $\mathbf{N} \geq \mathbf{0}$, $\mathbf{F}^{-1} \geq \mathbf{0}$ *und* $\mathbf{H} \geq \mathbf{0}$ *gilt (äußeres und inneres Splitting ist regulär).* $\mathbf{T}_1$ *hat keine positiven Diagonaleinträge (verletzt hinreichende Bedingung c in Proposition 5.5). Eine innere JAC Iteration mit Relaxation (JOR) erzeugt positive Diagonaleinträge im modifizierten Iterationsoperator*

$$\mathbf{H}_\omega = (1-\omega)\mathbf{I} + \omega\mathbf{H}.$$

Wie bereits oben ausgeführt gibt es auf die Frage, welchen Einfluss q auf die theoretische Konvergenzrate einer *BJAC-JAC* Iteration zur Lösung eines singulären Gleichungssystems $\mathbf{x}\cdot\mathbf{Q}=\mathbf{0}$ hat, keine allgemeingültige Antwort. In dieser Arbeit soll jedoch für das konkrete System aus Bsp. 3.5 eine Antwort gegeben werden.

Beispiel 5.7 (Einfluss innerer Iterationen auf asymptotische Konvergenzrate) .
Untersuchungsgegenstand ist die BJAC-JAC-Iteration aus Bsp. 5.6. Es sei (willkürlich) $\mu=1$ *und für ein festes aber beliebiges* q *betrachte man die Familie der Iterationsoperatoren* $\mathbf{T}_q(\lambda)$ *mit* $\lambda>0$ *(zur Interpretation von* μ *und* λ *vgl. Bsp. 3.5). Man kann nachrechnen, dass* $\rho(\mathbf{T}_q(\lambda))=1$ *für alle zulässigen* λ *und* μ *ist (vgl. Bsp. 5.2), so dass die asymptotische Konvergenzrate* R_∞ *durch die Magnitude* $\gamma(\mathbf{T}_q(\lambda))$ *bestimmt wird. Zwischen der Magnitude und* q *bzw.* λ *existiert für* $q=1,2,3,4$ *und* $\lambda>0$ *eine analytische Beziehung*[14]*:*

$q=$	*1*	*2*	*3*	*4*
$\gamma(\mathbf{T}_q(\lambda))=$	$\dfrac{\sqrt{\lambda}}{\lambda+1}$	$\max\left(\dfrac{1}{\lambda+1},\dfrac{\lambda}{\lambda+1}\right)$	$\dfrac{\lambda^2+\lambda+1}{\lambda^2+2\lambda+1}$	$\max\left(\dfrac{\lambda^2}{\lambda^2+2\lambda+1},\dfrac{2\cdot\lambda}{\lambda^2+2\lambda+1}\right)$

Die Abb. 5.2 zeigt die asymptotische Konvergenzrate $R_\infty(\mathbf{T}_q(\lambda))$ *in Abhängigkeit von* λ *für* $q=1,2,3,4$*. Der Abbildung ist zu entnehmen, dass mit Zunahme von* q *nicht zwangsläufig* R_∞

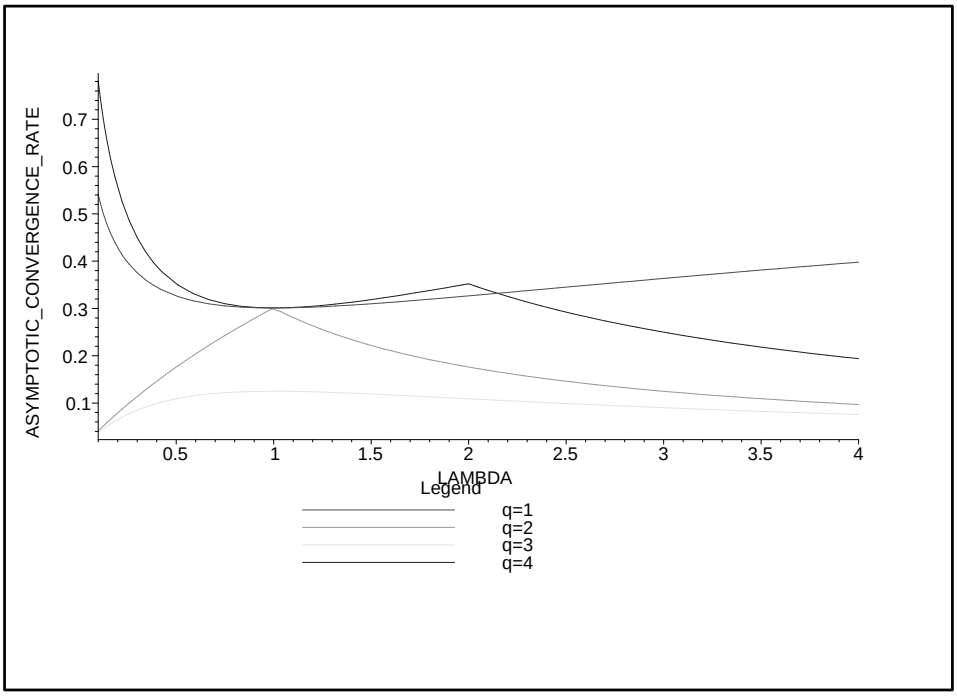

Abbildung 5.2: Asymptotische Konvergenzrate $R_\infty(\mathbf{T}_q(\lambda))$

steigt. Im Einzelnen gilt:

[14]Mit z.B. Maple^{TM} können aufwendige "händische" Berechnungen umgangen werden. Für $q>4$ werden die Ausdrücke sehr groß.

$$\begin{array}{rrcccccc}
\lambda \in (0,1); \lambda \in (1,2.15): & R_\infty(\mathbf{T_4}(\lambda)) & > & R_\infty(\mathbf{T_1}(\lambda)) & > & R_\infty(\mathbf{T_2}(\lambda)) & > & R_\infty(\mathbf{T_3}(\lambda)) \\
\lambda = 1: & R_\infty(\mathbf{T_4}(\lambda)) & = & R_\infty(\mathbf{T_1}(\lambda)) & = & R_\infty(\mathbf{T_2}(\lambda)) & > & R_\infty(\mathbf{T_3}(\lambda)) \\
\lambda \in (2.15,\infty): & R_\infty(\mathbf{T_1}(\lambda)) & > & R_\infty(\mathbf{T_4}(\lambda)) & > & R_\infty(\mathbf{T_2}(\lambda)) & > & R_\infty(\mathbf{T_3}(\lambda))
\end{array}$$

Beispielsweise liefert $q = 1$ (in realer Implementierung sehr preiswert) eine relativ gute (theoretische) Konvergenzrate (für $\lambda \in (2.15, \infty)$ relativ am besten), während $q = 3$ relativ am schlechtesten abschneidet (für alle λ).

5.2.3 Stochastisches asynchrones Szenario

In Ergänzung zum Szenarien-Begriff für hierarchische Iterationen kann auch ein Szenario asynchroner Iterationen definiert werden. Die Szenario-Definition und die grundlegenden Anforderungen an asynchrone Iterationen aus Bedingung 4.7 helfen wiederum Konvergenzresultate zu strukturieren.

Definition 5.8 (Szenario asynchrone Iteration) .
Das Szenario einer asynchronen Iteration ist durch die variable Auswahl zu iterierender Komponenten modelliert durch zeitdiskrete stochastische Ketten ("Aktualisierungsmengen")

$$\{\mathcal{I}^t \,|\, t \in \mathbb{N} \wedge \mathcal{I}^t \subseteq \mathcal{Z}^0 \wedge \mathcal{I}^t \neq \emptyset\}$$

und durch den variablen Rückgriff auf historische Iterationsvektoren modelliert durch zeitdiskrete stochastische Ketten ("Verzögerungen"), die nachfolgend in Matrizen subsummiert sind

$$\{\mathcal{D}^t \,|\, t \in \mathbb{N} \wedge \mathcal{D}^t \triangleq \begin{pmatrix} \tau_1^1(t) & \dots & \tau_1^N(t) \\ \vdots & \ddots & \vdots \\ \tau_N^1(t) & \dots & \tau_N^N(t) \end{pmatrix} \in \mathbb{N}^{N\times N} \wedge \forall t : 0 \leq \tau_s^r(t) \leq t\}$$

definiert.

Die Konvergenzanalyse asynchroner Iterationen geht auf Pionierarbeiten von Chazan und Miranker [38] und Baudet [9] aus den Jahren 1969 bzw. 1978 zurück.

Chazan und Miranker: Chazan und Miranker [38] haben gezeigt, dass ein kontrahierender synchroner Iterationoperator $\rho(\mathbf{T}) < 1$ für ein nicht-singuläres System notwendig und hinreichend für Konvergenz der asynchronen Iteration ist, wenn das Szenario die Eigenschaften 4.25, 4.26 und $|\mathcal{I}^t| = 1$ (nur eine Komponente wird je Schritt aktualisiert) erfüllt.

Baudet: Baudet [9] setzt ebenfalls $\rho(\mathbf{T}) < 1$ voraus, verallgemeinert aber das Chazan-Miranker-Modell, indem $\mathcal{I}^t$ frei wählbar ist und die Bedingung 4.25 durch die schwächere Bedingung 4.24 ersetzt wird. Allerdings müssen alle Komponenten r, r', die im Schritt t aktualisiert werden ($r, r' \in \mathcal{I}^t$), eine identische "lokale Sicht" auf den Iterationsvektor haben ($\boldsymbol{\pi}^r(t) = \boldsymbol{\pi}^{r'}(t)$), d.h. $\tau_1^r(t) = \tau_1^{r'}(t) \dots \tau_N^r(t) = \tau_N^{r'}(t)$. Baudet ermittelt eine untere Schranke für die asymptotische Konvergenzrate asynchroner Iterationen aus den Level-Mengen der synchronen Iteration

$$X(k) \triangleq \{\, \mathbf{x} : \|\mathbf{x} - \boldsymbol{\pi}\| \leq \|\mathbf{x} - \boldsymbol{\pi}(0)\|\, \rho(T)^k \,\}.$$

Die untere Schranke hängt von $\rho(T)$ und damit von der asymptotischen Konvergenzrate der synchronen Iteration ab. Baudet konstruiert aus Trajektorien der asynchronen Iteration Teilfolgen ("*Baudet-Folgen*") asynchroner Iterationsvektoren, deren Schritte bzgl. obiger Level-Mengen

einen "messbaren" Fortschritt in Richtung der Lösung garantieren. Es ist wichtig zu betonen, dass die Trajektorie - "modelliert" durch die Belegungen der $\mathcal{D}^t$ und $\mathcal{I}^t$ - im Detail berücksichtigt wird. Eine aggregierte Sicht auf Trajektorien, wie sie zum Beispiel die Asynchronitätskonstante B darstellt, ist nicht ausreichend. Die Notwendigkeit, einen detaillierten Einblick in den Ablauf der asynchronen Iteration zu erlangen, hat Casanova [35] motiviert, durch eine stochastische Modellierung des Szenarios die (implizite) Trajektorie zu ermitteln.

Bertsekas und Tsitsiklis: Bertsekas und Tsitsiklis untersuchen asynchron-iterative Verfahren in einem breiten Anwendungskontext [19]. In Erweiterung zu Baudet erlauben ihre Modelle asynchroner Iterationen unabhängige bzw. individuelle "lokale Sichten" (für $r, r' \in \mathcal{I}^t$ ist $\tau_1^r(t) \neq \tau_1^{r'}(t)) \ldots$ erlaubt). Aus der Vielzahl ihrer Ergebnisse seien nachfolgend 4 mit Verweis auf die Stellen in [19] zitiert:

1. **Analytischer Zusammenhang zwischen Asynchronität und theoretischer asynchroner Konvergenzrate** ([19], Abschnitt 1.4.2): Für eine Klasse von 2×2 Matrizen wird der Einfluss der Asynchronität auf die Konvergenzrate untersucht (wegen kleiner Dimension besteht ein analytischer Zusammenhang zwischen Szenario und Konvergenzrate). Die Asynchronität ist auf die $\tau_s^r(t)$ Variablen beschränkt, d.h. in jedem Schritt werden alle Komponenten aktualisiert ($\mathcal{I}^t = \mathcal{Z}^0$ für alle t). Dieser Spezialfall, der auch in [14, 84] betrachtet wird, hat die interessante Eigenschaft, dass aufgrund der Konstruktion des expandierten Iterationsoperators $\mathbf{T}_{\mathcal{I}^t,\mathcal{D}^t}$ sein Spektralradius nicht steigt, d.h. $\rho(\mathbf{T}_{\mathcal{I}^t,\mathcal{D}^t}) \leq \rho(\mathbf{T})$ gilt (Einträge aus $\mathbf{T}$ werden in Abhängigkeit von Werten der $\tau_s^r(t)$ Variablen in $\mathbf{T}_{\mathcal{I}^t,\mathcal{D}^t}$ "verteilt" und der Rest mit Nullen aufgefüllt).

2. **Relation synchrone und asynchrone theoretische Konvergenzrate** ([19], Abschnitte 6.1 und 6.2): Ein zulässiges Szenario und $\rho(\mathbf{T}) < 1$ sind hinreichend für asynchrone Konvergenz bzgl. einer (gewichteten) Maximumnorm[15]. Wenn das Szenario reguliert ist, kann analog zu Baudet eine untere Schranke für die theoretische asynchrone Konvergenzrate in Abhängigkeit von der theoretischen synchronen Konvergenzrate angegeben werden. Die theoretische asynchrone Konvergenzrate kann im Vergleich zum synchronen Gegenstück um den Faktor $2 \cdot B$ schlechter sein, d.h. asynchrone Iterationen können im Vergleich zu synchronen Iteration in einem Maße ineffektiver sein, das direkt vom Grad der Asynchronität B abhängt. Es ist wichtig zu betonen, dass sich die theoretische asynchrone Konvergenzrate verschlechtern kann, aber nicht muss. In den oben zitierten Arbeiten sind Beispiele angeführt, in denen die theoretische asynchrone Konvergenzrate im Vergleich zum synchronen Gegenstück besser ist. Es ist auch wichtig anzumerken, dass alle Aussagen nur auf theoretische Konvergenzraten zutreffen, die lediglich eine untere Schranke bzw. Abschätzung (asymptotische Konvergenzrate) für die tatsächliche Anzahl notwendiger Iterationsschritte liefern. Es bleibt anzumerken, dass $\rho(\mathbf{T}) = 1$ die hinreichenden Bedingungen dieser Konvergenzuntersuchungen nicht erfüllt, weil ein synchroner Schritt "theoretisch" keinen Fortschritt in Richtung der Lösung erzielt.

3. **Synchrone n-Schritt Konvergenz (Spektralradius=1)** ([19], Abschnitt 7.2) Für synchrone Iterationen mit $\rho(\mathbf{T}) = 1$ und der Eigenschaft nach (spätestens) $n = \dim \mathbf{T}$ Schrit-

[15] Ein zulässiges Szenario heißt hier totale Asynchronität (Annahme 6.1.1), $\rho(\mathbf{T}) < 1$ erfüllt die "Synchrone Konvergenzbedingung mit geschachtelten Level-Mengen" (Annahme 6.2.1.a) und die Maximumnorm erfüllt die "Box-Bedingung" (Annahme 6.2.1.b). Damit ist nach Proposition 6.2.1 Konvergenz garantiert.

ten einen Fortschritt in Richtung der Lösung zu erzielen, verlangsamt sich die theoretische asynchrone Konvergenzrate ebenfalls um den Faktor $2 \cdot B$[16]. Bertsekas und Tsitsiklis geben mehrere Charakterisierungen und Beispiele für synchrone n-Schritt Konvergenz. Leider sind die Ergebnisse hier nur bedingt verwertbar, weil die Schranken (synchroner und asynchroner Fall) von der Problemgröße n (hier sehr groß) abhängen.

4. **Stochastischer, irreduzibler und aperiodischer synchroner Iterations-Operator** ([19], Abschnitt 7.3) Für den Spezialfall einer stochastischen, irreduziblen und aperiodischen Matrix $\mathbf{T}$ mit $\rho(\mathbf{T}) = 1$ ist geometrische Konvergenz garantiert und der (vom Szenario abhängige) Grenzwert $\lim_{t\to\infty} \boldsymbol{\pi}(t)$ kann nach oben und unten abgeschätzt werden, vgl. auch die Originalarbeit hierzu von Lubachevsky und Mitra [76] unten. Die Anwendbarkeit dieses Ergebnisses ist durch starke Annahmen bzgl. $\mathbf{T}$ limitiert. Für eine Power-Iteration gilt, dass die Annahmen bzgl. $\mathbf{T}$ erfüllt sind, sofern die (zeitkontinuierliche) Markov-Kette irreduzibel und ergodisch ist, für komplexere Iterationen (z.B. $BJAC$) gilt dies aber nicht.

Lubachevsky und Mitra: Lubachevsky und Mitra führen in [76] einen Konvergenzbeweis für nicht-negative und irreduzible Iterationsoperatoren $\mathbf{T}$ mit $\rho(\mathbf{T}) = 1$ unter der Annahme eines regulierten Szenarios und Bedingung 4.28. Ihre Hauptaussage ist, dass die Distanzfunktion

$$\mathrm{d}(\boldsymbol{\pi}(t)) \triangleq \max_{0\le\tau\le B} \max_{1\le i\le n} \frac{\boldsymbol{\pi}_i(t-\tau)}{\boldsymbol{\pi}_i} - \min_{0\le\tau\le B} \min_{1\le i\le n} \frac{\boldsymbol{\pi}_i(t-\tau)}{\boldsymbol{\pi}_i} \tag{5.3}$$

für $t \to \infty$ geometrisch gegen 0 konvergiert und dass ein $c \in \mathbb{R}$ (hängt vom Szenario ab) mit $\lim_{t\to\infty} \boldsymbol{\pi}(t) = c \cdot \boldsymbol{\pi}$ existiert. Die Distanzfunktion d basiert nicht auf einzelnen $\boldsymbol{\pi}(t)$ Vektoren, sondern auf einer Folge B aufeinanderfolgender Iterationsvektoren (vgl. äußere Maximum- bzw. Minimumbildung) - eine "Unschärfe", die durch die Asynchronität verursacht ist. Zum anderen basiert die Distanzfunktion auf einer projektiven (Pseudo)-Metrik, vgl. Abb. 5.3.
Lubachevsky und Mitra benutzen die Level-Mengen

$$X(k) = \{\, \mathbf{x} : d(\mathbf{x}) \le C \cdot (1-\sigma^r)^k \,\}.$$

C ist eine Konstante, die (analytisch) von der Lösung $\boldsymbol{\pi}$ und vom Startvektor $\boldsymbol{\pi}(0)$ abhängt, $\sigma < 1$ ist eine Konstante, die (analytisch) von $\boldsymbol{\pi}$ und $\mathbf{T}$ abhängt und $r \triangleq 1 + D + (n-1)(S+D)$ ist ebenfalls konstant (Definition von D und S vgl. 4.25 und 4.27). Es wird gezeigt, dass die Folge asynchroner Iterationsvektoren spätestens nach r Schritten einen Level-Mengen-Übergang vollzieht, also einen messbaren Fortschritt in Richtung der Lösung erzielt, d.h. es ist

$$\boldsymbol{\pi}(t) \in X(k) \;\forall\, t \ge k \cdot r.$$

Mit Zunahme der Asynchronität (=Zunahme von D und S) erhöht sich $r \triangleq 1+D+(n-1)(S+D)$ - die Anzahl von Schritten, die notwendig ist um einen Level-Mengen-Übergang zu vollziehen. Gleichzeitig steigt die obere Schranke $C \cdot (1-\sigma^r)^k$ der Distanzfunktion d, d.h. die obere Schranke für den messbaren Fortschritt hin zur Lösung, der durch den Level-Mengen-Übergang erzielt wird. Im Vergleich zu Baudet [9] nutzt man hier eine aggregierte Sicht auf das Szenario der asynchronen Iteration (r-Konstante) und betrachtet nicht unmittelbar deren Trajektorie. Die r-Konstante beeinflußt hier die Konstruktion der Level-Mengen und der Teilfolge, während bei

[16] Obwohl die Aussage intuitiv erscheint, ist der formale Beweis in Proposition 7.2.3 recht lang.

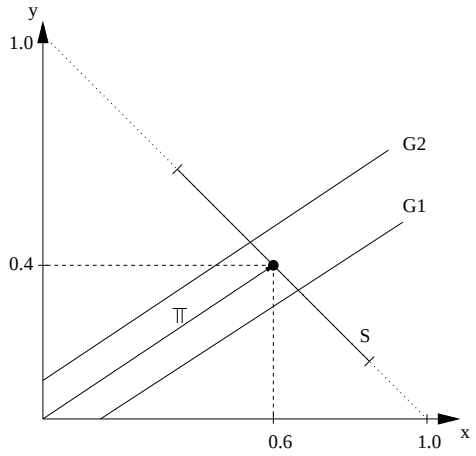

Abbildung 5.3: **Veranschaulichung der projektiven Norm aus [76] in $\mathbb{R}^2$.** Es sei $\boldsymbol{\pi} = (0.6, 0.4)$ die stationäre Verteilung. Die Geraden $G1$ oder $G2$ sind alle Punkte, die bzgl. der projektiven Norm aus [76] den Abstand 0.25 von $\boldsymbol{\pi}$ haben, d.h. es ist $G1 \cup G2 = \{\, (x,y) > (0,0) \,|\, \max\left(\frac{x}{0.6}, \frac{y}{0.4}\right) - \min\left(\frac{x}{0.6}, \frac{y}{0.4}\right) = 0.25 \,\}$. Im Vergleich dazu ist die Strecke S die Menge normierter Punkte, deren Abstand von $\boldsymbol{\pi}$ bzgl. der Maximum-Norm ≤ 0.25 ist, d.h. es ist $S = \{\, (x,y) \,|\, x + y = 1 \wedge \max(|x - 0.6|, |y - 0.4|) \leq 0.25 \,\}$.

Baudet Level-Mengen gänzlich unabhängig vom Szenario definiert werden. Bei Baudet werden Level-Mengen über Spektralradien definiert und man schätzt die relative Verzögerung der asynchronen Iteration im Vergleich zur synchronen Iteration ab. Bei Lubachevsky und Mitra fließen Eigenschaften des synchronen Iterationsoperators $\mathbf{T}$ lediglich über σ ein, eine Konstante die aus $\mathbf{T}$ vergleichsweise einfach berechenbar ist. Das Lubachevsky-Mitra Resultat ist in seiner praktischen Anwendbarkeit durch 3 Faktoren limitiert: alle Abschätzungen benutzen die unbekannte Lösung $\boldsymbol{\pi}$, in die Abschätzung fließt die Problemgröße n ein und die Annahmen bzgl. des synchronen Iterations-Operators (irreduzibel) sind sehr stark. Beispielsweise ist der synchrone Iterations-Operator einer hierarchischen *BJAC-JAC* Iteration mit zwei inneren Iterationen $\mathbf{T}_2$ (Bsp. 5.6) nicht irreduzibel.

Beidas et al, Strikwerda: stochastische Konvergenzanalyse Beidas und Papavassilopoulos [14] und Strikwerda [92] sind Referenzen für stochastische Konvergenzanalysen asynchroner Iterationen. Dem Ansatz in [14] liegen folgende Annahmen zugrunde: 1. die Asynchronität ist auf die $\tau_s^r(t)$ Variablen beschränkt ($\mathcal{I}^t = \mathcal{Z}^0$ für alle t) 2. es gilt Annahme 4.25 und 3. die zeitdiskreten stochastischen Ketten $\bar{\tau}_s^r(t) \triangleq t - \tau_s^r(t)$ (mit Zustandsraum $\{1, 2, \ldots, D\}$) sind Markov-Ketten und voneinander unabhängig. Zur Konvergenzanalyse werden die N^2 vielen zeitdiskreten Markov-Ketten als eine einzelne zeitdiskrete Markov-Kette $\{T(t)\}$ interpretiert. Ihr Zustandsraum ist wegen der Unabhängigkeit gleich dem Kreuzprodukt der $\bar{\tau}_s^r$ Zustandsräume und ihre Zustandsübergangswahrscheinlichkeiten das Kronecker-Produkt der Matrizen der Zustandsübergangswahrscheinlichkeiten der $\bar{\tau}_s^r$-Ketten (wird als gegeben angenommen). Es wird gezeigt, dass jeder der $D \cdot N^2$ Zustände der großen Markov-Kette $\{T(t)\}$ mit einem explizit definierbaren "expandierten" Iterationsoperator $\in \mathbb{R}^{n \cdot D \times n \cdot D}$ korrespondiert, der aus dem synchronen Iterationsoperator $\mathbf{T} \in \mathbb{R}^{n \times n}$ generierbar ist. Es sei $\mathbf{T}_t$ der "Zustand" ($\triangleq$ "expandierter" Iterationsoperator $\in \mathbb{R}^{n \cdot D \times n \cdot D}$) der großen Markov-Kette T im Schritt t. Somit erhält man eine

stochastische nicht-stationäre Iteration

$$\tilde{\boldsymbol{\pi}}(t+1) = \tilde{\boldsymbol{\pi}}(t) \cdot \mathbf{T}_t,$$

in der eine zeitdiskrete Markov-Kette $\{T(t)\}$ durch ihre Trajektorie die Auswahl des Iterationsoperators $\mathbf{T}_t$ "moduliert".

Moga, Dubois: Experimentelle Konvergenzanalyse Moga und Dubois [84] untersuchen experimentell den Einfluss der Asynchronität auf die Anzahl notwendiger Iterationsschritte bis zum Erreichen einer Genauigkeitsschranke. Hierbei betrachten sie eine Menge konvergenter synchroner Iterations-Operatoren $\mathbf{T} \in \mathbb{R}^{2\times 2}$ und untersuchen für jeden einzelnen Operator $\mathbf{T}$ den Einfluss der Asynchronität.
Eine Monte-Carlo Simulation stochastischer Variablen für Rechen- und Kommunikationszeiten emuliert eine Ausführung der asynchronen Iteration und erzeugt so eine Belegung der Variablen des asynchronen Szenarios, vgl. Def. 5.8. Hierbei emulieren sie die bekannten Typen asynchroner Iterationen: 1. Asynchronität ist auf $\mathcal{D}^t$ beschränkt, 2. Asynchronität ist auf $\mathcal{I}^t$ beschränkt. Jedes Szenario definiert eine Folge asynchroner Iterations-Operatoren und somit eine vom Startvektor ausgehende Trajektorie von Iterationsvektoren. Für einen einzelnen synchronen Operator $\mathbf{T} \in \mathbb{R}^{2\times 2}$ werden 100 Szenarien emuliert/simuliert und die mittlere Länge der Trajektorien bis zum Erreichen der Genauigkeitsschranke ermittelt. Diese Länge wird mit der deterministischen Anzahl synchroner Iterationsschritte bis zum Erreichen der Genauigkeitsschranke ins Verhältnis gesetzt (=asynchroner Speed-Up). Wenn die Asynchronität auf $\mathcal{I}^t$ beschränkt ist, stellen sie fest, dass der asynchrone Speed-Up über dem betrachteten Definitionsbereich $\subset \mathbb{R}^{2\times 2}$ mit $516^2 = 266\,256$ Matrizen für eine hohe Anzahl der Matrizen kleiner als 1 ist, also eine asynchrone Ausführung keine Beschleunigung erzielt. Sie beobachten aber auch einen Kulminationspunkt, in dem der Speed-Up sehr hoch wird und der durch Spektraleigenschaften erklärbar ist (dominanter Eigenwert ist negativ, deswegen oszilliert die synchrone Iteration und konvergiert langsam; eine asynchrone Ausführung verbessert die Spektraleigenschaften).

5.2.4 Diskussion

Die Tab. 5.2.4 stellt zusammenfassend Konvergenzresultate für hierarchische und asynchrone Iterationen aus der Literatur mit Referenzen in Übersichtsform dar. Ein Iterations-Operator mit Spektralradius 1 ($\rho(\mathbf{T}) = 1$) verweist auf den hier relevanten Fall: die Lösung eines singulären Gleichungssystems.
Die Intention mathematischer Modelle für Fixpunkt-Iteration ist die Ableitung von Konvergenzresultaten. Hierbei wird überprüft, inwieweit Struktureigenschaften der Koeffizientenmatrix $\mathbf{Q}$ in Abhängigkeit von der Auswahl des Splittings $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ auf $\mathbf{T}$ übertragbar und dann für die Bewertung der Konvergenz(rate) nützlich sind. Hierbei treten zwei Zielkonflikte auf. Der erste Zielkonflikt ist die Ausdrucksmächtigkeit von Splittings, um Eigenschaften der Iteration (hierarchisch, asynchron) zu erfassen versus der analytischen Handhabbarkeit der resultierenden generischen Klassen von Iterations-Operatoren. Der zweite Zielkonflikt sind schwache Annahmen bezüglich der Koeffizientenmatrix versus ableitbarer Aussagen zum Iterations-Operator. Bedingt durch diese Zielkonflikte sind Konvergenzaussagen für kombiniert asynchrone und hierarchische Iterationen (= komplexes Splitting bzw. Iterations-Operator) angewendet auf singuläre Gleichungssysteme (schwache Annahme bzgl. Koeffizientenmatrix) schwierig.

	Konvergenz				Konvergenzrate			
	$\rho(\mathbf{T}) = 1$		$\rho(\mathbf{T}) < 1$		$\rho(\mathbf{T}) = 1$		$\rho(\mathbf{T}) < 1$	
Iteration		Ref		Ref		Ref		Ref
asynchron	$\oplus$	[76, 19, 94]	$\oplus$	[9, 19, 38],[14, 92][2]	$\odot$	[76] [19][1]	$\oplus$	[9, 19][1]
hierarchisch	$\oplus$	[82]	$\oplus$	[73, 95]	$\ominus$		$\odot$	[73, 95][1]
asy. + hier.	$\oplus$	[7, 93]	$\oplus$	[21, 62]	$\ominus$		$\ominus$	

Tabelle 5.2: $\oplus$="bekannt"; $\odot$=unter Einschränkungen bekannt bzw. Aussagen schwach; $\ominus$=unbekannt; [1]theoretische asynchrone Konvergenrate relativ zur theoretischen synchronen Konvergenzrate; [2]stochastische Konvergenzanalyse;

5.3 Parallele Rechnerarchitektur und Programmmodell

Die Abbildung 5.4 skizziert das lokale Rechnernetz am Lehrstuhl Informatik 4, Universität Dortmund, das als Laufzeitumgebung für *ASYNC* genutzt wird. Das Rechnernetz besteht aus 3 Teilen: zwei Rechner-Cluster ("Compute-Server") und eine typische IT-Arbeitsumgebung bestehend aus heterogenen Arbeitsplatzrechnern, 10 bzw. 100 MBit Ethernet-Verbindungen, File-, Web- und Print-Servern und sonstiger Peripherie. Die Rechner-Cluster sind für rechen- und kommunikationsintensive Anwendungen konzipiert und mit 1 GBit Switches und leistungsfähigen Einzelrechnern (4× SUN Fire V20 mit AMD Opteron Prozessoren, 64 bit, 2.4 GHz, 6 GB RAM und 6× Xeon Doppelprozessoren, 32 bit, 3.06 GHz, 6 GB RAM) ausgestattet.

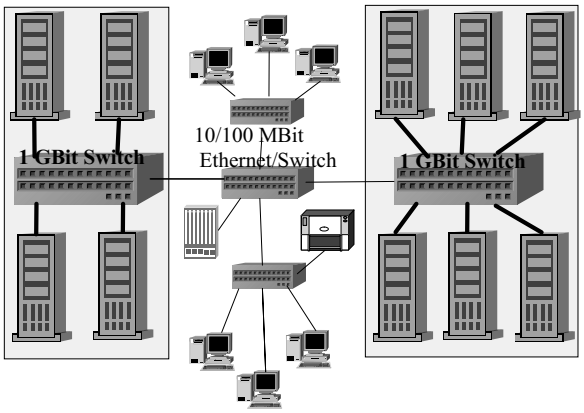

Abbildung 5.4: Lokales Netzwerk als Rechenressource

Eine verteilte oder parallele Lösung eines Berechnungsproblems erfordert, **Teilprobleme** (=Last) zu identifizieren und den Teilproblemen verfügbare Rechenressourcen zuzuordnen (= **Lastverteilung**). In *ASYNC* ist das Berechnungsproblem die Ausführung blockorientierter Jacobi (*BJAC*) Iterationsschritte. Die Ausführung erfordert die Lösung nicht-homogener Gleichungssysteme, die hier die Teilprobleme darstellen, vgl. Gl. 5.1. Die verfügbaren Rechenressourcen seien formal durch die Indexmenge $\mathcal{P} = \{1, \ldots, P\}$ repräsentiert. Die Lastverteilung $\mathcal{R}$ sei eine Abbildung von $\mathcal{P}$ auf disjunkte Teilmengen von $\mathcal{Z}^0 \triangleq \{1, \ldots, N\}$, der Indexmenge der Teilprobleme.

Definition 5.9 (Lastverteilung) .

Die Lastverteilung $\mathcal{R}$ ist definiert durch

$$\begin{aligned} \mathcal{R}(p) &= \{c \mid \textit{Prozess } p \in \mathcal{P} \textit{ berechnet } \boldsymbol{\pi}_{[c]}\} \\ \mathcal{R}^{-1}(c) &= \{p \in \mathcal{P} \mid \textit{Prozess } p \textit{ berechnet } \boldsymbol{\pi}_{[c]}\} \end{aligned}$$

In *ASYNC* hält jeder Prozess p "seine" Komponenten des Iterationsvektors $\boldsymbol{\pi}_{[r]}$ $(r \in \mathcal{R}(p))$ für andere Prozesse nicht zugreifbar im lokalen Speicher, vgl. Abb. 5.5 links. Des Weiteren hält jeder Prozess eine Kopie der gesamten Generatormatrix im lokalen Arbeitsspeicher. Dies wird durch die sehr platzeffiziente Datenstruktur der Generatormatrix ermöglicht, vgl. Abschnitt 3.2. Zur Aktualisierung des anteiligen Iterationsvektors (=Iterationsschritt) wird die aktuelle Version desselben und Daten aus dem Kommunikationspuffer (alles im lokalen Speicher) gelesen. Der Kommunikationspuffer gewährleistet eine asynchrone, von der Verfügbarkeit kommunizierter Daten unabhängige Iteration. Der Inhalt des Kommunikationspuffers wird durch kollaborierende Prozesse in unregelmäßigen Abständen (Asynchronität) aktualisiert. Die Kommunikation basiert auf asynchronen Sende- und Empfangsroutinen (Message-Passing Paradigma), vgl. Abschnitte 5.6.2 und 5.6.3. Das Berechnungsmodell ist vom Typ **Single-Program Multiple-Data**, weil jeder *Worker*-Prozess das gleiche Programm ausführt, aber auf unterschiedlichen Daten (= Komponenten des Iterationsvektors) rechnet.

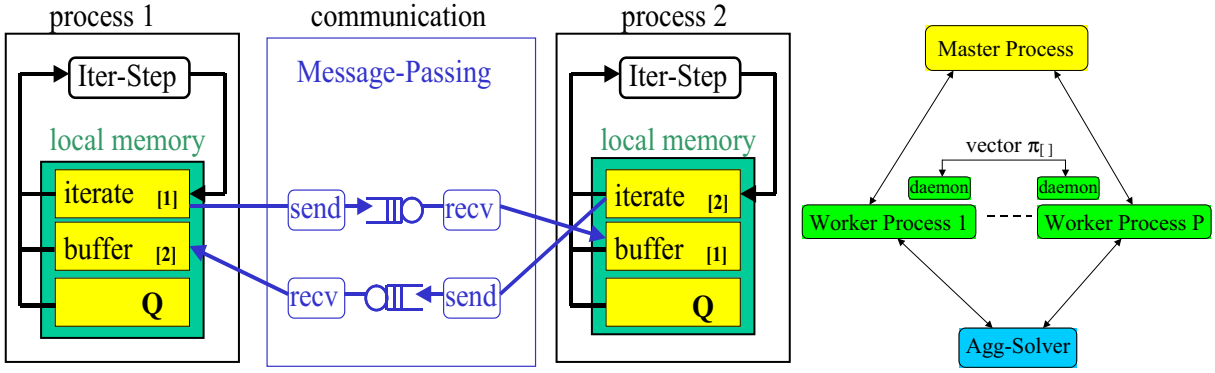

Abbildung 5.5: Links: *ASYNC* Berechnungsmodell; Rechts: Prozess-Typen

Worker-Prozesse realisieren die verteilte, numerische Berechnung, vgl. Abb. 5.5 rechts. Der *Master*-Prozess verantwortet koordinierende und steuerende Funktionen, die eine globale Sicht auf den Zustand der Berechnung erfordern. *Daemon*-Prozesse wickeln die Prozess-Kommunikation ab, hier wesentlich die Kommunikation zwischen *Worker*-Prozessen. Darüber hinaus realisiert ein *Agg-Solver*-Prozess "on-the-fly" Aggregierungsschritte, d.h. Iterationsschritte in einem niederdimensionalen Lösungsraum.

5.4 Algorithmisches Modell

Mit der Lastverteilung $\mathcal{R}$ (Def. 5.9) kann für die Gl. 5.1 ein grundlegendes algorithmisches Modell zur verteilten Ausführung von *BJAC*-Schritten abgeleitet werden. Das algorithmische *ASYNC*-

1. **for all** $p' \in \mathcal{P}$ **do in parallel**
2. **choose** $c \in \mathcal{R}(p')$
3. **solve** $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$

Abbildung 5.6: *ASYNC* algorithmischer Rahmen: Block-Jacobi Iteration

Gerüst in Abb. 5.6 beinhaltet 3 verschachtelte iterative Prozesse und realisiert eine hierarchische Iteration. Es wird zwischen einer *Inter-Block-Iteration* (kurz Block-Iteration) und einer *Intra-Block-Iteration* unterschieden. Weil die Anzahl der Blöcke oft die Anzahl der verfügbaren Rechenressourcen übersteigt ($|\mathcal{Z}^0| > |\mathcal{P}|$) und einzelne Prozesse für die Iteration mehrerer Komponenten des Iterationsvektors verantwortlich sind, kann die Block-Iteration in eine *Inter-Prozess-Iteration* und eine *Intra-Prozess-Iteration* unterteilt werden. Diese Unterteilungen sind in der Abb. 5.7 veranschaulicht. Auf oberster, Inter-Prozess-Ebene realisiert *ASYNC* ei-

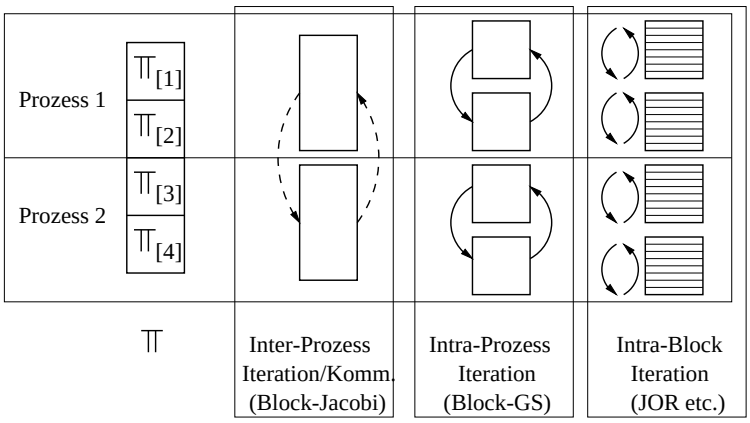

Abbildung 5.7: Die Hierarchie der Iterationsprozesse in *ASYNC*

ne verteilte, asynchrone *BJAC* Iteration. Auf mittlerer, Intra-Prozess-Ebene können Bereiche des Iterationsvektors in lokale Blöcke strukturiert sein. Wenn die Blöcke in Zeile 2 des algorithmischen Modells 5.6 in gleichbleibender Reihenfolge gewählt werden, realisiert *ASYNC* eine Block-Gauss-Seidel (*BGS*) Iteration, weil die im lokalen Speicher verfügbaren Komponenten des Iterationsvektors natürlich ohne kommunikationsbedingte Verzögerungen in Berechnungen einfließen. Die unterste, Intra-Block Iteration resultiert aus der Größe der Blöcke, die eine direkte Lösung verbietet und eine iterative Lösung (z.B. *JOR*, *SOR*) erzwingt. Die Intra-Block-Iteration ist im Algorithmus 5.6 der 3. Zeile zuzuordnen. In Spezialfällen können einzelne Ebenen kollabieren, z.B. wenn nur eine Rechenressource verfügbar ist (oberste Ebene), jeder Rechenprozess nur einen Block iteriert (mittlere Ebene) und jeder Block einelementig ist (unterste Ebene). Die Blockgröße ist durch die Modellstruktur bestimmt, zu Skalaren degenerierte Blöcke treten praktisch nie auf.
Die Rechenprozesse müssen Daten (Vektoren) austauschen, weil Datenabhängigkeiten bestehen, die sich in den rechten Seiten $\mathbf{b}_{[c]}$ des Gleichungssystems ausdrücken. Die rechten Seiten $\mathbf{b}_{[c]}$ sind aggregierte Sichten auf den Kontext der Berechnung von $\boldsymbol{\pi}_{[c]}$. Die Datenabhängigkeit ist

hinsichtlich der Datenproduzenten präzisierbar:

$$\mathbf{b}_{[c]} = -\sum_{p \in \mathcal{P}} \mathbf{b}^p_{[c]} \qquad \text{mit} \qquad \mathbf{b}^p_{[c]} = \sum_{\substack{r \in \mathcal{R}(p) \\ r \neq c}} \boldsymbol{\pi}_{[r]} \cdot \mathbf{Q}[r, c] \tag{5.4}$$

Der additive Beitrag $\mathbf{b}^p_{[c]}$ für $\mathbf{b}_{[c]}$ wird per Definition durch den Datenproduzenten p erbracht, weil Prozess p für die Berechnung der $\boldsymbol{\pi}_{[r]}$ mit $r \in \mathcal{R}(p)$ verantwortlich ist, vgl. Def. 5.4.
ASYNC ermöglicht zu Testzwecken zwischen Varianten der algorithmischen Ausführung und deren Implementierung zu wählen. Die Auswahl wird im C-Source-Code mit Präprozessor-Direktiven `#define` bzw. `#undef` gesteuert und bleibt zur Laufzeit konstant. Zur Laufzeit wird lediglich überprüft, ob die gewählte C-Source-Code Ausprägung zulässig ist. Die Varianten können wie folgt kategorisiert werden:

- **Semantik kommunizierter Vektoren:** *ASYNC* ermöglicht wahlweise Teile des Iterationsvektors oder additive Beiträge für die rechten Seiten der lokal zu lösenden Gleichungssysteme zu verschicken, vgl. Abschnitt 5.4.1.
- **Varianten asynchroner Iterationen:** *ASYNC* implementiert verschiedene Modellvarianten asynchroner Iterationen. Die Varianten werden im Abschnitt 5.4.2 erläutert und später experimentell untersucht, vgl. Abschnitte 6.1.3 “Asynchronität und Lösungszeit” und 6.1.4 “Totale vs. Partielle Asynchronität”.
- **Selektive Iteration:** *ASYNC* implementiert mehrere Kriterien für die Auswahlreihenfolge der zu iterierenden Komponenten des Iterationsvektors, vgl. Abschnitt 5.4.3
- **Steuerung der Kommunikation:** *ASYNC* implementiert eine bedarfgesteuerte Kommunikation, vgl. Abschnitt 5.4.4.
- **Steuerung von Kontrollausgaben:** *ASYNC* erzeugt diverse Kontrollausgaben in frei wählbarem Umfang auf dem Bildschirm oder für eine spätere Weiterverarbeitung in Dateien. Insbesondere wichtig sind Trace-Dateien, die Informationen zum Laufzeitverhalten sammeln, vgl. Abschnitt 6.1.
- **Realisierung Kommunikation und Kommunikationspuffer:** *ASYNC* nutzt ein Kommunikations-Interface, das auf die speziellen Bedürfnisse der verteilten numerischen Berechnung abstellt. Dieses Interface implementiert “Message-Passing”-Funktionalitäten, die wahlweise durch die Kommunikationstools *PVM* oder *MPI* angeboten werden. Des Weiteren implementiert *ASYNC* mehrere Kommunikations-Puffer Ausprägungen, vgl. Abschnitt 5.6.3.

Nachfolgend werden Varianten näher betrachtet, die einen engen Bezug zum mathematischen Modell haben und festlegen, was, wann, wie berechnet und kommuniziert wird.

5.4.1 Semantik kommunizierter Vektoren

In *ASYNC* werden wahlweise Vekoren vom Typ $\mathbf{b}^p_{[c]}$ oder $\boldsymbol{\pi}_{[r]}$ kommuniziert. Ein Datenproduzent p kann mittels Vektor-Matrix-Multiplikationen Vektoren $\mathbf{b}^p_{[c]}$ berechnen und verschicken, oder alternativ seine in $\mathbf{b}^p_{[c]}$ enthaltenen Komponenten $\boldsymbol{\pi}_{[r]}$ des Iterationsvektors verschicken, vgl.

Prozess p:	Prozess p:

for all $c \in \mathcal{Z}^0$
  if $\mathbf{b}^p_{[c]} \neq \mathbf{0}$ **then send** $\mathbf{b}^p_{[c]}$ **to** $\mathcal{R}^{-1}(c)$
	for all $c \in \mathcal{R}(p)$
	  for all $p' \in \mathcal{P}\backslash\{p\}$
	  if $\sum_{r \in \mathcal{R}(p')} \mathbf{Q}[c,r] \neq \mathbf{0}$ **then send** $\boldsymbol{\pi}_{[c]}$ **to** p'

Tabelle 5.3: **Variante 1 (links):** Datenproduzent p verschickt additiven Beitrag $\mathbf{b}^p_{[c]}$ für $\mathbf{b}_{[c]}$ an $\mathcal{R}^{-1}(c)$; **Variante 2 (rechts):** Datenproduzent p verschickt "seinen" Iterationsvektor $\boldsymbol{\pi}_{[c]}$ an alle Konsumenten p'

Tab. 5.3. Die Berechnung der Lastverteilung unterstützt beide Semantiken kommunizierter Daten und man kann fallbedingt entscheiden, welche Typen von Vektoren zu kommunizieren sind. Ein mögliches Auswahlkriterium hierfür ist die Höhe des Kommunikationsaufwands. Ein weiteres Auswahlkriterium ist die Frage, ob der Rechenaufwand der Vektor-Matrix-Multiplikationen produzentenseitig (Variante 1) oder konsumentenseitig (Variante 2) anzusiedeln ist. Für asynchrone, entkoppelte Berechnungen ist die Variante 2 effektiver, weil sie überflüssige Berechnungen beim Produzenten vermeidet. In entkoppelten Berechnungen - wie sie in *ASYNC* auftreten - kann der Fall eintreten, dass Vektoren $\mathbf{b}^p_{[c]}$ berechnet und verschickt werden, ohne dass ein Konsument sie zwischenzeitlich gelesen und benutzt hat. Die Variante 2 ist nur mit potentiell überflüssiger Kommunikation behaftet, die bei Bedarf (geringe Bandbreite im Kommunikationsmedium) durch ein "Demand-Driven Messaging" Protokoll (Def. 5.10) vermieden werden kann.

5.4.2 Varianten asynchroner Iterationen

Im Abschnitt 6.1.3 wird der Einfluss von Asynchronität auf Lösungszeiten experimentell untersucht. In Vorbereitung dafür werden an dieser Stelle Varianten schwach-asynchroner Iterationen definiert, vgl. Tab. 5.4. Das Laufzeitverhalten dieser Implementierungs-Varianten wird dann später experimentell untersucht. Alle nutzen Sende-Befehle mit asynchroner Semantik

Variante 1:	Variante 2:	Variante 3:

repeat
  for all $c \in \mathcal{R}(p')$
  sync $\mathbf{b}_{[c]}$
  for all $c \in \mathcal{R}(p')$
  solve $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$
  for all $c \in \mathcal{R}(p')$
  asy_send $\boldsymbol{\pi}_{[c]}$
until convergence
	repeat
	  for all $c \in \mathcal{R}(p')$
	  sync $\mathbf{b}_{[c]}$
	  solve $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$
	  asy_send $\boldsymbol{\pi}_{[c]}$
	until convergence
		repeat
		  barrier_sync
		  for all $c \in \mathcal{R}(p')$
		  solve $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$
		  asy_send $\boldsymbol{\pi}_{[c]}$
		until convergence

Tabelle 5.4: Schwach-asynchrone Iterationen und ihre Implementierungen (*Worker*-Prozess p')

(**asy_send**), kommunizieren Vektoren vom $\boldsymbol{\pi}_{[c]}$-Typ, nicht vom $\mathbf{b}^p_{[c]}$-Typ (vgl. "Semantik kommunizierter Vektoren" oben), und lösen ihnen zugeordnete Teilprobleme (**solve**).

Im Pseudo-Code der *Worker*-Prozesse aus Tab. 5.4 sind keine Befehle zum **Datenempfang** eingefügt! Es wird angenommen, dass der Datenempfang durch einen separaten "Message-Handler" realisiert wird. Deswegen müssen *Worker*-Prozesse nicht aktiv am Empfang partizipieren und Empfangsbefehle aufrufen, vgl. Abschnitt 5.6.3. Mit Ausnahme des "**sync**"-Befehls sind alle Befehle in dem Sinne atomar, dass Daten, die während ihrer Ausführung empfangen werden, keinen Einfluss auf die Ausführung ausüben und erst nach Befehlsende wirksam werden.

Variante 1: Die Variante 1 realisiert eine probabilistische Gauss-Seidel Iteration. In der Implementierung sind Synchronisations-, Rechen- und Datenversandphase strikt separiert, d.h. jeder Prozess erreicht zuerst eine Synchronisationsphase, danach löst er die ihm zugeordneten Probleme "**solve** $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$" und versendet neu berechnete Vektoren "**asy_send** $\boldsymbol{\pi}_{[c]}$". Der Befehl "**sync** $\mathbf{b}_{[c]}$" realisiert eine abgeschwächte Synchronisation. Der Befehl "**sync** $\mathbf{b}_{[c]}$" blockiert bis alle zu $\mathbf{b}_{[c]}$ beitragenden $\boldsymbol{\pi}_{[\cdot]}$-Vektoren aus dem aktuellen oder (=Freiheitsgrad bzw. probabilistischer Einfluss) aus dem vorherigen Iterationsschritt stammen ($\triangleq$Iteration vom Grad 1). Die schwache Asynchronität dieser Variante resultiert aus diesem Freiheitsgrad.
Die strikte Separierung der bedingten Synchronisationsphase und der Rechenphase ist eine nachteilige Implementierung, weil zu kommunizierende Daten ggf. "im Vorlauf" synchronisiert werden, ohne dass sie zu diesem Zeitpunkt bereits benötigt werden. Die nachfolgende Variante 2 behebt dieses Defizit. Trotzdem soll in den Messexperimenten im Abschnitt 6.1.3 auch diese - offensichtlich nachteilige - Variante 1 betrachtet werden, um ihren Nachteil auch in Messexperimenten zu belegen.

Variante 2: Die Variante 2 realisiert ebenfalls eine probabilistische Gauss-Seidel Iteration. In der Implementierung wird jedes Teilproblem c individuell synchronisiert, gelöst und das Ergebnis verschickt. Dies ist im Vergleich zu Variante 1 vorteilhaft, weil Synchronisationen, die auf der Aktualität kommunizierter Daten basieren, zeitnäher an der Nutzung kommunizierter Daten eingefügt sind. Die Einbettung des Datenversands in lokale Rechenphasen bewirkt eine gleichmäßige Belastung der Kommunikationsressourcen.

Variante 3: Gemäß der Eigenschaften einer Gauss-Seidel Iteration werden in Variante 1 und 2 die Komponenten des Iterationsvektors synchron iteriert und Komponenten, die in die Aktualisierung anderer Komponenten einfließen, enstammen dem aktuellen oder vorherigen Iterationsschritt (Iteration vom Grad 1).
In der Variante 3 wird die 2. Bedingung gelockert und eine Iteration vom Grad ∞ zugelassen. Der Synchronisationspunkt **barrier_sync** blockiert bis alle Prozesse diese Stelle im Source-Code erreicht und den Befehl aufgerufen haben. Der Synchronisationspunkt gewährleistet, dass alle Komponenten des Iterationsvektors mit der selben Rate iteriert werden. Die Beschränkung der Asynchronität auf die Kommunikationsverzögerungen wird zum Beispiel in [19, 14, 84] betrachtet, vgl. auch Abschnitt 4.4.
Die Variante 3 ohne den **barrier_sync**-Synchronisationspunkt ist eine vollständig asynchrone Iteration.

Partielle Asynchronität In der Bedingung 4.7 (Seite 36) ist partielle Asynchronität definiert basierend auf Aktualisierungsmengen (jede Komponente des Iterationsvektors wird innerhalb

von S Schritten mindestens einmal aktualisiert) und basierend auf der "Aktualität" beitragender Komponenten (die Differenz zwischen aktuellem Iterationsschritt und dem beitragender historischer Komponenten ist durch D beschränkt). In einer Implementierung, in der die partielle Asynchronität des 1.Typs gesteuert werden soll, muss die Information über die historische Reihenfolge aktualisierter Komponenten global zugreifbar sein. Wenn beispielsweise eine Komponente nicht in den letzten $S-1$ Schritten aktualisiert wurde, muss die Aktualisierung aller anderen Komponenten blockiert werden, bis die betreffende Komponente aktualisiert ist. In einer Architektur mit verteiltem Speicher ist eine korrekte Ermittlung der Reihenfolge lokaler Ereignisse schwierig und es wäre ein hoher Koordinationsaufwand nötig um sicherzustellen, dass alle *Worker*-Prozesse Zugriff auf die gleiche und aktuelle Reihenfolge aktualisierter Komponenten besitzen (Zugriff ist notwendig um ggf. Komponenten von der Aktualisierung auszuschließen). Deswegen wird die partielle Asynchronität des 1.Typs in *ASYNC* nicht unterstützt. Demgegenüber wird die partielle Asynchronität des 2.Typs in *ASYNC* unterstützt. Vektoren werden zusammen mit dem Iterationsschritt, in dem sie beim Produzenten berechnet wurden, verschickt. Der Konsument kann so leicht entscheiden, ob eine lokale Iteration erlaubt ist, oder ob ein weiterer Schritt eine maximal erlaubte Iterationsschrittdifferenz überschreiten würde und deswegen auf die Aktualisierung der Vektoren im Kommunikationspuffer gewartet werden muss.

5.4.3 Selektive Iteration

Die asynchrone Kommunikation und die Pufferung kommunizierter Daten beim Datenkonsumenten entkoppeln lokale Berechnungen und ermöglichen erst eine asynchrone Iteration. Die Entkopplung von Datenproduzenten und -konsumenten innerhalb asynchroner Iterationen flexibilisiert die lokale Auswahl zu lösender Teilprobleme. Die lokale Auswahl wird im algorithmischen Modell für *ASYNC* in Abb. 5.6 durch Zeile 2 repräsentiert. Weil die Anzahl der Blöcke in der Generatormatrix gewöhnlich die Anzahl verfügbarer Rechenprozesse übersteigt, ist jeder Rechenprozess p' für die Berechnung mehrerer Teilprobleme aus $\mathcal{R}(p')$ verantwortlich und muss innerhalb der *BJAC*-Blockiteration sukzessive Auswahlentscheidungen treffen (**choose**-Anweisung). Neben einer vorab definierten statischen Auswahlreihenfolge können auch dynamische Auswahlreihenfolgen benutzt werden, die Iterationsschrittdifferenzen zwischen den Teilproblemen, den Aktualisierungsgrad der in die Lösung lokaler Teilprobleme einfließenden Daten oder die bereits erzielte Genauigkeit der Lösungen dieser Teilprobleme berücksichtigen. Alle Kriterien sind Heuristiken für gutes Konvergenzverhalten. Die Beschränkung von Iterationsschrittdifferenzen und die Adaption von Datenzugriffsraten und -mustern basierend auf dem Aktualisierungsgrad einfließender Daten machen asynchrone Iterationen effektiver, weil lokale Berechnungen auf "alten" bzw. mehrfach genutzten Vektoren vermieden werden. Diese Form der Selbststeuerung intendiert "synchronisierte" Ausführungen asynchroner Iterationen. Die Adaption der Berechnung an der bereits erzielten Güte der Teillösungen zielt darauf ab, mehr Rechenkapazität für Teilprobleme aufzuwenden, die globale Konvergenz stören, also den "numerischen Flaschenhals" darstellen. Die Auswahlkriterien sind im Modul `CONTROL` implementiert, vgl. Abschnitt 5.6.1.

5.4.4 Steuerung der Kommunikation

Die Semantik des asynchronen Iterierens hat den Vorteil, dass Datenproduzenten und -konsumenten nicht synchronisieren (keine Wartezeiten) und flexibel (selektiv) lokale Iterationen ausführen. Die Aufhebung von Synchronisationspunkten in der Kommunikation hat weitreichen-

de Konsequenzen für die verteilte Berechnung. Datenproduzenten (Sender) können entkoppelt von der Bereitschaft der Datenkonsumenten (Empfänger) Daten propagieren. Dies ist einerseits vorteilhaft, weil ungenutzte Rechenzeiten beim Produzenten und Konsumenten vermieden werden. Andererseits birgt es die Gefahr einer nicht an die Bedarfe der Konsumenten adaptierten Datenpropagation. Wenn die Kommunikationsrate die konsumentenseitige Bedarfsrate übersteigt, müssen entweder viele nichtempfangene Nachrichten gepuffert werden oder nichtempfangene Nachrichten werden ungelesen überschrieben (=ineffiziente Nutzung der Kommunikationsressourcen). Im umgekehrten Fall versucht ein Konsument oft vergeblich Daten zu empfangen und ist gezwungen, die Zeit bis zur Verfügbarkeit mehr oder weniger sinnvoll zu überbrücken. Ein Lösungsansatz ist, den Datenfluss durch einfache Regelkreise zu steuern, indem Konsumenten nach Verwendung kommunizierter Daten einen Aktualisierungsbedarf beim Produzenten anzeigen. Der Konsument steuert somit die Rate, mit der Daten zwischen Produzent und Konsument transferiert werden. Diese einfache Form eines selbststeuernden Regelkreises wird durch 2 Protokolle realisiert.

Definition 5.10 (Demand/Supply Driven Messaging) .
*Das **Demand Driven Messaging** (DDM) Protokoll steuert die Kommunikation aus der Sicht der Datenkonsumenten. Wenn ein Konsument aktualisierte Daten aus "seinem" Kommunikationspuffer liest, signalisiert er Aktualisierungsbedarf und verschickt eine Anforderung an den Produzenten. Bei erneutem Zugriff ohne zwischenzeitliche Aktualisierung wird keine weitere Anforderung initiiert. Das DDM-Protokoll stellt sicher, dass alle kommunizierten Daten in die Iteration einfließen.*
*Das DDM-Protokoll kann durch ein **Supply Driven Messaging** (SDM) sinnvoll ergänzt werden, dessen wesentliche Funktion es ist, den wiederholten Versand duplizierter oder "ähnlicher" Daten zu vermeiden und die Reaktionszeit des Produzenten auf Bedarfanzeigen der Konsumenten (siehe DDM) zu verkürzen. Nach Empfang einer Bedarfsanzeige vom Konsumenten wird überprüft, ob die angeforderten Daten seit der letzten Bedarfsanzeige "ausreichend" modifiziert wurden. Im Fall der Kommunikation der $\mathbf{b}^p_{[c]}$-Vektoren ist die Anzahl der aktualisierten $\boldsymbol{\pi}_{[.]}$-Vektoren, die in $\mathbf{b}^p_{[c]}$ aufsummiert einfließen, ein Grad der Modifikation, vgl. Tab. 5.3 und Gl. 5.4.*

Komponenten des Iterationsvektors, für die eine Bedarfsanzeige anliegt, können in der selektiven Iteration (Abschnitt 5.4.3) höher gewichtet werden. Weil Produzenten auch gleichzeitig Konsumenten sind, kann bei der Reaktion auf Bedarfsanzeigen der Konflikt auftreten, dass die für die Aktualisierung der angefragten Komponente benötigten Daten aus dem Kommunikationspuffer nicht hinreichend aktuell sind.
Die Methodik in *ASYNC* dient der Analyse logistischer Systeme. Umgekehrt weist *ASYNC* gewisse Analogien zu logistischen Systemen auf. Die asynchrone Kommunikation und asynchrone Iteration arbeitet wie ein logistisches System, in dem Daten produziert (Beschaffung), verschickt (Transport) und konsumiert (Produktion) werden und in dem Datenkommunikation durch ein dezentrales Steuersystem mit selbststeuernden Regelkreisen (ähnlich Kanban-Fertigungssteuerung) überwacht wird. Im Gegensatz zu Materialflussystemen, in denen die Regelkreise kettenförmig und Akteure entweder als Produzent oder Konsument agieren, sind hier die Regelkreise in einen zusammenhängenden Graphen eingebettet und alle *Worker*-Prozesse (Akteure) sind gleichzeitig Konsument und Produzent.

5.5 Lastverteilung und Kommunikationsaufwand

Bisher wurde eine Lastverteilung $\mathcal{R}$ (Def. 5.9) als gegeben angenommen. In diesem Abschnitt wird nun die Berechnung der Lastverteilung betrachtet. Eine notwendige Bedingung für gutes Laufzeitverhalten ist eine effektive (optimale) Lastverteilung, in der eingesetzte Rechenressourcen gleichmäßig und das Kommunkationsmedium möglichst gering beansprucht werden.
Eine Lastverteilung ist formal die Zuordnung von Teilproblemen aus $\mathcal{Z}^0$ (vgl. Gl. 5.1) zu den verfügbaren Rechenressourcen aus $\mathcal{P}$. In der durch *ASYNC* realisierten hierarchischen *BJAC*-Iteration ist der Rechenaufwand je Teilproblem durch die Anzahl der Einträge in den Hauptdiagonalblöcken $\mathbf{Q}[c,c]$ determiniert (Notation: $\mathrm{nz}(\mathbf{Q}[c,c])$). Der Aufwand zur Kommunikation eines Vektors $\boldsymbol{\pi}_{[\cdot]}$ ist proportional zu dessen Größe $\dim \boldsymbol{\pi}_{[\cdot]}$. Diese quantitative Bewertung kann durch einen knoten- und kantengewichteten Graphen formal notiert werden. Knoten repräsentieren Teilprobleme und Kanten deren Abhängigkeiten. Bei den Kantengewichten (= Kommunikationsaufwand) ist eine Fallunterscheidung hinsichtlich der Semantik kommunizierter Vektoren notwendig.

Definition 5.11 (Rechenaufwand und -abhängigkeiten) .
Es sei $G = (V, E)$ ein knoten- und kantengewichteter, gerichteter Graph mit

i) $V = \{1, \ldots, N\}$ *und Gewicht* $w(c) = \mathrm{nz}(\mathbf{Q}[c,c])$ *für alle* $c \in V$

ii) $E = \{(r,c) \,|\, \mathbf{Q}[r,c] \neq \mathbf{0}\}$ *mit Gewicht* $w(r,c) = \begin{cases} \dim \boldsymbol{\pi}_{[c]} & \textit{Variante 1, vgl. Tab.5.3} \\ \dim \boldsymbol{\pi}_{[r]} & \textit{Variante 2, vgl. Tab.5.3} \end{cases}$

Dabei ist $\mathbf{Q}[c,c]$ *der c-te Hauptdiagonalblock der Generatormatrix* $\mathbf{Q}$, $\mathrm{nz}(\mathbf{Q}[c,c])$ *die Anzahl der Einträge ungleich von Null in* $\mathbf{Q}[c,c]$ *und* $\dim \boldsymbol{\pi}_{[\cdot]}$ *die Dimension bzw. Größe des Teilvektors* $\boldsymbol{\pi}_{[\cdot]}$.

Eine Partitionierung des Graphen durch Gruppierung von Knoten in Partitionen repräsentiert eine Lastverteilung. Alle Knoten einer Partition werden von einer Rechenressource bearbeitet.
Die Partitionierung von Graphen unter Nebenbedingungen (gleichmäßige Verteilung der Knotengewichte auf Partitionen, Minimierung der Summe der Gewichte von Inter-Partition-Kanten) ist NP-vollständig. In *ASYNC* sind mit *Chaco* 2.0 [68] und *ParMETIS*[17] zwei Tools eingebunden, die eine Vielzahl heuristischer und graphen-theoretischer Partitionierungsansätze für knoten- und kantengewichtete Graphen anbieten. Die Verfahren agieren auf der durch die hierarchische Kronecker-Struktur bereits vorgegebenen Partitionierung. Dadurch ist die Problemgröße von der Große des Zustandsraums n (Größenordnung $10^5 - 10^7$) auf die Anzahl der Makrozustände N (Größenordnung $10 - 10^3$) reduziert. Die vorgegebene Partitionierung stellt bereits eine gute Dekomposition des Berechnungsproblems dar, sie muss hier lediglich vergröbert werden, weil N gewöhnlich die Anzahl der Rechenressourcen übersteigt.
Die Anzahl der Partitionen (= Anzahl der Prozesse) und die Gewichtung der Knotensummen einzelner Partitionen (= Rechenlast je Prozess) sind Aufrufparameter von *Chaco* 2.0 und *ParMETIS*. Weil Chaco 2.0 und *ParMETIS* nur auf ungerichteten Graphen operieren, wird G in einen ungerichteten Graphen $G^* = (V^*, E^*)$ transformiert. Es sei $V^* = V$ und $E^* == \{\{r,c\} \,|\, (r,c) \in E \vee (c,r) \in E\}$ mit $w(\{r,c\}) = \delta_{rc} \cdot w(r,c) + \delta_{cr} \cdot w(c,r)$. Die Verfahren aus *Chaco* 2.0 und *ParMETIS* konstruieren heuristisch Partitionen, in denen die Anzahl der Kanten ($=msg^*$) bzw. die

[17] `http://www-users.cs.umn.edu/ karypis/metis/parmetis/index.html`

Summe der Gewichte von Kanten (=$data^*$) zwischen Partitionen minimiert wird und die Nebenbedingung der Balanziertheit eingehalten wird. msg^* stimmt nicht mit der tatsächlichen Anzahl von Nachrichten msg überein, weil im ungerichteten Graphen zwei reflexiv gerichtete Kanten durch eine einzelne ungerichtete Kante ersetzt sind und weil nicht alle ungerichteten Kanten zwischen Partitionen separate Nachrichten induzieren (Vektoren werden aufsummiert verschickt (Variante 1) oder ein Vektor fließt in die Berechnung mehrerer Teilprobleme eines Konsumenten ein (Variante 2). Ebenso verhält es sich mit $data^*$ und $data$. Die Partitionierung des Graphen bzw. die Lastverteilung schafft eine Datenlokalität, in der nicht jede Datenabhängigkeit auch eine Kommunikation erfordert. Die Techniken aus *Chaco* 2.0 und *ParMETIS* beruhen auf der Annahme, dass Intra-Partition Kanten genau die Datenabhängigkeiten repräsentieren, die keine Kommunikation induzieren. Sie können jedoch nicht berücksichtigen, dass Inter-Partition Kanten zusammengefasst werden können, wenn sie die Kommunikation ein und desselben Vektors $\boldsymbol{\pi}_{[c]}$ repräsentieren.
Wie der tatsächliche Kommunikationsaufwand (Anzahl Nachrichten und Datenvolumen je verteiltem *BJAC*-Schritt) für eine gegebene Lastverteilung $\mathcal{R}$ berechnet wird, ist in der Prop. 5.12 erläutert.

Proposition 5.12 (Ermittlung Kommunikationsaufwand) .
Es sei δ_{rc} eine Indikatorfunktion mit

$$\delta_{rc} \triangleq \begin{cases} 1 & \textit{falls } \mathbf{Q}[r,c] \neq \mathbf{0} \\ 0 & \textit{sonst.} \end{cases}$$

Es sei msg die Anzahl von Nachrichten je verteiltem BJAC-Schritt. Es gilt:

$$msg = \sum_{c \in \mathcal{Z}^0} \sum_{p \in \mathcal{P}} \delta^p_{[c]} \qquad \textit{mit} \qquad \delta^p_{[c]} \triangleq \begin{cases} 0 & \textit{falls } c \in \mathcal{R}(p) \\ 1 - \prod\limits_{r \in \mathcal{R}(p)} (1 - \delta_{rc}) & \textit{sonst.} \end{cases}$$

Die Definition von $\delta^p_{[c]}$ garantiert, dass

$$\delta^p_{[c]} = 1 \iff \mathbf{b}^p_{[c]} \neq \mathbf{0} \wedge c \notin \mathcal{R}(p)$$

gilt, d.h. $\delta^p_{[c]}$ zeigt genau die Datenabhängigkeiten an, die eine Kommunikation induzieren. Die Doppelsumme über der Indexmenge der Teilprobleme $\mathcal{Z}^0$ und der involvierten Rechenprozesse $\mathcal{P}$ erfasst alle potentiell auftretenden Datenabhängigkeiten.
Es sei data das durch einen verteilten BJAC-Schritt generierte Datenvolumen. Es gilt:

$$data = \sum_{c \in \mathcal{Z}^0} \sum_{p \in \mathcal{P}} \delta^p_{[c]} \cdot \dim \boldsymbol{\pi}_{[c]}.$$

Ein Block $\mathbf{Q}[r,c]$ mit $\mathbf{Q}[r,c] \neq \mathbf{0}$, $r \neq c$ und $\mathcal{R}^{-1}(r) \neq \mathcal{R}^{-1}(c)$ induziert eine Kommunikation, deren Größe proportional zu $\dim \boldsymbol{\pi}_{[r]} \cdot \mathbf{Q}[r,c] = \dim \boldsymbol{\pi}_{[c]}$ ist. Deswegen ist im Graphen aus Def. 5.11 das Gewicht einer Kante (r,c), die eine Datenabhängigkeit durch $\mathbf{Q}[r,c]$ repräsentiert, durch $\dim \boldsymbol{\pi}_{[c]}$ quantifiziert.

Abschließend soll die Anwendbarkeit und Qualität der durch *Chaco* 2.0 und *ParMETIS* ermittelten Lastverteilungen untersucht werden. *ParMETIS* realisiert eine verteilte Variante des

Multi-Level-KL Verfahrens, das auch in *Chaco* 2.0 verfügbar ist. Das Multi-Level-KL Verfahren aus *Chaco* 2.0 ist robust, andere Verfahren erzeugen oft Laufzeitfehler oder variieren problemabhängig stark in ihrer Laufzeit. Die Tab. 5.5 dokumentiert die Qualität der Lastverteilungen, die durch *ParMETIS* und *Chaco* 2.0 für das Markov-Ketten Modell der Stückgutumschlaghalle mit den Parameterbelegungen $k = 4 \ldots 7$ berechnet wurden, vgl. Abschnitt 3.4. Die Qualität der Ergebnisse ist nahezu gleich, weil in *ParMETIS* und *Chaco* 2.0 ein Multi-Level-KL Verfahren angewandt wird, wobei auch Abweichungen auftreten, z.B. für $(\mathcal{P} = 6, k = 4)$, $(\mathcal{P} = 6, k = 5)$ und $(\mathcal{P} = 8, k = 5)$. Die Laufzeiten betragen ca. 20 Sekunden (k=4), ca. 100 Sekunden (k=5, *Chaco* 2.0) und ca. 70 Sekunden (k=5,*ParMETIS* mit 2 Rechenprozessen).

	k=4 (n=30 690 750)				k=5 (n=122 189 237)			
	Chaco 2.0		*ParMETIS*		Chaco 2.0		*ParMETIS*	
$\mathcal{P}$	*msg*	*data*	*msg*	*data*	*msg*	*data*	*msg*	*data*
2	68	130.8	67	131.3	132	517.6	132	517.6
4	137	261.8	137	256.6	266	1033.0	271	1032.5
6	185	349.9	164	317.1	356	1367.0	285	1158.9
8	208	407.1	205	398.1	396	1541.1	443	1773.9
10	222	430.3	225	429.2	443	1724.2	424	1717.5

	k=6 (n=423 807 404)				k=7 (n=1 313 059 781)			
	Chaco 2.0		*ParMETIS*		Chaco 2.0		*ParMETIS*	
$\mathcal{P}$	*msg*	*data*	*msg*	*data*	*msg*	*data*	*msg*	*data*
2	181	1439.3	199	1564.2	318	4416.6	297	4431.6
4	316	2612.0	359	2904.8	586	8448.7	511	7789.5
6	[1)]	[1)]	580	4354.8	855	12677.9	1126	15135.4
8	606	4801.9	567	4586.8	969	14408.1	885	13161.3
10	[1)]	[1)]	665	5269.0	1189	17427.7	1202	17260.7

Tabelle 5.5: `SUH1`-Modell: Lastverteilungen für $\mathcal{P}$=Prozesse (k=Modellparameter; *msg*=Anzahl Nachrichten synchroner *BJAC*-Schritt; *data*=Datenvolumen [MB] synchroner *BJAC*-Schritt;)
[1)] *Chaco 2.0* endet fehlerhaft

Die Tab. 5.6 beinhaltet die mit *Chaco* 2.0 erzielte Qualität der Lastverteilungen für das `FMS`-Modell. Im nachfolgenden Bsp. 5.13 wird für das `FMS`-Modell ($k = 10$) gezeigt, wie eine kon-

	k=5 (n=152 712)		k=10 (n=25 397 658)		k=15 (n=724 284 864)	
$\mathcal{P}$	*msg*	*data*	*msg*	*data*	*msg*	*data*
2	6	1.2	**10**	**152.6**	14	3875.0
4	10	1.6	29	409.1	42	11615.1
6	14	1.9	37	483.9	68	17985.1

Tabelle 5.6: `FMS`-Modell: Lastverteilungen für $\mathcal{P}$=Prozesse (k=Modellparameter; *msg*=Anzahl Nachrichten synchroner *BJAC*-Schritt; *data*=Datenvolumen [MB] synchroner *BJAC*-Schritt;)

krete Lastverteilung Kommunikationsabhängigkeiten induziert. Auf dieses Beispiel wird in Abschnitt 6.1.3 zurückgegriffen.

Beispiel 5.13 (FMS-Modell: Datenabhängigkeiten, Lastverteilung, Kommunikation) .
Das FMS-Modell hat für $k = 10$ (Modellparameter) $N = 11$ Makrozustände. Die Abb. 5.8 links zeigt Datenabhängigkeiten (=■), die potentiell bei der verteilten Ausführung von BJAC-Iterationsschritten auftreten. Das Chaco 2.0 Tool berechnet für $\mathcal{P} = \{1, 2\}$ (2 Prozesse) folgende Lastverteilung

$$\mathcal{R}(1) = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \text{ und } \mathcal{R}(2) = \{0, 10\},$$

vgl. Def. 5.9 zur Lastverteilung. Basierend auf der Lastverteilung $\mathcal{R}$ sind in Abb. 5.8 rechts Datenabhängigkeiten in Intra- (⋆) und Inter-Prozess-Abhängigkeiten unterteilt. Letztere induzieren Kommunikation zwischen den Prozessen und sind nummeriert. Wenn die Semantik kommunizierter Daten "Variante 2" ist (Abschnitt 5.4.1), induziert die verteilte Ausführung eines BJAC-Iterationsschritts 10 *Nachrichten ($\boldsymbol{\pi}_{[1]}, \ldots, \boldsymbol{\pi}_{[10]}$). Blöcke mit identischer Nummer induzieren eine einzelne Nachricht. Der mit den 10 Nachrichten verbundene Kommunikationsaufwand ist (8 Byte je `double`-Eintrag):*

$$8Byte \cdot \sum_{i=1}^{10} \dim \boldsymbol{\pi}_{[x]} = 8Byte \cdot 19\,999\,122 = 152.6MB.$$

	0	1	2	3	4	5	6	7	8	9	10		2	1	1	1	1	1	1	1	1	1	2
0	■										■	2	⋆										⋆
1	■	■	■	■	■	■	■	■	■	■	■	1	**1**	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	**1**
2	■	■	■	■	■	■	■	■	■	■	■	1	**2**	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	**2**
3	■		■	■	■	■	■	■	■	■	■	1	**3**		⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	**3**
4	■			■	■	■	■	■	■	■	■	1	**4**			⋆	⋆	⋆	⋆	⋆	⋆	⋆	**4**
5	■				■	■	■	■	■	■	■	1	**5**				⋆	⋆	⋆	⋆	⋆	⋆	**5**
6	■					■	■	■	■	■	■	1	**6**					⋆	⋆	⋆	⋆	⋆	**6**
7	■						■	■	■	■	■	1	**7**						⋆	⋆	⋆	⋆	**7**
8	■							■	■	■	■	1	**8**							⋆	⋆	⋆	**8**
9	■								■	■	■	1	**9**								⋆	⋆	**9**
10	■									■	■	2										**10**	⋆

Abbildung 5.8: Links: Datenabhängigkeiten zwischen Blöcken (■); Lastverteilung mit Intra- (⋆) und Inter-Process-Kommunikation (**1** ... **10** verweisen auf Nachrichten) für das FMS Modell mit $k = 10$

5.6 Implementierungsaspekte

5.6.1 Software-Entwurf

Der C-Quellcode der *ASYNC*-Implementierung ist in 8 Module unterteilt, vgl. Tab. 5.7. Die Aufrufschnittstellen einzelner C-Funktionen sind in [60] dokumentiert.

LOAD Die wesentliche Funktionalität des Moduls ist die Berechnung und Bewertung der initialen Lastverteilung, vgl. Abschnitt 5.5. Das Modul greift auf die externen Tools *Chaco 2.0*, *ParMETIS* und *APNN-Toolbox* zu, siehe unten.

Modul	Funktionalität
LOAD	Berechnung und Bewertung der initialen Lastverteilung
MAIN	Konfiguration der verteilten Applikation und Ergebnisaufbereitung
MASTER	Master-Prozess: Überwachung und Steuerung der verteilten Applikation
WORKER	Worker-Prozess: verteilte Ausführung der numerischen Berechnung
BUF	Puffer-Management für asynchrone Kommunikation
COM	Schnittstelle Kommunikations (Prozess-Konfiguration, Message-Passing, etc)
CONTROL	Überwachung, Steuerung und Protokollierung von Kommunikation und Iteration
AGGSOLVE	Aggregierungs-Disaggregierungsschritte

Tabelle 5.7: Module des Software-Entwurfs im Überblick

MAIN Das Modul beinhaltet das eigentliche Hauptprogramm, das den *Master*-Prozess ansteuert und das Ergebnis der Berechnung - die stationäre Verteilung - für eine spätere Weiterverarbeitung in einer Datei speichert.

MASTER Der *Master*-Prozess erfüllt administrative und koordinierende Funktionen, die eine globale Sicht auf den Zustand der Berechnung und der verteilten Applikation erfordern. Der *Master*-Prozess instanziiert die *Worker*-Prozesse (s.u.) und versendet Steuerparameter an diese. Er erfasst das globale Konvergenzverhalten während der Berechnung, löst 'on-the-fly' Aggregierungsschritte aus und sorgt für eine sichere Terminierung der verteilten Applikation. Das MASTER-Modul benutzt die Module COM und AGGSOLVE und greift auf die *APNN-Toolbox* zu.

WORKER Die Worker-Prozesse implementieren die in Kapitel 4 vorgestellten mathematischen Modelle hierarchischer und asynchroner Iterationen. Das WORKER-Modul benutzt die Module BUFFER, COM und CONTROL und greift auf die *APNN-Toolbox* zu.

BUF Das Modul realisiert eine Schnittstelle für die Verwaltung und den Zugriff auf einen Datenpuffer, der kommunizierte Daten speichert. Die Zugriffsoperationen sind an die Bedürfnisse der verteilten Berechnung angepasst und implementieren "Message-Passing"-Funktionen mit blockierender und nicht-blockierender Semantik. Die Schnittstelle abstrahiert von der konkreten (technischen) Realisation des Puffers und implementiert Varianten, die durch *PVM* bzw. *MPI* unterstützt werden. Die gewünschte Variante wird zur Laufzeit statisch ausgewählt. Im Abschnitt 5.6.3 werden die Varianten im Detail erläutert. Das BUF-Modul benutzt das COM-Modul.

COM Das Modul realisiert eine Schnittstelle für Kommunikationsfunktionen, die auf die speziellen Bedürfnisse von *ASYNC* angepasst ist. Die Schnittstelle besitzt Funktionen zur Instanziierung und Verwaltung von Prozessen und Funktionen zum asynchronen und synchronen Senden und Empfangen *ASYNC*-spezifischer Datenobjekte. Die Schnittstelle abstrahiert vom tatsächlich eingesetzten Kommunikationstool (*PVM* oder *MPI*) und implementiert Funktionen wahlweise (via Compiler-Direktive) generisch durch *PVM*- oder *MPI*-Funktionen. Das zu implementierende Kommunikationstool wird zum Zeitpunkt der Kompilierung ausgewählt. Das COM-Modul benutzt *PVM* oder *MPI*.

CONTROL Das Modul unterstützt die Messung und Protokollierung des Laufzeitverhaltens (Trajektorie der probabilistischen Ausführung der Implementierung, Kommunikationszeiten und

-umfang) und des Konvergenzverhaltens der Iteration (lokale/globale Genauigkeit der Lösung). Protokollierte Daten können für eine ex-post-Analyse in Tabellen und Diagrammen (durch `gnuplot` erzeugt) aufbereitet, in einen LaTex-basierten "Laufzeit-Bericht" integriert und via `SLOG-2`-Traces visualisiert werden. Des Weiteren implementiert das Modul Auswahlkriterien für die selektive asynchrone Iteration.

AGGSOLVE Das Modul spezifiziert einen *Agg-Solver*-Prozess, der "on-the-fly" Aggregierungs- und Disaggregierungsschritte durchführt. Hierfür erhält er von *Worker*-Prozessen nach Aufforderung durch den *Master*-Prozess die Inhalte des aggregierten Systems, löst das aggregierte System und schickt die aggregierte Lösung an den *Master*-Prozess, der wiederum die Lösung selektiv (Steuerung) an die *Worker*-Prozesse verteilt.

5.6.2 Anbindung und Verwendung externer Software

ASYNC greift auf Funktionalitäten externer Software-Werkzeuge zu, vgl. Tab. 5.8.

Tool	Funktionalität	Referenz
APNN-Toolbox	Datenstruktur Generatormatrix	[30]
Chaco 2.0	sequentielle Berechnung Lastverteilung	[68]
ParMETIS	parallele Berechnung Lastverteilung	s.u.
PVM	Kommunikationsroutinen ("Message-Passing")	[40, 51]
MPI	Kommunikationsroutinen ("Message-Passing")	[40, 48]
Jump-Shot	offline Trace-Visualisierung	[37]

Tabelle 5.8: Von *ASYNC* benutzte Funktionalitäten externer Software-Werkzeuge

APNN-Toolbox [18] Das Werkzeug bietet einen Petri-Netz-Editor (GUI), funktionale Techniken zur Petri-Netz-Analyse, zahlreiche Verfahren zur Exploration und Erzeugung des Zustandsraums für zustandsraumbasierte Analysen, die Methodik für kompositionelle und hierarchische Kronecker Datenstrukturen, die der Speicherung der Generatormatrix der Markov-Kette dienen, und ein breites Spektrum numerischer Analyseverfahren für Markov-Ketten. Die Implementierung des Editors geht auf C. Tepper zurück, P. Buchholz und P. Kemper haben die Methodik der Petri-Netz- und Markov-Ketten Analyse implementiert. *ASYNC* ist in die *APNN-Toolbox* eingebunden und von der graphischen Benutzeroberfläche aus aufrufbar, vgl. Abschnitt 7.4.

Chaco 2.0 und ParMETIS Beide Werkzeuge beinhalten Methoden zur Partitionierung knoten- und kantengewichteter Graphen. Derartige Graphen werden hier zur Beschreibung des Berechnungs- und Kommunikationsaufwands herangezogen, vgl. Abschnitt 5.5.
Chaco 2.0 verfügt über ein breites Spektrum sequentieller Methoden. Das Werkzeug ermöglicht es, 7 globale Methoden (verfeinern rekursiv bestehende Partitionen unter Verwendung verschiedener Heuristiken) mit 3 lokalen Methoden (lassen Anzahl der Partitionen invariant, aber versuchen durch veränderte Zuordnungen einzelner Knoten zu Partitionen die Qualität zu verbessern) kombiniert anzuwenden. Mit dem Modul `LOAD` können alle Methoden angesteuert werden. Es

[18] `http://www4.cs.uni-dortmund.de/APNN-TOOLBOX/`

hat sich aber herausgestellt, dass *Chaco 2.0* für große Graphen ($\geq$ 100 Knoten bzw. Makrozustände der Generatormatrix) instabil ist (Laufzeitfehler) und die Laufzeit verfahrens- und problemabhängig stark variiert.
ParMETIS[19] implementiert eine verteilte (*MPI*-basierte) Variante des Multi-Level-KL Verfahrens.

PVM und MPI *PVM*[20] und *MPI*[21] sind Bibliotheken zur Unterstützung paralleler Programmierung, die in Forschung und Praxis weit verbreitet und in Kombination mit verschiedenen Programmiersprachen (Fortran, C, JAVA) anwendbar sind. Die grundlegende Idee ist es, eine hardware- und betriebssystemunabhängige Kommunikationsschnittstelle zu etablieren (sichert Portabilität und Code-Wiederverwertbarkeit), die gebräuchliche Funktionalitäten der parallelen Programmierung offeriert. Dies beinhaltet Instanziierung, Management und Verwaltung von Prozessen, Datenaustausch mit dem "Message-Passing"-Paradigma (expliziter Aufruf von Sende- und Empfangsroutinen mit blockierender und nicht-blockierender Kommunikationssemantik) und verschiedene Synchronisationskonzepte (Broadcast, Barrier, Synchronisation von Prozessgruppen etc).
MPI II (Message-Passing-Interface) ist ein Standard, der durch gegenwärtig verfügbare *MPI*-Distributionen unterstützt wird, die teilweise frei verfügbar sind. Die Implementierungen unterliegen dem Zielkonflikt zwischen hoher Funktionalität und Portabilität. Frei verfügbare Implementierungen wie zum Beispiel die hier verwendete LAM-MPI 7.0.3 Distribution unterstützen ein breites Spektrum von Rechnerarchitekturen, aber nicht die komplette *MPI* II Funktionalität. Dies betrifft zum Beispiel die Kommunikationssemantik der "one-sided communication", bei der ein Datenproduzent in den Puffer des Konsumenten schreiben kann, ohne dass der Konsument an der Kommunikation partizipiert, vgl. Abschnitt 5.6.3. Andererseits versuchen Rechnerhersteller (SUN etc.) durch eigene *MPI*-Distributionen die Verbreitung ihrer Hardware in zukunftsträchtigen Anwendungsbereichen wie die des "Grid-Computing" und "Cluster-Computing" voranzutreiben, indem sie den *MPI* II Standard voll unterstützen, aber natürlich keine Portabilität in konkurrierende Rechnersysteme zulassen[22].

Jump-Shot Das Werkzeug bietet eine leistungsfähige offline-Visualisierung und einfache statistische Auswertungen von `SLOG-2` Traces, vgl. Abschnitt 6.1.

5.6.3 Asynchrone Kommunikation mit Puffer

Um die Abhängigkeit der Datenkonsumenten von der Verfügbarkeit kommunizierter Daten aufzuheben und ein asynchrones Voranschreiten der Berechnung zu ermöglichen, müssen zuletzt empfangene Daten konsumentenseitig gespeichert werden. In *ASYNC* wird eine asynchrone Iteration durch einen (verteilten) Datenpuffer realisiert. Seine Inhalte (=Vektoren) zeigen Datenabhängigkeiten der verteilt ausgeführten Iteration an. Der Speicherplatz des Puffers ist der für die asynchrone Ausführung der Iteration erforderliche Mehraufwand.

[19] `http://www-users.cs.umn.edu/ karypis/metis/parmetis/index.html`
[20] `http://www.csm.ornl.gov/pvm/pvm_home.html`
[21] `http://www.mpi-forum.org`
[22] `http://www.lam-mpi.org/mpi/implementations/fulllist.php` gibt einen guten Überblick zu existierenden MPI-Implementierungen und den jeweiligen Funktionsumfang

Der Puffer wird durch das Modul `BUF` implementiert, vgl. Abschnitt 5.6.1. Die angebotene Schnittstelle abstrahiert von den technischen Details der Puffer-Realisation. Die physikalische Verteilung der Puffereinträge im verteilten Speicher, die technische Realisation des Puffers und die durch Zugriffoperationen (`Write`,`Read`) intern aufgerufenen "Message-Passing"-Funktionen inklusive der Adressierung von Sendern und Empfängern bleiben der aufrufenden Instanz verborgen. Das Modul unterscheidet 4 Varianten für die technische Realisation des Puffers, vgl. Abb. 5.9.

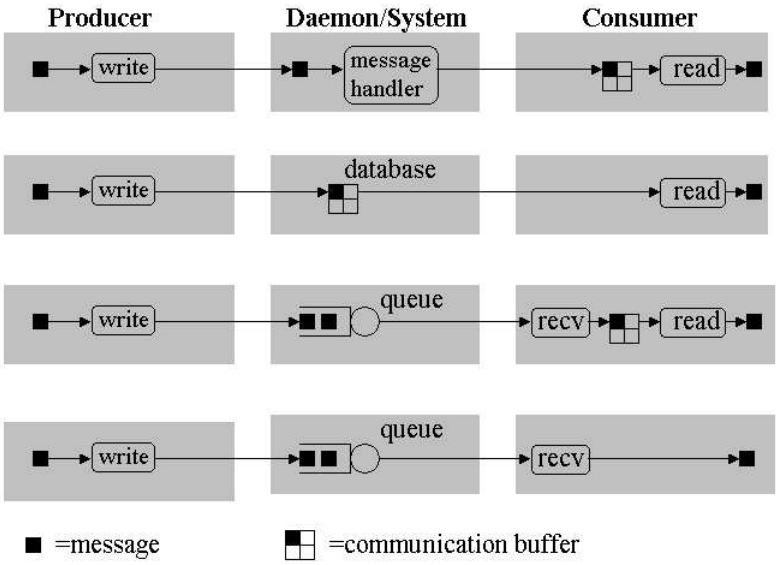

Abbildung 5.9: Kommunikations-Puffer: Varianten der Implementierung

1. **One-sided Communication (Active Messaging, Message Handler)**: Der Produzent überschreibt nicht-blockierend ausgewählte Inhalte des Datenpuffers beim Konsumenten, ohne dass der Konsument an der Kommunikation partizipiert. Dies ist vorteilhaft in Situationen, in denen der Konsument nicht ständig die Verfügbarkeit neuer Daten prüfen kann und somit die Gefahr zahlreicher hängender Nachrichten (=gesendet, aber nicht empfangen) besteht. Die Schwierigkeit ist, Dateninkonsistenz durch Vermeidung von Zugriffskonflikten zu gewährleisten. Zugriffskonflikte können prinzipiell auf Hardwareebene (Schaltnetze), Betriebssystemebene und Applikationsebene gelöst werden. In *PVM* wird der Konflikt pragmatisch gelöst. Eine ankommende Nachricht triggert einen "Message Handler". Dieser unterbricht die Ausführung des Programms nicht und wird erst abgearbeitet, wenn das Programm einen *PVM*-Befehl aufruft und somit die Kontrolle beim *PVM*-System liegt. Dieses Konzept hat sich in *ASYNC* sehr bewährt und wird vorrangig angewendet. Die Wartezeiten der "Message Handler" sind gering, weil im C-Quellcode zahlreiche *PVM* Funktionsaufrufe plaziert sind. Der *MPI* II Standard unterstützt verschiedene Semantiken der "one-sided communication" bzw. des "remote memory access", die Zugriffskonflikte durch unterschiedlich starke Synchronisationsmechanismen lösen. Die

am wenigsten synchronisierte Variante ("passive target communication") ist im Modul `BUF` realisiert. Leider ist in der LAM-*MPI* Distribution die "passive target communication" via `MPI_WIN_LOCK` und `MPI_WIN_UNLOCK` nicht implementiert, so dass die *MPI* II Ausprägung dieser Variante nicht getestet werden konnte.

2. **Datenbank in Daemon-Prozessen**: Der Datenpuffer ist als Datenbank im *Daemon*-Prozess (Abb. 5.5 rechts) des Konsumenten realisiert (wird ab *PVM* Version 3.4 unterstützt). Der Vorteil des Ansatzes liegt in der Trennung von *Worker*-Prozess (Berechnung) und *Daemon*-Prozess (Puffer-Management). Nachteilig sind wenig performante Zugriffszeiten, weil alle Daten durch "Message-Passing" zum Konsumenten transferiert werden müssen (verteilter Speicher). *MPI* II unterstützt dieses Konzept nicht.

3. **Message-Passing mit Kommunikations-Puffer**: Die Variante bildet das "Message-Passing"-Paradigma mit nicht-blockierenden Sende- und Empfangsroutinen unverändert ab und ergänzt es durch einen konsumentenseitigen Datenpuffer. Wenn ein Konsument Daten aus dem Datenpuffer liest, wird darin eingebettet überprüft, ob "hängende" Nachrichten anliegen und ob Puffereinträge durch diese aktualisiert werden können.

   Der Nachteil dieser Variante ist, dass Konsumenten aktiv an der Kommunikation partizipieren müssen und deswegen die Gefahr einer Ansammlung zahlreicher hängender Nachrichten besteht. Die Länge der Warteschlange hängender Nachrichten ist bedingt durch das unabhängige Agieren der Produzenten und Konsumenten prinzipiell unbeschränkt. In der Praxis ist dies nachteilig und führt zu hoher Speicherplatzanforderung in den *Daemon*-Prozessen. *PVM* und *MPI* unterstützen diesen Modus.

4. **Message-Passing ohne Kommunikations-Puffer**: Diese Variante bildet ebenfalls das "Message-Passing"-Paradigma ab, allerdings ohne Kommunikationspuffer. Konsumenten rufen blockierende Empfangsroutinen auf, d.h. lokale Berechnungen schreiten synchronisiert mit kommunizierten Daten voran und alle Datenabhängigkeiten sind potentielle Synchronisationspunkte. *PVM* und *MPI* unterstützen diesen Modus.

Kapitel 6

Messung und Modellierung der Performance von *ASYNC*

Dieses Kapitel fokussiert auf die Messung und Modellierung der Performance von *ASYNC*. Performance ist hier als Lösungszeit bzw. Ausführungszeit von *ASYNC* definiert und wird durch die Anzahl notwendiger Iterationsschritte und die Zeit der einzelnen Iterationsschritte bestimmt. Ein Messung der Lösungszeit ist prinzipiell einfach. Um aber die Ursachen für schwankende oder ggf. schlechte Lösungszeiten zu finden, ist es notwendig, einen Einblick in den dynamischen Ablauf der Berechnung und Kommunikation in *ASYNC* zu erhalten. Die Visualisierung von Rechen- und Kommunikationsmustern kann hierbei sehr hilfreich sein.
Der Abschnitt 6.1 führt in die Performance-Messung und -Visualisierung basierend auf Trace-Files ein. Anschließend werden Performance-Engpässe durch Messungen auf Netzwerkebene identifiziert. Zusätzlich wird der Einfluss der Asynchronität auf die Lösungszeit anhand einer Messung auf der Ebene des *ASYNC*-Source-Codes bewertet. Danach wird die Qualität der parallelen Berechnung mit etablierten Maßen (Beschleunigung bzw. Effizienz) evaluiert. Anhand sehr großer Markov-Ketten-Modelle werden die Grenzen des *ASYNC*-Instrumentariums mit den zur Verfügung stehenden Rechen- und Kommunikationsressourcen aufgezeigt.
Der Abschnitt 6.2 behandelt die Performance-Modellierung von *ASYNC* unter Berücksichtigung von stochastischen Modellen für die zeitliche Bewertung synchroner und asynchroner Iterationsschritte im Zusammenspiel mit bekannten Konvergenzrate-Resultaten. Hierfür werden Varianten von synchronen und schwach-asynchronen Iterationen als System interpretiert, korrespondierende stochastische Modelle abgeleitet und ihre analytische Behandelbarkeit untersucht.

6.1 Performance Messung und Auswertung

Beim Testen paralleler Programme müssen oft Ursachen funktionalen Fehlverhaltens (z.B. Prozesse "hängen") und schlechter Performance (z.B. kein linearer Speed-up) gefunden werden. Eine Analyse des Programmablaufs, die auf einer Visualisierung oder statistischen Auswertung eingetretener Ereignisse und Zustände basiert, kann hierbei sehr hilfreich sein. Im Vergleich zu sequentiellen Programmen ist es für parallele Programme oft schwierig, Verhalten und Performance a priori zu verstehen, wenn die Folge auftretender Ereignisse im Programmablauf indeterministisch ist. Der Indeterminismus geht hauptsächlich darauf zurück, dass der inhärent stochastische Charakter der Rechen- und Kommunikationsressourcen auf die Ausführung der Implementierung "durchschlägt". Dieser Effekt wird durch die Aufhebung von Synchronisationspunkten in Kommunikation und Berechnung weiter verstärkt.
Die messungsbasierte Analyse paralleler Programme beinhaltet 2 Schritte: 1. Messung interessierender Ereignisse und Speicherung im Trace-File 2. Auswertung (visuell und/oder statistisch) der im Trace-File aufgezeichneten Ereignisse und Zustände.
Trace-Files werden während der Programmausführung erzeugt. Sie speichern beobachtete Ereignisse mit Zeitstempeln ab. Die Herausforderung hierbei ist eine zeit- und platzeffiziente Aufzeichnung der Ereignisse. Dann ist gewährleistet, dass die eigentliche Ausführung des gemessenen "Systems" (Programm-Source-Code, Kommunikation) unbeeinflusst bleibt und große Datenmengen gespeichert und in der Auswertung effizient aufbereitet werden können. Des Weiteren sollten die Trace-Spezifikationssprachen portierbar, skalierbar, (dynamisch) erweiterbar und leicht erzeugbar (benutzerfreundliche Code-Instrumentation) sein. Prominente Vertreter von Trace-Spezifikationssprachen, die diesen Anforderungen gerecht werden, sind `SLOG-2` [37] und das Pablo `Self-Defining Data Format (SDDF)` [87].
Trace-Files können "online" zur Laufzeit oder "offline" visuell oder statistisch ausgewertet werden. Eine online-Analyse ist gewöhnlich Bestandteil eines Steuerungsmechanismus in Form eines einfachen Regelkreises oder eines intelligenteren (z.B. zustandsraumbasierten) Entscheiders, der z.B. ein dynamisches Ressourcen-Management realisiert. Eine statistische Auswertung ermöglicht automatisierte online-Entscheider, demgegenüber ist eine visuelle Auswertung für menschliche online-Entscheider hilfreich. *Autopilot-Virtue* [97] ist eine Toolkombination, die eine visuelle und statistische Auswertung (basierend auf `SDDF`) online (Schwerpunkt) und offline unterstützt. *Autopilot-Virtue* ist Bestandteil des *Pablo*-Tools [87]. *XPVM* ist ein Tool zur online- und offline-Visualisierung von Funktionen des Message-Passing Tools *PVM* (vgl. Abschnitt 5.6.2) basierend auf `SDDF` Traces. *Jump-Shot* [37] bietet eine leistungsfähige offline-Visualisierung von `SLOG-2` Traces, die u.a. mit der *MPICH-MPI* Distribution generierbar sind. Darüber hinaus gibt es weitere Werkzeuge, die auf Programm-, Rechen- und Kommunikationsebene ein offline- bzw. online-"Performance Monitoring" mit benutzerdefinierten Metriken (wenige auch mit adaptiven Steuerungen) unterstützen.

6.1.1 Netzbelastung

Erste Experimente mit *ASYNC* haben gezeigt, dass eine ungesteuerte Kommunikation (kein DDM, vgl. Def. 5.10) bei der Lösung von Markov-Ketten Modellen mit wenigen Millionen Zuständen in Rechnernetzen mit 10- und 100-MBit Verbindungen (mittlerer Teil in Abb. 5.4) Engpässe und Instabilitäten erzeugt [55]. Eine typische Beobachtung sind hohe Speicherplatzanforderungen der Daemon-Prozesse (Abb. 5.5 rechts), die vermutlich durch eine hohe Anzahl

"hängender Nachrichten" (=gesendet, aber nicht empfangen) verursacht werden. Generell war die Performance und die Effektivität der verteilten Berechnung in *ASYNC* oft nicht zufriedenstellend, wobei Engpässe nicht analysierbar und nicht in ihrem Enstehen beobachtet werden konnten.
In Zusammenarbeit mit J. Mäter und einer studentischen Projektgruppe zur "Entwicklung eines Java-basierten Frameworks für verteilte Netzwerk-Messungen" [81] wurden agentenbasierte Messungen der durch *ASYNC* verursachten Netzbelastung durchgeführt. Die Testumgebung war ein *Cisco Catalyst 2900* Switch und 4 *SunBlade 100 Rechner* (2 GB Arbeitsspeicher, *UltraSparc IIe* Prozessor, 500 MHz), wobei 3 Rechner mit einem 100 MBit/s Links und der 4. Rechner über ein 10 MBit/s Uplink-Port verbunden waren. Die Testumgebung ist ein Ausschnitt des mittleren Teils des lokalen Netzwerks in Abb. 5.4. Das zu lösende Markov-Ketten Modell hatte 10 773 430 Zustände und 126×126 Blöcke und ein synchroner *BJAC*-Iterationsschritt induzierte einen Kommunikationsaufwand von 92 Nachrichten mit insgesamt 64.9 MB Datenumfang. Das Ziel des Experiments war zu prüfen, ob für das beschriebene Berechnungsproblem die Bandbreite des Kommunikationsmediums ein Engpass ist. Hierfür wurde der Netzverkehr am Switch gemessen und die Anzahl übertragener Bits je Zeitintervall (= 10 Sekunden) protokolliert.

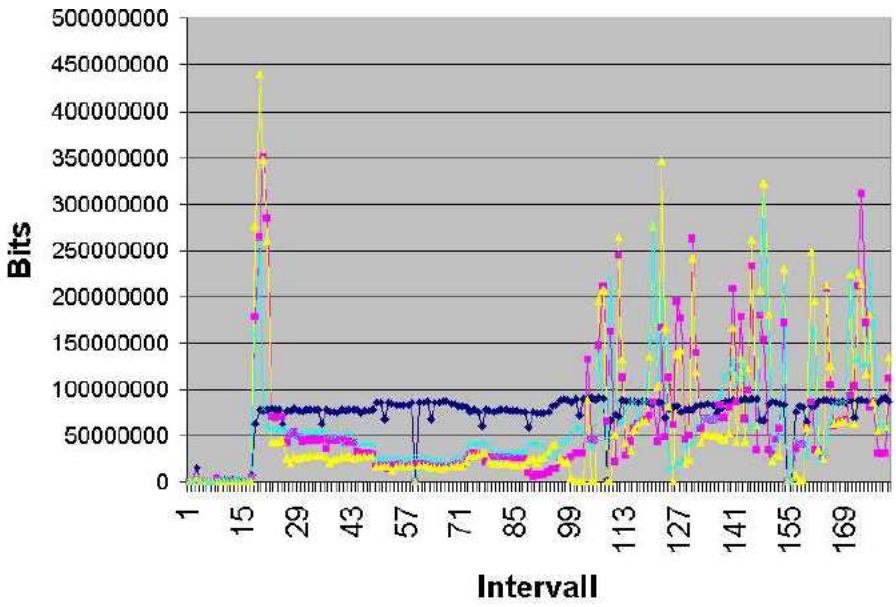

Abbildung 6.1: Datendurchsatz gemessen an allen Ports

Die Abb. 6.1 zeigt den Durchsatz für alle Ports. Es ist zu erkennen, dass die Netzbelastung anfänglich sehr hoch ist (Spitze), dann folgt von Intervall 29 bis 99 (=700 Sekunden) eine geringe Belastung und danach folgt ein stochastisches Muster schwankender Belastung, das sich über das Zeitfenster in Abb. 6.1 hinaus fortsetzt. Diese Beobachtungen sind leicht erklärbar. Am Anfang führt *ASYNC* einen synchronen *BJAC*-Schritt aus, in dem alle Daten ("All-To-All" Kommunikation) zeitnah verschickt werden. Hierdurch wird das Kommunikationsmedium kurzzeitig hoch beansprucht. Die zeitliche Überlappung von Rechen- und Kommunikationsphasen, die im späteren Verlauf der Berechnung bedingt durch die Asynchronität eintritt, ist für die Beanspruchung des Kommunikationsmediums vorteilhaft. In Abb. 6.1 ist zu erkennen, dass

die Spitzen in der Beanspruchung deutlich geringer sind als am Anfang. Die anfänglich geringe Belastung des Mediums von Intervall 29 bis 99 ist dadurch zu erklären, dass durch die initiale Datenpropagation zunächst nur geringer Bedarf an neuen zu kommunizierenden Daten signalisiert wird (DDM und SDM sind aktiviert, vgl. Def. 5.10). Die Separierung von Kommunikations- und Rechenphasen, die am Anfang beobachtbar ist, wird durch die Asynchronität zunehmend verwischt.
Des Weiteren ist zu erkennen, dass die Belastung des 10 MBit Uplink-Port (blaue bzw. dunkelste Linie in Abb. 6.1 bzw. Abb. 6.2) nicht schwankt. Der gemessene Durchsatz von nahezu 100 000 000 Bit je 10 Sekunden ist permanent nahe der Grenz-Kapazität von 10 MBit je Sekunde.

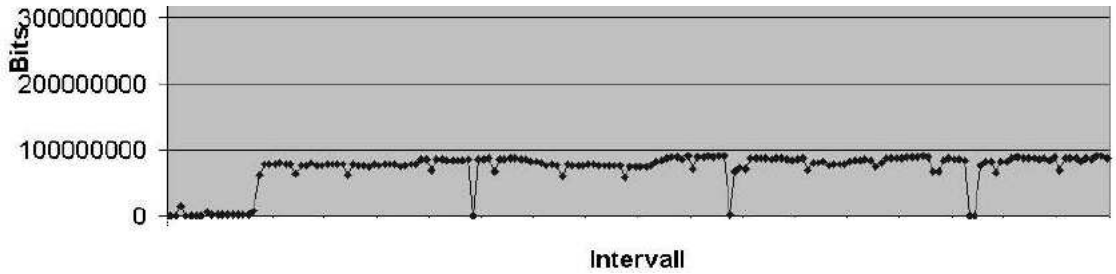

Abbildung 6.2: Datendurchsatz gemessen am kritischen 10 MBit Port

Fazit: Messungen der Netzbelastung haben gezeigt, dass bereits für Problemgrößen mit etwa 10 Millionen Zuständen und langsamen Rechnern (Kommunikationsrate gering) 10 MBit/s Links nicht ausreichend sind. Messungen der Netzbelastung geben einen gewissen Einblick in das Verhalten von *ASYNC* auf unterster Ebene. Weil die Enstehung des Kommunikationsmusters eng an die softwaretechnische Realisation der Kommunikation durch *PVM* und *MPI* gekoppelt ist und ebenfalls sehr vom Verhalten des Programms abhängt, deren Zustände hier nicht erfasst werden, kann über die Ursachen des beobachteten Verhaltens auf Netzebene zunächst keine Aussage gemacht werden

6.1.2 Messung in *ASYNC*

Im nächsten Schritt wurden Messungen oberhalb der Netzwerkebene in *PVM* und *MPI* durchgeführt. *PVM* und *MPI* unterstützen die Erzeugung von Trace-Files in verschiedenen Formaten. Eine visuelle online-Auswertung muss große Datenmengen in graphische Objekte übersetzen und anzeigen. Der Umgang mit *XPVM* hat gezeigt, dass *XPVM* zur online-Visualisierung durch *ASYNC* erzeugter Traces (`SDDF`) nicht ausreichend performant ist. *XPVM* kann aber gut zur offline- Animation/Analyse des Ablaufs verwendet werden. Messungen in *PVM* und *MPI* erfassen - wie auch die Messungen auf der Ebene der Netzwerkprotokolle oben - nicht das Zusammenspiel von Programmausführung (Berechnung) und Kommunikation, dessen Analyse bedeutsam ist. Dies soll nachfolgend erläutert werden.

1. **Pufferzugriff und -aktualisierung:** In *ASYNC* kann der Zugriff auf gepufferte (kommunizierte) Daten von der Kommunikation entkoppelt werden, d.h. Pufferinhalte können mehrfach ohne zwischenzeitliche Aktualisierung gelesen werden (vermindert Effizienz der

Berechnung) oder aber ungelesene Pufferinhalte können überschrieben werden (vermindert Effizienz der Kommunikation). Deswegen ist es für das Verständnis des Ablaufs der Berechnung wichtig, neben der Kommunikation auch das Zugriffsmuster auf den Puffer, das eng mit dem Ablauf der Berechnung verbunden ist, auswerten zu können.

2. **Interaktion Berechnung und Kommunikation:** In *ASYNC* kann die ablauflogische Reihenfolge (asynchroner) Sendebefehle indeterministisch sein, wenn Daten zustandsabhängig versendet werden. Mögliche Vorbedingungen für einen Sendebefehl ist das Vorliegen einer Bedarfsanzeige seitens eines Konsumenten (vgl. "Demand-Driven-Messaging" Abschnitt 5.4) oder eine hinreichende Veränderung der Daten seit dem letzten Versand (vgl. "Supply-Driven-Messaging" Abschnitt 5.4). Deswegen ist die Entstehung von Kommunikationsmustern (wer, wann, was kommuniziert) nur zusammen mit dem Zustand der Berechnung nachvollziehbar und verifizierbar. Umgekehrt kann die ablauflogische Reihenfolge bearbeiteter Teilprobleme indeterministisch sein, wenn Teilprobleme zustandsabhängig gewählt werden. Eine Vorbedingung kann der Aktualisierungsgrad der Pufferinhalte sein (siehe erster Punkt), von der die Berechnung des Teilproblems abhängt.

3. **Synchronisationspunkte:** *ASYNC* bietet eine flexible Experimentierumgebung, in der zahlreiche Implementierungsvarianten leicht getestet werden können. Ein Unterscheidungsmerkmal ist der Grad der Asynchronität. So ist es beispielsweise möglich, zustandsabhängige Synchronisationspunkte zu aktivieren. Eine mögliche Aktivierungsbedingung ist der Aktualisierungsgrad zu lesender Pufferinhalte. Deswegen ist es für das Verständnis beobachteter Synchronisationsphasen wiederum wichtig, auch den Zustand der Berechnung auswerten zu können.

4. **Defizite existierender Trace-Generatoren:** Auf der Ebene von *PVM* und *MPI* tragen Nachrichten lediglich ein vordefiniertes Attribut - ein sogenanntes "message-tag". Existierende Trace-Generatoren zeichnen Nachrichten mit dem Wert des "message-tag" auf. In *ASYNC* werden weitere Attribute benutzt, die Bestandteil des benutzerdefinierten Datentyps der zu versendenden Daten sind (z.B. der Iterationsschritt, in dem Daten berechnet wurden). Existierende Trace-Generatoren bieten nicht die Möglichkeit, über das vordefinierte "message-tag" hinaus weitere zu protokollierende Attribute zu definieren.

Das Fazit ist, dass ein Kommunikations-Trace allein für die Auswertung von *ASYNC* nicht ausreichend informativ ist. Deswegen ist in *ASYNC* eine eigenständige Trace-Generierung implementiert, die sowohl Ereignisse der Berechnung, als auch der Kommunikation erfasst. Die Trace-Generierung in *ASYNC* ist mit Präprozessor-Direktiven zum Zeitpunkt der Kompilierung (also nicht zur Laufzeit) ein- oder ausschaltbar. Zur Laufzeit erzeugt jeder Prozess ein Trace-File mit einer selbstdefinierten Trace-Spezifikation. Nach Beendigung des Programms liest ein awk-Skript die lokalen Trace-Files und integriert die Informationen in einer Trace-Datenstruktur. Dabei werden logische Abhängigkeiten zwischen lokal protokollierten Ereignissen erkannt. Zum Beispiel wird ein Sende-Ereignis dem zugehörigen Empfangsereignis zugeordnet. Danach nutzt das awk-Skript die eingelesenen Daten für Statistiken und erzeugt zum Zweck der Trace-Visualisierung einen textuellen `SLOG-2`-Trace. Die `SDDF`-Spezifikationssprache bietet zwar mehr Möglichkeiten und Flexibilität, es hat sich aber gezeigt, dass diese hier nicht benötigt werden. Des Weiteren war die Installation und Handhabung des Java-basierten `SLOG-2` Viewers *Jump-Shot* einfacher als die des `SDDF`-Pendants *Pablo*.

6.1.3 Asynchronität und Lösungszeit

Um den Einfluss der Asynchronität auf die Lösungszeit zu bewerten, werden 4 Implementierungsvarianten $V = 1 \ldots 4$ mit zunehmender Asynchronität getestet. Die Varianten 1 bis 3 haben bedingte Synchronisationspunkte, vgl. Tab. 5.4. Die vierte Variante realisiert eine vollständig asynchrone Iteration ohne Synchronisationspunkte. Visuelle und statistische Auswertungen der durch *ASYNC* erzeugten Traces geben Einblicke in das dynamische Laufzeitverhalten der Varianten.

Jede Implementierungsvariante löst mit 2 *Worker*-Prozessen die Markov-Kette des `FMS`-Modells (Modellparameter $k = 10$) für eine vorgegebene Genauigkeitsschranke ($\|\epsilon\|_\infty < 10^{-6}$, vgl. Gl. 4.10). Für die Interpretation des beobachteten Laufzeitverhaltens ist es sehr hilfreich, die Lastverteilung für 2 Prozesse und die damit verbundenen Datenabhängigkeiten des `FMS`-Modells zu kennen. Die Lastverteilung und die Datenabhängigkeiten wurden im Bsp. 5.13 vorgestellt.

Die Abb. 6.4-6.6 zeigen visualisierte Traces der ersten drei Implementierungsvarianten. Die dargestellten Ausschnitte entstammen dem *Jump-Shot* Viewer. Violette Rechtecke (dunkelster Grauton) repräsentieren Synchronisationsphasen (S-Phasen) an datenabhängigen Synchronisationspunkten, grüne Rechtecke (hellster Grauton) kommunikationsabhängige Rechenphasen (*KAR*) und blaue Rechtecke (mittlerer Grauton) kommunikationsunabhängige Rechenphasen (*KUR*). Pfeile zeigen Nachrichten mit ihrem Sendezeitpunkt und dem Zeitpunkt der Verfügbarkeit im Puffer des Empfängers an. Die Abb. 6.3 verdeutlicht die eingesetzten Symbole für *KAR*s, *KUR*s und Nachrichten in beispielhaften Szenarien der 3 Implementierungsvarianten. Zusätzlich ist für jede *KAR* und *KUR* der Iterationsschritt notiert.

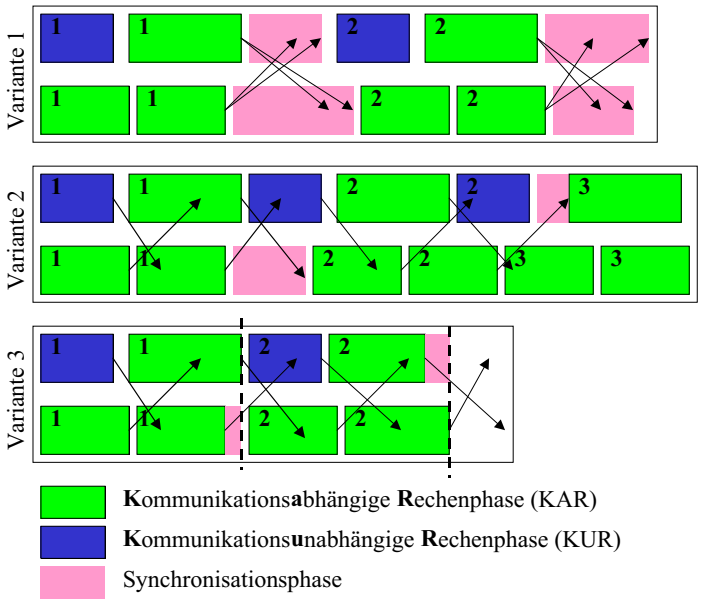

Abbildung 6.3: Visualisierung Rechen- und Kommunikationsmuster

In Tab. 6.1 sind Lösungszeiten und Auswertungsergebnisse für V=1 . . . 4 zusammengetragen. Nachfolgend werden die Ergebnisse genauer betrachtet.

V	Zeit_{USR}	Zeit_{CPU}		Iter		Zeit_{USR}/Iter		Zeit_{CPU}/Iter		Comp [%]		Sync [%]	
		P=1	P=2	P=1	P=2	P=1	P=2	P=1	P=2	P=1	P=2	P=1	P=2
1	6194	6189	6192	149	149	41.6	41.6	41.5	41.6	71	74	14	19
	6343	6335	6340	149	149	42.6	42.6	42.5	42.6	70	71	16	22
2'	5475	5473	5472	149	149	36.7	36.7	36.7	36.7	79	91	4	0
	5394	5390	5392	149	149	36.2	36.2	36.2	36.2	82	85	0	6
2	5655	5650	5647	148	148	38.2	38.2	38.2	38.2	76	92	7	0
	5459	5448	5457	148	148	36.9	36.9	36.8	36.9	83	82	0	9
3	5395	5232	5393	150	150	40.0	40.0	34.9	40.0	79	91	3	0
	5700	5692	5190	150	150	38.0	38.0	37.9	34.6	82	82	0	9
	5830	5819	5603	150	150	38.9	38.9	38.8	34.6	82	87	0	4
4	5148	5144	5144	135	155	38.1	33.2	38.1	33.2	83	90	0	0
	5721	5715	5708	148	150	38.7	38.1	38.6	38.1	74	91	0	0
	5551	5545	5543	160	145	34.7	38.3	34.7	38.2	82	91	0	0
	5309	5297	5307	133	155	39.9	34.3	39.8	34.2	83	90	0	0

Tabelle 6.1: Lösungszeit (Sekunden) vs. Asynchronität: V=Grad der Asynchronität; Zeit=Lösungszeit (wall-clock); Iter=Anzahl Iterationen bis Genauigkeitsschranke; Comp=Anteil Dauer Rechenphasen für Teilprobleme; Sync=Anteil Wartezeiten in Synchronisationspunkten

Variante 1 Die Ausführung der Implementierungsvariante 1 ist durch zwei strikt alternierende Phasen α und β gekennzeichnet. Die α-Phase ist eine Rechenphase, in der zugeordnete Blöcke gelöst werden. Die β-Phase ist eine Kommunikations- und S-Phase. Die β-Phase endet bzw. die nächste α-Phase beginnt, sobald alle am Anfang der aktuellen β-Phase initiierten Nachrichten im Puffer der Empfänger vorliegen, vgl. Abb. 6.4. Insofern ist die β-Phase eine S-Phase, deren Dauer vom Zustand des Kommunikationspuffers abhängt. Die so realisierte Iteration mischt stochastisch Jacobi- und Gauss-Seidel-Schritte, weil kommunizierte und dann in die Berechnung einfließende Vektoren aus dem vorherigen oder aktuellen Iterationsschritt entstammen. In der Iteration in Abb. 6.4 treten nur Jacobi-Schritte auf. Der Synchronisationsaufwand und die Lösungszeiten der Variante 1 sind sehr hoch, vgl. Tab. 6.1. Zwischen 14% und 22% der Gesamtzeit geht in S-Phasen verloren, vgl. Spalte "Sync".

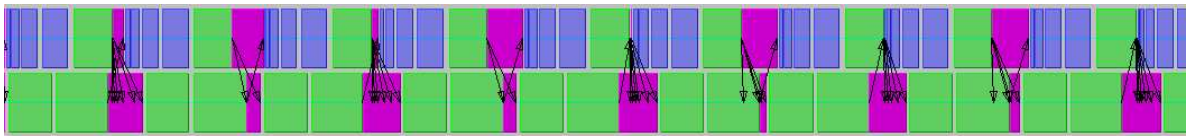

Abbildung 6.4: Trace-Visualisierung der *ASYNC*-Implementierungsvariante 1

Variante 2 Der wesentliche Unterschied zu Variante 1 ist, dass die Vektoren sofort nach ihrer Aktualisierung verschickt werden (nicht bei Variante 2') und Synchronisationspunkte für kommunizierte Vektoren erst unmittelbar vor deren Benutzung positioniert sind. Insofern sind die oben definierten α- und β-Phasen nun überlappend. In Variante 1 wurden zuerst alle Vektoren

aktualisiert (α-Phase) und danach verschickt (Pfeile starten alle am gleichen Punkt) und die β-Phase endet, wenn alle kommunizierten Vektoren vorliegen sind. Dies ist nachteilig, weil die Beanspruchung der Kommunikationsressourcen punktuell hoch ist und die Synchronisation mit kommunizierten Vektoren ggf. nicht zeitnah zur tatsächlichen Benutzung ist.
In Variante 2' wird der Zugriff auf kommunizierte Vektoren zeitnah synchronisiert, die "Bulk"-Kommunikation aus Variante 1 bleibt aber bestehen. Die Abb. 6.5 oben zeigt zwei Traces der Variante 2'. Prozess 1 hat 8 *KUR*s (für $\boldsymbol{\pi}_{[1]}$ bis $\boldsymbol{\pi}_{[8]}$) gefolgt von einer *KAR* (für $\boldsymbol{\pi}_{[9]}$). Im oberen Trace ist zu erkennen, dass Prozess 1 (oben) mit der α-Phase starten kann, obwohl noch eine Nachricht unterwegs ist. Die Nachricht trifft von α-Phase zu α-Phase zunehmend später ein und nach einigen Schritten muss Prozess 1 vor der *KAR* synchronisieren. Prozess 2 hat am Ende der ersten α-Phase eine lange S-Phase (α-Phase hat 2 *KAR*s). Diese wirft Prozess 2 im Vergleich zu Prozess 1 hinreichend weit zurück, so dass alle folgenden Nachrichten von Prozess 1 zu Prozess 2 rechtzeitig eintreffen. Der obere Trace zeigt, dass sich nach einigen Iterationsschritten ein Zustand einstellt, der durch S-Phasen in Prozess 1 gekennzeichnet ist und stabil bleibt (auch über den dargestellten Bereich hinaus). Im zweiten Trace der Variante 2' stellt sich ein anderer stabiler Zustand ein, in dem S-Phasen nur in Prozess 2 auftreten. Prozess 2 hat am Ende der ersten α-Phase eine lange S-Phase. Im Gegensatz zum ersten Experiment wirft diese den Prozess 2 nicht soweit zurück, so dass alle weiteren Nachrichten von Prozess 1 zu Prozess 2 rechtzeitig ankommen (S-Phasen in Prozess 2 bleiben) und die Nachricht von Prozess 2 zu Prozess 1 ist rechtzeitig verfügbar (keine S-Phasen in Prozess 1).
In Abb. 6.5 ist in den beiden unteren Traces der Variante 2 zu erkennen, dass Pfeile nun keinen gemeinsamen Startpunkt haben. Die Beseitigung der Bulk-Kommunikationen verbessert die Performance (Zeit_{USR}, Zeit_{CPU}) leider nicht (vgl. Tab. 6.1), weil Performance-kritische Kommunikation weiterhin eine beschleunigte Ausführung verhindert. Was genau "Performance-kritisch" ist hängt vom a-priori nicht bekannten Berechnungs- und Kommunikationsmuster ab. Im ersten Trace treten zunächst keine S-Phasen, dann nur im Prozess 1 auf. Im zweiten Trace ist es umgekehrt, der Prozess 2 schreitet zu schnell voran und wird durch S-Phasen ausgebremst. Die Kommunikation des Vektors $\boldsymbol{\pi}_{[10]}$ von Prozess 2 an Prozess 1 bzw. des Vektors $\boldsymbol{\pi}_{[9]}$ von Prozess 1 an Prozess 2 ist im ersten bzw. zweiten Trace kritisch, weil diese Vektoren innerhalb der Iterationsschritte zuletzt berechnet und verschickt werden. Es ist unklar, warum trotz gleichbleibender Lastverteilung einmal Prozess 1 und dann Prozess 2 zu schnell ist und ausgebremst wird. Vorliegende aber hier nicht angeführte Statistiken zeigen, dass CPU-Zeiten einzelner *KAR*s und *KUR*s sowohl innerhalb eines Laufs, als auch deren Mittelwerte von Lauf zu Lauf stark (bis zu 10%) schwanken.
Der Anteil der Synchronisationsphasen sinkt im Vergleich zu Variante 1 deutlich und liegt nun zwischen 4% und 9%, vgl. Tab. 6.1.
Die Dynamik in der zeitlichen Ausführung bewirkt (noch) keine indeterministische Numerik, d.h. die Anzahl notwendiger Schritte ist stets 149 (Variante 2') und 148 (Variante 2), vgl. Tab. 6.1. Die S-Phasen verhindern eine dynamische Rekombination in *KAR*s einfließender kommunizierter Vektoren, d.h. einfließende Vektoren entstammen stets in gleicher Weise dem aktuellen oder vorherigen Iterationsschritt. Damit ist die Folge der berechneten Iterationsvektoren deterministisch.

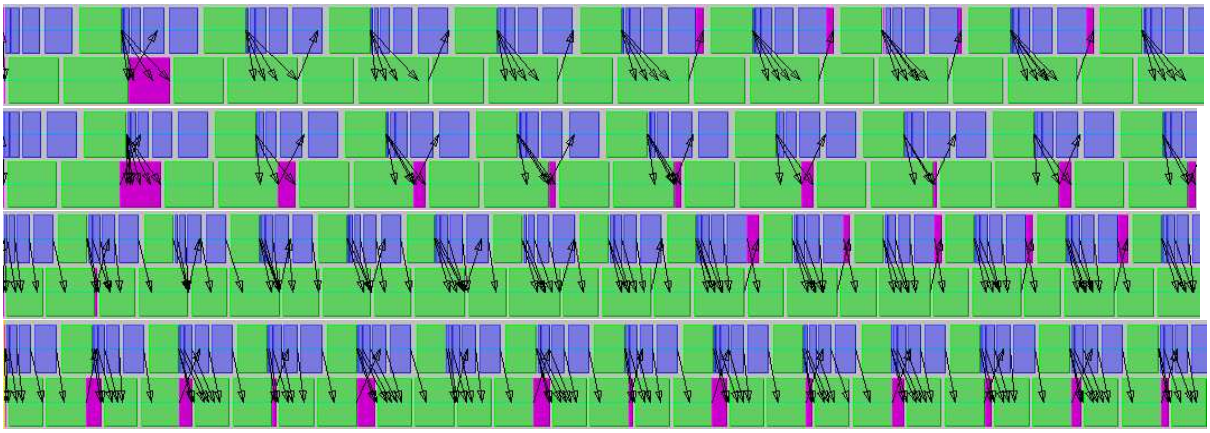

Abbildung 6.5: Trace-Visualisierungen der *ASYNC*-Implementierungsvariante 2' (1. und 2. Trace von oben) und 2 (3. und 4. Trace von oben)

Variante 3 In den Varianten 1 und 2 synchronisiert die Berechnung mit der Aktualität kommunizierter Daten und alle *KAR*s und *KUR*s müssen mit gleicher Rate ausgeführt werden. In Variante 3 werden *KAR*s und *KUR*s ebenfalls mit gleicher Rate ausgeführt, es sind aber nun Synchronisationspunkte (Barrieren) in den Source-Code integriert und kommunizierte Daten können beliebig alt sein.
Die Abb. 6.6 zeigt, dass Prozess 2 (unten im oberen Trace) schneller ist und durch S-Phasen ausgebremst wird. Die α-Phasen starten prozessübergreifend zeitlich synchron. Des Weiteren zeigt Abb. 6.6, dass die Bandbreite des Kommunikationsmediums (1 GBit Switch) ausreicht, um Vektoren ohne große Verzögerungen zu transferieren. Die ausbleibende Dynamik in den Übertragungszeiten führt wie oben zu einer deterministischen numerischen Ausführung, die Anzahl notwendiger Schritte ist stets gleich 150, vgl. Spalte "Iter" in Tab. 6.1.
Die Performance der Variante 3 ist im Vergleich zu Variante 2 etwas schlechter, vgl. Spalten "Zeit$_{USR}$" und "Zeit$_{CPU}$" in Tab. 6.1. Für Prozesse mit S-Phasen gilt Zeit$_{USR}$ > Zeit$_{CPU}$, weil durch die Art der S-Phasen Implementierung keine CPU-Zeit verbraucht wird (bei Variante 1 und Variante 2 findet in S-Phasen ein "Polling" neuer Nachrichten statt, das CPU-Zeit verbraucht). Trotz gleichbleibendem numerischen Aufwand und S-Phasen ohne Einfluss auf die CPU-Zeit schwanken die CPU-Zeiten einzelner Prozesse recht stark. Die Ursache für die Schwankungen ist unklar. Um den Grund für die Schwankungen einzugrenzen, wurden Messungen an dedizierten Source-Code-Blöcken durchgeführt. Dabei zeigte sich, dass Vektor-Matrix-Multiplikationen, die zahlreiche Zeigeroperationen und natürlich arithmetische Operationen beinhalten, schwankende CPU-Zeit Verbräuche verursachen.
Die Barriere-Synchronisation hat bei erhöhtem Kommunikationsaufwand eine schlechte Performance. Die Abb. 6.6 zeigt unten einen Trace mit 4 *Worker*-Prozessen, wobei jeweils 2 Prozesse auf einem Cluster laufen und die beiden Cluster über das langsame Lehrstuhl-Netzwerk verbunden sind, vgl. Abb. 5.4. Der Kommunikationsaufwand steigt von 10 Nachrichten mit 153 MB auf 29 Nachrichten mit 409 MB je Iterationsschritt, vgl. Tab. 5.6. Die Abb. 6.6 zeigt, dass in allen Prozessen lange S-Phasen auftreten. Der Anteil der S-Phasen liegt zwischen 32% (2. Prozess von unten) und 70% (oberster Prozess). Wiederholungen dieser und ähnlicher Experimente bestätigten diese Beobachtung. Auffällig ist, dass keine Nachrichten über S-Phasen hinweg auftreten. Die Implementierung der Barriere-Synchronisation löst ein "Message-Passing" aus, mit

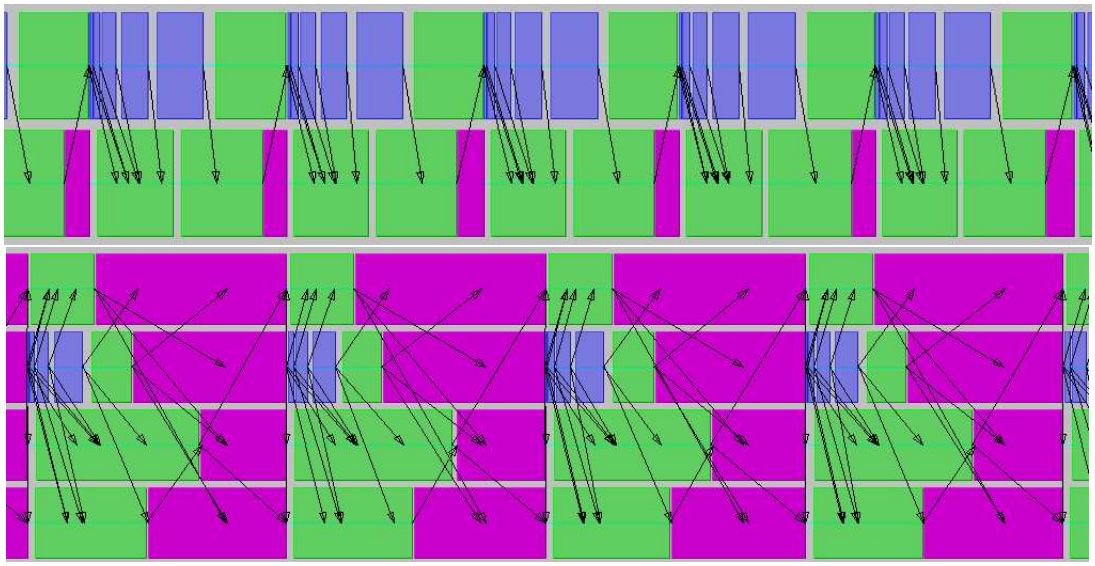

Abbildung 6.6: Trace-Visualisierung der *ASYNC*-Implementierungsvariante 3 mit 2 Prozessen (oben) und 4 Prozessen (unten)

dem Prozesse das Erreichen einer Barriere signalisieren. Im hier verwendeten Kommunikations-Tool *PVM* scheinen diese Signale keine höhere Priorität zu erhalten als die Kommunikation, die durch Sende- und Empfangsbefehle ausgelöst wird. Dies heisst die Performance der Barriere-Synchronisation (=kürzeste Prozess-S-Phase) hängt von der aktuellen Last der sonstigen Kommunikation ab. Im Experiment mit 4 Prozessen kann dies wie folgt quantifiziert werden: die erste Barriere-Synchronisation am Start der Iteration (noch keine konkurrierende Kommunikation) benötigt 0.925 ms, die folgenden Barriere-Synchronisationen benötigen im Mittel 10 s. Die ersten Einzelwerte lauten 9.91 s, 10.19 s, 9.93 s, 10.31 s ...

Variante 4 Diese Variante enthält keine Synchronisationspunkte. Die Folge der Iterationsvektoren ist indeterministisch und die Anzahl notwendiger Iterationsschritte bis zum Erreichen der Genauigkeitsschranke variiert, vgl. Tab. 6.1. Der durch Asynchronität ggf. erforderliche numerische Mehraufwand in Form zusätzlicher Iterationsschritte ist im Beispiel sehr moderat bzw. tritt nicht auf. Die probabilistische Anzahl der Schritte weicht nur unwesentlich von der Anzahl in den Varianten 1 bis 3 ab. Im nachfolgenden Abschnitt 6.1.5 wird ein Beispiel betrachtet, in dem Asynchronität einen numerischen Mehraufwand auslöst. Auf der anderen Seite kann die Asynchronität keinen signifikanten Performance-Gewinn im Vergleich zu Variante 2 erzielen, weil der Synchronisations-Mehraufwand dort gering ist. Die mittlere USER-Zeit (Zeit_{USR}) über alle 4 Läufe ist 5496 s (Variante 2) und 5432 s (Variante 4).

Fazit Eine a-posteriori visuelle und statistische Auswertung von *ASYNC*-Traces verdeutlicht Rechen- und Kommunikationsmuster und gestattet auf diese Weise Einblicke in die indeterministischen *ASYNC*-Abläufe. Implementierungsvarianten mit schwacher Asynchronität erreichen - mit Ausnahme der stark synchronisierten Variante 1 - die selbe Performance wie eine vollständig asynchrone Implementierung unter der Bedingung, dass die Lastverteilung gut balanziert ist.

Diese Bedingung ist hier erfüllt und deswegen sind Blockierungen in bedingten Synchronisationspunkten kurz. Schwache Asynchronität verhindert hier ein "Durchschlagen" des inhärent stochastischen Charakters der Rechen- und Kommunikationsressourcen auf die Numerik und so ist die Anzahl auszuführender Iterationsschritte deterministisch. In den beobachteten *ASYNC*-Abläufen stellen sich nach einigen Iterationsschritten stationäre Rechen- und Kommunikationsmuster ein. Trotz gleichbleibendem Experimentierumfeld ist das Muster nicht reproduzierbar, Synchronisationsphasen treten zum Beispiel in unterschiedlichen Prozessen auf.
Doppelprozessoren sind für *ASYNC* sehr vorteilhaft, weil *Worker*- und *Daemon*-Prozesse nicht um Rechenressourcen konkurrieren.

6.1.4 Totale vs. partielle Asynchronität

In den Gl. 4.25 und 4.27 (Seite 36) wurden 2 Konstanten D und S definiert, die als Gradmesser der Asynchronität interpretierbar sind. Die D-Konstante legt grob gesprochen fest, wie alt - gemessen in Iterationsschrittdifferenzen - kommunizierte Vektoren sein dürfen, um beim Konsumenten verwendet werden zu dürfen. *ASYNC* kann die D-Konstante kontrollieren, in dem ggf. schnelle *Worker*-Prozesse solange "ausgebremst" werden bis hinreichend aktuelle Daten vorliegen.
Nachfolgend soll der Einfluss der D-Konstante auf Lösungszeiten experimentell untersucht werden. Hierfür werden die Markov-Ketten des `SUH2`-Modells mit dem Parameter k=5 (67 920 112 Zustände) und des `FMS`-Modells mit dem Parameter k=10 (25 397 658 Zustände) für Genauigkeitsschranken $\|\boldsymbol{\epsilon}\|_\infty < 10^{-9}$ bzw. $\|\boldsymbol{\epsilon}\|_\infty < 10^{-6}$ (vgl. Gl. 4.10) mit 7 bzw. 2 *Worker*-Prozessen gelöst.
In Tab. 6.2 sind alle Messwerte aufgelistet. Die Spalte "Zeit" verweist auf die USER-Lösungszeit und die Spalte "NA" gibt den numerischen Aufwand gemessen in der Summe der Iterationsschritte aller beteiligter *Worker*-Prozesse an. Fett gedruckte Werte sind Mittelwerte über jeweils 3 Experimente.

1. **Numerischer Aufwand vs. Asynchronität :** Eine Schlussfolgerung aus der Theorie zum Konvergenzverhalten asynchroner Iterationen ist, dass mit Zunahme der Asynchronität die Effektivität der Iterationen sinken kann, vgl. Abschnitt 5.2.3. Das `SUH2`-Modell zeigt dieses Verhalten. Mit Zunahme von D, insbesondere im total asynchronen Szenario ($D = \infty$), steigt die Gesamtzahl der Schritte, vgl. Spalte "NA" in Tab. 6.2. Demgegenüber ist beim `FMS`-Modell der numerische Aufwand wenig sensitiv bzgl. D, im total-asynchronen Szenario sinkt der numerische Aufwand sogar.

2. **Deterministischer numerischer Aufwand :** Für alle partiell-asynchronen Abläufe (D endlich) ist der numerische Aufwand sehr deterministisch, vgl. "Iterationen"-Spalten in Tab. 6.2. Dies ist insofern erstaunlich, als dass sich die Iteration bis zum Erreichen der D-Schranke wie eine total asynchrone Iteration verhält, für die der numerische Aufwand schwankt (siehe $D = \infty$ Experimente). Die transiente stochastische Beeinflussung des numerischen Aufwands hat geringen Einfluss auf den Gesamtaufwand.

3. **Lösungszeit vs. Asynchronität :** In den Experimenten ist die Lösungszeit wenig sensitiv gegenüber der Wahl von D. Die total-asynchrone Iteration erzielt trotz Wegfall potentieller Synchronisationspunkte keine Vorteile, was durch einen erhöhten numerischen Aufwand (`SUH2`-Modell) oder eine geringe Sensitivität bzgl. der Asynchronität (`FMS`-Modell) bedingt ist.

D	SUH2-Modell									FMS-Modell			
	Iterationen							Zeit	NA	Iterationen		Zeit	NA
	p=1	p=2	p=3	p=4	p=5	p=6	p=7	[s]	[iter]	p=1	p=2	[s]	[iter]
1	210	209	210	210	210	210	210	3235	1469	144	145	2934	289
1	210	209	210	210	210	210	210	3169	1469	144	145	2945	289
1	210	209	210	210	210	210	210	3157	1469	144	145	2939	289
								3187	**1469**			**2939**	**289**
5	211	206	211	211	211	211	211	3117	1472	141	146	3094	287
5	210	205	210	210	210	210	210	3120	1465	141	146	3066	287
5	212	207	212	212	212	212	212	3160	1479	141	146	3064	287
								3132	**1472**			**3075**	**287**
10	214	204	214	214	211	212	214	3096	1483	138	148	2787	286
10	214	204	214	214	211	214	214	3097	1485	139	149	2831	288
10	214	204	214	214	212	214	214	3103	1486	138	148	2815	286
								3099	**1485**			**2811**	**286**
15	216	201	216	216	208	211	216	3033	1484	136	151	2938	287
15	216	201	216	216	208	210	216	3036	1483	136	151	2955	287
15	216	201	216	216	209	209	216	3053	1483	136	151	2904	287
								3041	**1483**			**2932**	**287**
∞	231	203	248	360	175	178	233	3227	1628	126	150	3031	276
∞	236	204	249	371	177	180	238	3261	1655	130	150	3128	280
∞	225	185	223	347	194	199	220	3312	1593	126	151	3076	277
								3267	**1625**			**3078**	**278**

Tabelle 6.2: Lösungszeit und numerischer Aufwand (NA) vs. Grad der Asynchronität gemessen durch die Konstante D

Fazit Aus den vorliegenden Experimen-Ergebnissen läßt sich keine Empfehlung für die Wahl der D-Konstante ableiten. In den dokumentierten (und nicht-dokumentierten) Experimenten erzielt eine total asynchrone Iteration keine signifikanten Performance-Vorteile gegenüber einer partiell asynchronen Iteration, obwohl in allen Experimenten das *DDM*-Protokoll (Abschnitt 5.4.4) aktiviert ist (Kommunikation nur bei Bedarf). Ohne *DDM*-Protokoll werden in total asynchronen Iterationen Performance-Verschlechterungen durch schnell iterierende *Worker*-Prozesse riskiert, die mit hoher Rate und über Bedarf Vektoren propagieren.

6.1.5 Effizienz der verteilten Berechnung

Die Beschleunigung und die Effizienz sind etablierte Maße zur Bewertung der Qualität paralleler Berechnungen.

Definition 6.1 (Beschleunigung und Effizienz) *Es sei $T(1)$ die Laufzeit einer 1-Prozess-Lösung und $T(p)$ mit $p > 1$ die Laufzeit einer verteilten p-Prozess-Lösung. Dann ist*

$$S(p) \triangleq \frac{T(1)}{T(p)}$$

die Beschleunigung (speed-up) und

$$E(p) \triangleq \frac{S(p)}{p}$$

die Effizienz der verteilten/parallelen Lösung des Berechnungsproblems. Des Weiteren sei $T_i^{cpu}(p)$ *mit* $i \leq p$ *die CPU-Zeit von Prozess* i *innerhalb einer* p*-Prozess-Lösung.*

Dieser Abschnitt dokumentiert *ASYNC*-Laufzeitexperimente zur Messung von Beschleunigung und Effizienz. In Tab. 6.3 sind Lösungszeiten $T(p)$, Beschleunigung $S(p)$ und Effizienz $E(p)$ in Abhängigkeit von der Anzahl verteilt ablaufender *Worker*-Prozesse p angegeben. Fett gedruckte Werte sind Mittelwerte über Experimente mit gleichem p-Wert. Des Weiteren ist in Tab. 6.3 die Anzahl ausgeführter Iterationsschritte je Prozess p (= numerischer Aufwand) aufgeführt.
Gelöst wird das `SUH1`-Modell mit dem Parameter k=5 (= 122 189 237 Zustände, vgl. Abschnitt 3.4) für eine feste Genauigkeitsschranke ($\|\epsilon\|_\infty < 10^{-7}$, vgl. Gl. 4.10). Hierfür wird ein Rechner-Cluster mit 6 Rechnern benutzt, vgl. Abschnitt 5.3, insb. Abb. 5.4 rechts. Die Größe des `SUH1`-Modells ist nahe am Platzlimit einer 1-Prozess-Lösung. Der virtuelle Arbeitsspeicher pro Prozess ist unter dem Linux-Betriebssystem unabhängig von der Größe des Arbeitsspeichers (hier 6 GB) gegenwärtig auf 3 GB beschränkt[1], ein einzelner *ASYNC*-Prozess benötigt etwa 2.2 GB. Das nächst größere `SUH1`-Modell mit $k = 6$ kann wegen des Platzlimits nicht durch einen einzelnen *ASYNC*-Prozess gelöst werden und ist deswegen zur Bewertung der Beschleunigung ungeeignet, vgl. Abschnitt 6.1.6 zur Bewertung des Platzaufwands sehr großer Modelle.

p	$T(p)$	$\sum_{i=1}^{p} T_i^{cpu}(p)$	Iter						$S(p)$	$E(p)$
	[s]	[s]	p=1	p=2	p=3	p=4	p=5	p=6		
1	50 140	50 135	197							
2	**34 064**	**68 102**							**1.47**	**0.74**
	34 263	68 516	237	320					1.46	0.73
	33 925	67 791	237	314					1.48	0.74
	34 004	68 000	236	314					1.47	0.74
4	**19 205**	**76 702**							**2.61**	**0.65**
	19 415	77 520	230	301	292	336			2.58	0.65
	18 931	75 633	223	300	300	384			2.65	0.66
	19 268	76 953	220	309	300	342			2.60	0.65
6	**14 254**	**85 363**							**3.52**	**0.59**
	14 805	88 698	227	364	270	326	317	445	3.39	0.56
	13 809	82 683	238	337	265	311	311	386	3.63	0.61
	14 147	84 710	235	348	265	326	293	414	3.54	0.59

Tabelle 6.3: Beschleunigung und Effizienz der verteilten numerischen Analyse mit *ASYNC*

1. **CPU-Zeit vs. USER-Zeit :** Bei allen *Worker*-Prozessen erreicht die CPU-Zeit nahezu die USER-Zeit ($T(p) \approx p^{-1} \cdot \sum_{i=1}^{p} T_i^{cpu}(p)$), weil *Worker*-Prozess und *Daemon*-Prozess wegen der Doppelprozessoren nicht um Rechenressourcen konkurrieren. Praktisch bedeutet dies, dass der durch den *Daemon*-Prozess geleistete Mehraufwand zur Abwicklung der Kommunikation keinen Einfluss auf die USER-Lösungszeit hat und so *ASYNC* sehr von den Doppelprozessoren profitiert. In früheren Messungen auf Singleprozessoren wur-

[1] Anfrage bei "Informatik-Rechner-Betriebsgruppe", Fachbereichs Informatik, Uni Dortmund

de festgestellt, dass *Daemon*-Prozesse 10%-20% der insgesamt aufzubringenden CPU-Zeit beanspruchen [52].

2. **Stochastik :** Schwankungen der Beschleunigung und in der Anzahl ausgeführter Iterationsschritte (= numerischer Aufwand) verdeutlichen den dynamischen Charakter der asynchronen Iteration und Kommunikation. Die beobachte Schwankungsintensität (= Verhältnis langsamste zu schnellste Lösungzeit $T(p)$) beträgt 1.001 ($p = 2$) 1.026 ($p = 4$) 1.072 ($p = 6$). Eine Zunahme von p bewirkt mehr Kommunikationsabhängigkeiten und damit mehr stochastische Einflussgrößen.

3. **Beschleunigung und Effizienz :** *ASYNC* erzielt keine lineare Beschleunigung, weil die Asynchronität in der verteilten Berechnung einen signifikanten numerischen Mehraufwand erzeugt. Der durch die Kommunikation verursachte Mehraufwand wird hauptsächlich durch *Daemon*-Prozesse geleistet. Weil die *Daemon*-Prozesse hier über eigene Rechenressourcen verfügen, hat die von ihnen konsumierte CPU-Zeit keinen Einfluss auf die USER-Zeiten der *Worker*-Prozesse und damit auch keinen Einfluss auf die gemessene Beschleunigung[2]. Dass die Asynchronität für einen erhöhten, in Iterationsschritten gemessenen numerischen Aufwand verantwortlich ist, kann man gut für $p = 1$ und $p = 2$ sehen. Die Erhöhung des CPU-Zeit Aufwands von 50 135 für $p = 1$ auf 68 102 (+35.8%) für $p = 2$ wird durch die Erhöhung des numerischen Aufwands von c.a. 40% ausgelöst. Die mittlere Anzahl von Iterationsschritten über alle Prozesse steigt von 197 für $p = 1$ auf 277 für $p = 2$. Die Erhöhung des CPU-Zeit Aufwands von 50 135 für $p = 1$ auf 76 702 (+53%) für $p = 4$ und 85 363 (+70.3%) für $p = 6$ wird hauptsächlich - aber nicht ausschließlich - durch den numerischen Mehraufwand (49.6% bzw. 60.1% - Schätzung auf Iterationsschritten, s.o.) erzeugt. Obwohl der vermehrte Kommunikationsaufwand hauptsächlich durch die *Daemon*-Prozesse getragen wird, verursacht er auch in den *Worker*-Prozessen eine Erhöhung der CPU-Zeit.

Nachfolgend wird das `FMS`-Modell mit dem Parameter k=11 (54 682 992 Zustände) für eine feste Genauigkeitsschranke ($\|\boldsymbol{\epsilon}\|_\infty < 10^{-9}$, vgl. Gl. 4.10) gelöst und experimentell untersucht, welche Beschleunigung *ASYNC* im Vergleich zur konventionellen *SOR*-Iteration erzielt. Des Weiteren wird der Einfluss von Aggregierungs-/Disaggregierungsschritten (AD-Schritte) betrachtet. Die Ergebnisse sind in Tab. 6.4 notiert.

1. **Einfluss AD-Schritte :** Die Einbettung von AD-Schritten in die *SOR*-Iteration führt zu einer deutlichen Reduktion der Lösungszeit (Exp. 1+2), weil sich die Anzahl auszuführender Iterationsschritte von 1 020 auf 170 vermindert. Demgegenüber bleiben AD-Schritte in der synchronen *BSOR*-Iteration mit 4 inneren Schritten je Block nahezu ohne Wirkung (Exp. 3+4, entspricht *ASYNC* mit p=1). Ebenso verhält es sich mit *ASYNC* und p=5 (Exp. 5+6).

2. ***ASYNC*-Beschleunigung gegenüber *SOR*-Iteration :** Die sequentielle *BSOR*-Iteration in Exp. 3 ist deutlich schneller als die *SOR*-Iteration in Exp. 2. Allerdings kann die Lösungszeit von 8 964 Sekunden der *SOR*+AD-Iteration durch keine *ASYNC*-Variante unterboten werden. Sogar die Lösungszeit der 5-Prozess-Lösung ist mit 10 309 Sekunden (ohne AD) bzw. 10 090 Sekunden (mit AD) deutlich höher.

[2] Ansonsten wären die Werte für die Beschleunigung schlechter als die hier angegebenen.

Exp	Verfahren	$T(p)$ [s]	$\sum_{i=1}^{p} T_i^{cpu}(p)$ [s]	Iter
1	*SOR*	50 880	50 880	1 020
2	*SOR*+AD	8 964	8 964	170
3	*ASYNC* p=1 (=*BSOR*)	27 980	27 989	384
4	*ASYNC*+AD p=1 (=*BSOR*+AD)	27 419	27 424	367
5	*ASYNC* p=5	10 309	51 799	321-480
6	*ASYNC*+AD p=5	10 090	50 650	339-461

Tabelle 6.4: Performance: Methodenvergleich und Einfluss von AD-Schritten

Fazit Die Konvergenzrate synchroner und asynchroner Block-Iterationen ist im betrachteten Beispiel (`FMS`-Modell) wenig sensitiv gegenüber eingebetteten AD-Schritten. Aus diesem Grund kann die durch AD-Schritte deutlich verminderte Lösungszeit einer *SOR*+AD-Iteration durch keine *ASYNC*-Variante (mit/ohne AD, sequentielle/verteilte Lösung) erreicht werden.

6.1.6 Lösung sehr großer Modelle

Die Lösung sehr großer Markov-Ketten Modelle erfordert leistungsfähige Rechen-, Platz- und Kommunikationsressourcen. *ASYNC* ermöglicht Modelle mit mehreren 100 Millionen Zuständen verteilt zu lösen. Mit Ausnahme der Arbeit von Bell [17] ist bisher nicht über die Lösung derartig großer Modelle berichtet worden.
Bell löst ebenfalls große Markov-Ketten verteilt und iterativ auf einem Rechner-Cluster (verteilter Speicher), die Realisierung unterscheidet sich jedoch in mehreren Punkten von *ASYNC*. Bell betrachtet eine konventionelle Jacobi-Fixpunkt-Iteration und ein klassisches projektives Verfahren (hier: hierarchische und asynchrone Erweiterung einer *BJAC*-Iteration) und bei Bell sind die Lösungsverfahren Disk-basiert implementiert, d.h. die Generatormatrix wird vollständig im Festplattenspeicher gehalten und wird bei Bedarf nur partiell in den Arbeitsspeicher geladen (hier: alle Daten inklusive der Generatormatrix können im Arbeitsspeicher gehalten werden).
Das größte behandelbare Modell in [17] ist das `FMS`-Modell mit k=15 (724 Millionen Zustände). Die Lösung mit einer geringen Genauigkeitsschranke erfordert 292 Stunden (eine "übliche" Genauigkeitsschranke wird nicht erreicht). In der vorliegenden Arbeit konnte ein etwas größeres Modell mit 897 Millionen Zuständen in 55 Stunden gelöst werden, vgl. Tab. 6.5. Das `FMS`-Modell ermöglicht einen Performance-Vergleich. Leider kann *ASYNC* das `FMS`-Modell mit k=13 . . . 15 nicht lösen, weil die gegebene Blockstruktur der Generatormatrix sehr imbalanziert ist, insbesondere einen sehr großen Block aufweist, der 20% (k=13), 19% (k=14) bzw. 18% (k=15) der Gesamtgröße ausmacht. Das `FMS`-Modell mit k=12 wird in [17] in c.a. 79 Stunden gelöst (übliche Genauigkeitsschranke, Jacobi-Iteration, 26 Prozesse). *ASYNC* benötigt für das gleiche Modell c.a. 8 Stunden (übliche Genauigkeitsschranke, 6 *Worker*-Prozesse, vgl. Tab. 6.5). Beim Performance-Vergleich ist allerdings zu beachten, dass unterschiedliche Abbruchkriterien angewendet werden (Bell normiert den Residualvektor mit der Maximum-Norm des Iterationsvektors, hier wird der Residualvektor mit der 1-Norm des Iterationsvektors normiert) und dass die Laufzeitumgebung unterschiedlich ist. Bell benutzt ein Rechner-Cluster mit deutlich mehr Einzelrechnern (26 Dualprozessoren, hier: 6 Dualprozessoren), dafür haben seine Einzelrechner eine geringere Rechenleistung und weniger Arbeitsspeicher und die maximal verfügbare Band-

breite des Kommunikationsmediums ist geringer (100 Mbit/s vs. 1Gbit/s).
Nachfolgend wird aufgezeigt, wo mit den hier zur Verfügung stehenden Rechen- und Platzressourcen Engpässe bei der *ASYNC*-Anwendung auftreten.

Zeit-Engpass Die Lösung des `SUH1`-Modells (k=6) mit 6 *Worker*-Prozessen benötigt für eine Genauigkeitsschranke von $\|\epsilon\|_\infty < 10^{-7}$ (vgl. Gl. 4.10) 44 900 Sekunden bzw. 12, 5 Stunden, vgl. Tab. 6.5. Die Beantwortung der Frage, ob derartige Lösungszeiten akzeptabel sind oder nicht, wird stets auch subjektiv beeinflusst sein. Eine Lösungszeit von 12, 5 Stunden ist akzeptabel, wenn eine Berechnung ausserhalb der gewöhnlichen Arbeitszeiten, in denen Rechen-, Platz und Kommunikationsressourcen weniger beansprucht sind, ausgeführt werden kann. Bei einer Akzeptanzbetrachtung ist des Weiteren zu berücksichtigen, dass mit einer Genauigkeitsschranke von 10^{-7} ein vergleichsweise schwaches Abbruchkriterium gesetzt ist. Legt man den üblichen Richtwert von 10^{-9} als Schranke zugrunde, gelangt man für das `SUH1`-Modell bereits zu Lösungszeiten von knapp 20 Stunden.
Die Lösung des `SUH2`-Modells (k=7) mit 6 *Worker*-Prozessen benötigt für eine Genauigkeitsschranke von $\|\epsilon\|_\infty < 10^{-9}$ 63 560 Sekunden bzw. 17, 7 Stunden. Der numerische Aufwand gemessen in der Anzahl der Iterationsschritte ist trotz verschärfter Genauigkeitsschranke geringer als der numerische Aufwand für das `SUH1`-Modell, vgl. Tab. 6.5. Deswegen ist die Lösungszeit für das `SUH2`-Modell trotz größerem Zustandsraum nahezu unverändert.
Das größte gelöste Modell in dieser Arbeit ist das `SUH3`-Modell mit 896 958 393 Zuständen, vgl. Tab. 6.5.

Modell	n	p	$\|\epsilon\|_\infty$	$T(p)$	$\sum_{i=1}^{p} T_i^{cpu}(p)$
`SUH1` k=6	423 807 404	6	10^{-7}	12:28:20	74:40:01
`SUH2` k=7	588 759 872	6	10^{-9}	17:39:20	105:43:30
`SUH3` k=7	896 958 393	14	10^{-9}	55:27:00	608:29:26
`FMS` k=12	111 414 940	6	10^{-9}	8:15:32	

Tabelle 6.5: Lösungszeiten (USER und CPU) für sehr große Modelle

Die Lösung des `SUH3`-Modells mit 14 *Worker*-Prozessen benötigt für eine übliche Genauigkeitsschranke gut 55 Stunden. Im Unterschied zu allen anderen Experimenten, die in dieser Arbeit dokumentiert sind, war es hier erforderlich, die Anzahl innerer Iterationsschritte (vgl. Abschnitt 4.3) von 4 auf 6 heraufzusetzen. Für 4 innere Schritte stagnierte die hierarchische *ASYNC*-Iteration bei geringer Genauigkeit.
Insgesamt werden während der Lösung 1 167 228 Vektoren mit einem Datenvolumen von 8.4496 TB (Tera Bytes) transferiert, vgl. Tab. 6.6. Dies ergibt eine durchschnittliche Größe von 7.59 MB je Vektor. Die durchschnittliche Größe kann unabhängig von der Lastverteilung über die mittlere Blockgröße in der Generatormatrix (hier Dimension =896 958 393/792 = 1 132 523) geschätzt werden. Die Schätzung liefert 8.64 MB je Vektor (jeder `double`-Eintrag benötigt 8 Bytes).

Platz-Engpass In Tab. 6.7 ist der durchschnittliche Platzaufwand je Prozess (Spalte TOTAL) für das `SUH1`-Modell mit den Parametern k=6 und k=7 angegeben. Die Haupttreiber des Platzaufwands in den Prozessen sind der anteilige Iterationsvektor (IT-VEK), die anteiligen Elemente

i	*data* [TB]	*msg*	Iter	$T_i^{cpu}(14)$ [s]
1	0.471283	60399	1181	182575
2	0.582539	72289	912	182534
3	0.647370	96658	1309	186497
4	0.628352	76490	1288	186241
5	0.778952	113684	1103	187715
6	0.515213	62157	759	188318
7	0.751338	107682	950	122691
8	0.729364	114305	971	122114
9	0.775837	99771	930	161088
10	0.533634	70584	877	162575
11	0.669291	111510	1278	178835
12	0.754863	101648	1749	184893
13	0.367509	47691	393	88569
14	0.244060	32360	226	55921
$\sum$	8.449600	1167228	-	2190566

Tabelle 6.6: Statistiken zur 14-Prozess-Lösung des `SUH3`-Modells mit 896 958 393 Zuständen

der Hauptdiagonalen der Generatormatrix **Q** (DG-VEK), der lokale Empfangspuffer (PUF) und diverse Vektoren für Zwischenresultate (REST). Aufgeführt sind jeweils Mittelwerte über die beteiligten *Worker*-Prozesse. Einzelne *Worker*-Prozesse können demnach mehr Platz benötigen als hier angegeben.
Mit den zur Verfügung stehenden Rechen- und Platzressourcen (10 Rechner mit 6 GB Arbeitsspeicher und 3 GB Platzlimit je Prozess, vgl. Abschnitt 5.3) kann das `SUH1`-Modell mit $k = 6$ ab $p = 4$ *Worker*-Prozessen gelöst werden. Leider ist das `SUH1`-Modell mit $k = 7$ nicht behandelbar, da der durchschnittliche Platzaufwand je *Worker*-Prozess für $p = 12$ 3.13 GB beträgt. Durch eine Erhöhung der Prozessanzahl p kann der Platzaufwand je *Worker*-Prozess unter 3 GB gedrückt werden. Damit unterschreitet man zwar das Platzlimit je Prozess, stößt aber an die Grenze des Arbeitsspeichers. Beispielsweise für $p = 18$ und 6 Rechner benötigt man $3 \cdot 2.69\,GB = 8.07\,GB > 6\,GB$. Ein Erhöhung der Anzahl der Rechner auf 9 würde dieses Problem lösen, erzwingt aber Rechner aus beiden Rechner-Cluster zu nutzen. In diesem Fall tritt jedoch ein Engpass im Inter-Cluster-Netzwerk auf, siehe Abschnitt zu "Kommunikation-Engpass" unten.

Kommunikation-Engpass Bereits bei Modellen mit wenigen Millionen Zuständen kommt es zu Engpässen und Instabilitäten, wenn man Netzwerke mit 10 Mbit/s- und 100 Mbit/s-Verbindungen einsetzt, vgl. Abschnitt 6.1.1.
Für die Berechnungen mit *ASYNC* stehen zwei Rechner-Cluster zur Verfügung, die über das 10/100 MBit/s Netzwerk verbunden sind, vgl. Abb. 5.4. Wenn für die Lösung sehr großer Modelle Rechner aus beiden Clustern genutzt werden (löst u.U. den Platzengpass, s.o.) wird das Netzwerk zum Engpass. Die *Daemon*-Prozesse beanspruchen zunehmend viel Speicher für hängende Nachrichten. Beispielsweise für das `SUH1`-Modell mit k=6 erreicht nach etwa 10 Iterationen (25 Minuten) der erste *Daemon*-Prozess das 3 GB Platzlimit und abgewiesene Speicherplatzanforderungen führen zum Absturz.

	SUH1 k=6					SUH1 k=7				
p	It-Vek	Dg-Vek	Puf	Rest	Total	It-Vek	Dg-Vek	Puf	Rest	Total
1	3.16	3.16	0.00	0.02	**6.34**	9.78	9.78	0.00	0.03	**19.59**
2	1.58	1.58	0.73	0.02	**3.92**	4.89	4.89	2.10	0.03	**11.91**
4	0.79	0.79	0.70	0.02	**2.30**	2.45	2.45	2.29	0.03	**7.22**
6	0.53	0.53	1.05	0.02	**2.13**	1.63	1.63	2.06	0.03	**5.35**
8	0.40	0.40	0.59	0.02	**1.41**	1.22	1.22	1.80	0.03	**4.27**
10	0.32	0.32	0.64	0.02	**1.30**	0.98	0.98	1.70	0.03	**3.69**
12	0.26	0.26	0.63	0.02	**1.17**	0.82	0.82	1.46	0.03	**3.13**
18	0.18	0.18	0.46	0.02	**0.84**	0.54	0.54	1.58	0.03	**2.69**
	SUH2 k=6					SUH2 k=7				
p	It-Vek	Dg-Vek	Puf	Rest	Total	It-Vek	Dg-Vek	Puf	Rest	Total
1	1.57	1.57	0.00	0.02	**3.16**	4.39	4.39	0.00	0.02	**8.80**
6	0.26	0.26	0.35	0.02	**0.89**	0.73	0.73	1.59	0.02	**3.07**

Tabelle 6.7: Platzaufwand [GB] je Prozess für sehr große Modelle

Innerhalb eines Rechner-Clusters mit 1 GBit/s Switch (vgl. Abb. 5.4 rechts und links) können sehr große Modelle mit c.a. 0.5×10^9 Zuständen verteilt gelöst werden, vgl. z.B. das SUH2-Modell mit $k = 7$ (588 759 872 Zustände) und 6 *Worker*-Prozessen. Die durch *ASYNC* verursachte Bandbreite beansprucht das Kommunikationsmedium adäquat. Engpässe treten für sehr kleine Modelle (0.5×10^6 Zustände) und Modelle mit nahezu 1×10^9 Zuständen auf. Bei kleinen Modellen ist bedingt durch die kurzen Rechenzeiten die Iterationsrate und damit auch die Rate, mit der Kommunikationen initiiert werden, extrem hoch. Es ist pragmatisch, bei kleinen Modellen einen sequentiellen Löser anzuwenden, der gewöhnlich nur wenige Minuten benötigt.

6.2 Performance Modellierung

Die Messung und Modellierung der Performance von *ASYNC* zielt auf eine Reduktion der Lösungszeit ab. Weil *ASYNC* ein iteratives Lösungsverfahren implementiert, hängt die Lösungszeit von der Anzahl und der Dauer auszuführender Iterationsschritte ab. Eine modellbasierte Performance-Bewertung von *ASYNC* macht es deswegen notwendig, mathematische Modelle (erfassen Konvergenzrate) und algorithmische Modelle (erfassen Zeit einzelner Iterationsschritte) integrativ zu analysieren.
Die Abb. 6.7 zeigt das Konzept einer integrativen Performance- und Konvergenzanalyse. Der Ausgangspunkt ist ein Performance-Modell, das zwei Aufgaben erfüllt (angedeutet durch ausgehende Pfeile 4 und 5): es liefert eine Schätzung des Zeitverbrauchs $time(t)$ einzelner Iterationsschritte t und das (implizite) Szenario einer asynchronen Iteration (vgl. Def. 5.8). Das Szenario einer hierarchischen Iteration (= Anzahl innerer Iterationsschritte, vgl. Def. 5.3) und die Asynchronitätskonstante B (vgl. Bed. 4.7) parametrisieren das Performance-Modell, weil die Anzahl innerer Iterationen (bestimmt Rechenphasendauer, vgl. Abschnitt 6.1.3 und Kommunikationsrate) und die Asynchronitätskonstante (verursacht bedingte Synchronisationspunkte falls $B < \infty$) die Dauer der Iterationsschritte beeinflussen. Der Interpretation des hierarchischen Szenarios als a-priori bekannter Parameter liegt die Annahme zugrunde, dass die Anzahl innerer Iterationsschritte nicht durch die (a-priori unbekannte) Numerik gesteuert wird (Abbruch über lokale Genauigkeitsschranken).

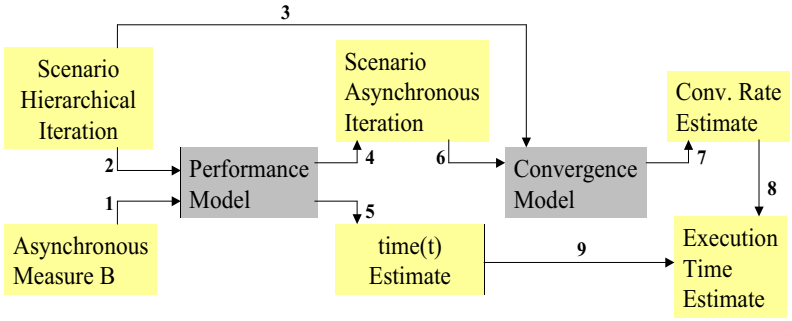

Abbildung 6.7: Konzept Performance-Modellierung hierarchischer und asynchroner Iterationen

Casanova [35] sowie Üresin und Dubois [96] sind Referenzen für eine integrative Performance-Bewertung asynchroner Iterationen. Casanova nutzt mathematische Modelle und Resultate von Baudet [9], Üresin und Dubois greifen auf Resultate von Bertsekas und Tsitsiklis [19] zurück, vgl. auch Abschnitt 5.2.3.
Baudet sowie Bertsekas und Tsitsiklis geben für unterschiedliche Varianten asynchroner Iterationen eine theoretische asynchrone Konvergenzrate in Abhängigkeit von (impliziten) theoretischen synchronen Konvergenzrate an, vgl. Abschnitt 5.2.3. Ihre Kernaussage ist, dass mit Zunahme der Synchronität in asynchronen Iterationen die theoretische asynchrone Konvergenzrate steigt. Die Crux hierbei ist, dass die Aussage sehr schwach ist, die Abschätzung der theoretischen asynchronen Konvergenzrate basiert auf der theoretischen synchronen Konvergenzrate, die selbst nur eine Abschätzung der realen Konvergenzrate ist.
Casanova sowie Üresin und Dubois konstruieren stochastische Modelle zur Bewertung der Ausführungszeit $time(t)$ einzelner oder mehrere asynchroner Iterationsschritte. Die asynchronen Schritte sind so konstruiert, dass theoretische Ergebnisse zur Konvergenz anwendbar sind, nach denen gemäß der mathematischen Modelle die asynchrone Iteration spätestens einen Fortschritt hin zur Lösung erzielt. Wenn man die Art und Weise der "Konstruktion" der asynchronen Schritte genauer betrachtet, stellt man fest, dass sie sehr spezifische Typen asynchroner Iterationen verwenden. Die Spezifik liegt in "versteckter" Synchronität oder auch in Eigenschaften, die in realen Implementierungen asynchroner Iterationen nicht sinnvoll, teilweise sogar nicht erfüllbar sind. Beispielsweise lassen Üresin und Dubois keine Kommunikationsverzögerungen zu und schränken die Auswahlreihenfolge zu iterierender Komponenten ein ("Age-Scheduling"). Casanova führt bedingte Synchronisationspunkte ein und separiert Berechnung und Kommunikation strikt, was aus praktischer Sicht keinen Sinn macht (keine gleichmäßige Belastung des Kommunikationsmediums). Durch diese Restriktionen erfüllen die betrachteten (a)synchronen Iterationen Anforderungen der mathematischen Modelle bzgl. dem Szenario der asynchronen Iteration, vgl. Abb. 6.7. Im Gegensatz zu Üresin und Dubois erlaubt Casanova Kommunikationszeiten. Dies hat zur Folge, dass sein stochastisches Performance-Modell nicht analytisch behandelbar ist, weil es gewisse Zufallsvariablen beinhaltet, deren Charakteristik implizit ist. Deswegen muss er die implizite Charakteristik durch ein Simulationsmodell quantifizieren, dessen Erstellung sehr aufwendig und dessen Auswertung in einigen Punkten unklar ist.
Im nachfolgenden Abschnitt wird eine im Vergleich zu Casanova sowie Üresin und Dubois andere Vorgehensweise gewählt. Es werden ausschlielich praxisrelevante Implementierungen schwach asynchroner Iterationen betrachtet, die keinen Restriktionen unterliegen, die durch mathema-

tische Modelle oder die analytische Behandelbarkeit erzwungen werden. Für die schwach asynchronen Iterationen werden stochastische Modelle zur zeitlichen Bewertung der Iterationsschritte entwickelt und hinsichtlich ihrer analytischen Behandelbarkeit kategorisiert. Dann werden die schwach asynchronen Iterationen zunehmend "entsynchronisiert" und im Detail untersucht, warum die Asynchronität die analytische Behandelbarkeit erschwert.

6.2.1 Performance Modelle

Eine stochastische Performance-Modellierung verteilt ablaufender synchroner und asynchroner Iterationen ist in 3 Schritte unterteilbar:

1. **Modellparameter :** Definition stochastischer Variablen, die a-priori bekannte Zeitverbräuche in der Iteration erfassen.
2. **Modellbildung :** Konstruktion eines analytischen Modells, das einen Zusammenhang zwischen den Modellparametern und der interessierenden Performance-Größe (= stochastische Variable) herstellt.
3. **Modellanalyse :** Berechnung charakterisierender Größen (Verteilungsfunktion, Momente etc) der stochastischen Performance-Variable.

Nachfolgend soll verdeutlicht werden, dass die Konstruktion analytischer stochastischer Modelle bereits für schwach-asynchrone Iterationen aufwendig ist. Hierzu sind in Tab. 6.8 Implementierungsvarianten für *Worker*-Prozesse aufgeführt, welche als Fortsetzung der Varianten aus Tab. 5.4 anzusehen sind.

Variante 4:	Variante 5:	Variante 6:
repeat	**repeat**	**repeat**
barrier_sync	**barrier_sync**	**barrier_sync**
for all $c \in \mathcal{R}(p)$	**for all** $c \in \mathcal{R}(p)$	**for all** $c \in \mathcal{R}(p)$
solve $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$	**solve** $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$	**solve** $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$
put $\boldsymbol{\pi}_{[c]}$	**for all** $c \in \mathcal{R}(p)$	**asy_send** $\boldsymbol{\pi}_{[c]}$
	asy_send $\boldsymbol{\pi}_{[c]}$	
	for all $c \in \mathcal{R}(p)$	**for all** $c \in \mathcal{R}(p)$
	sync $\mathbf{b}_{[c]}$	**sync** $\mathbf{b}_{[c]}$
until convergence	**until** convergence	**until** convergence

Tabelle 6.8: Modelle asynchroner Iterationen und ihre Implementierungen (Prozess p)

Notation und Annahmen für die Algorithmen in Tab. 6.8: Es seien $\mathcal{R}$ die Lastverteilung und $\mathcal{R}(p)$ die Blöcke, die Prozess p zugeordnet sind. Ein **repeat** - **until** Schleifendurchlauf sei per Definition ein Iterationsschritt. Der "**barrier_sync**"-Befehl realisiert einen Synchronisationspunkt, der blockiert, bis alle Prozesse den "**barrier_sync**"-Befehl aufrufen. "**solve** $\boldsymbol{\pi}_{[c]} \cdot \mathbf{Q}[c,c] = \mathbf{b}_{[c]}$" verweist auf die iterative Lösung eines Blocks mit Index c. Danach

startet die Kommunikation via Zugriff auf den geteilten Speicher mit "**put** $\boldsymbol{\pi}_{[c]}$" (nur Variante 4) bzw. mit den asynchronen Sendebefehlen "**asy_send** $\boldsymbol{\pi}_{[c]}$" (Varianten 5 und 6). Der "**sync** $\mathbf{b}_{[c]}$"-Befehl realisiert eine datenabhängige Synchronisation und blockiert bis alle zu $\mathbf{b}_{[c]}$ beitragenden, kommunizierten $\boldsymbol{\pi}_{[\cdot]}$-Vektoren aus dem aktuellen oder aus dem vorherigen Iterationsschritt stammen, vgl. Gl. 5.1.
Die Befehle "**barrier_sync**", "**solve**.." und "**sync**.." sind zeitbehaftet, die restlichen zeitlos. Der Zeitverbrauch für "**solve**.." ist a-priori abschätzbar, die Dauer der Synchronisationsphasen in "**barrier_sync**" und "**sync**.." ist durch die implizte Dynamik des Ablaufs bestimmt und damit implizit.
Die Algorithmen in Tab. 6.8 enthalten keine Befehle zum Empfangen von Daten. Es wird angenommen, dass der Daten-Empfang durch einen separaten "Message-Handler" ausgeführt wird, so dass die *Worker*-Prozesse nicht aktiv am Empfang partizipieren müssen, vgl. Abschnitt 5.6.3 zur "Asynchronen Kommunikation mit Puffer", insbesondere "One-sided"-Kommunikation. Dies bedeutet, dass ein Daten-Empfang nebenläufig zu allen Befehlen im Algorithmus möglich ist.

1. **Modellparameter :** Es ist $|\mathcal{P}|$ die Anzahl der *Worker*-Prozesse und N die Anzahl der Blöcke (zu lösenden Teilprobleme). Es seien $I_1, \ldots, I_N$ unabhängige kontinuierliche und nicht-negative Zufallsvariablen, die Zeitintervalle einzelner innerer Iterationsschritte im "**solve**.."-Befehl erfassen. Es wird angenommen, dass "**solve**.."-Befehle immer mit q inneren Schritte ausgeführt werden, vgl. Abschnitt 4.3 zu "Modellen hierarchischer Iterationen". Des Weiteren seien

   $$C_{1\to 1}, \ldots, C_{N\to 1}, \ldots\ldots C_{1\to|\mathcal{P}|}, \ldots, C_{N\to|\mathcal{P}|}$$

   $N \cdot |\mathcal{P}|$-viele unabhängige kontinuierliche und nicht-negative Zufallsvariablen, die den Zeitverbrauch für Nachrichten mit "Block-Index → Empfänger-Prozess-Index" erfassen.

2. **Modellkonstruktion :** Es sei I die Ausführungszeit eines verteilten, synchronen Iterationsschritts (**repeat** - **until** Schleifendurchlauf). Ein Performance-Modell für I basiert auf Summationen involvierter Zufallsvariablen (Zeitverbräuche werden sequentiell verursacht) und Maximumbildung involvierter Zufallsvariablen (langsamster Prozess oder langsamste Nachricht bestimmt Ausführungszeit). Die Gl. 6.1 bis 6.3 repräsentieren stochastische Modelle der Implementierungsvarianten aus Tab. 6.8 zur Bewertung von I.

   $$I = \max_{p\in\mathcal{P}} q \sum_{c\in\mathcal{R}(p)} I_c \tag{6.1}$$

   $$I = \max_{p\in\mathcal{P}} \left(q \sum_{c\in\mathcal{R}(p)} I_c + \max_{\substack{c\in\mathcal{R}(p)\\ p'\neq p}} C_{c\to p'} \right) \tag{6.2}$$

   $$I = \max_{p\in\mathcal{P}} \max_{l=1\ldots|\mathcal{R}(p)|} \left(q\sum_{i=1}^{l} I_{\mathcal{R}_i(p)} + \max_{p'\neq p} C_{\mathcal{R}_l(p)\to p'} \right) \tag{6.3}$$

   Das Modell 6.1 hat eine einfache Strukur, weil keine Kommunikationsverzögerungen auftreten. Die Maximumbildung über $\mathcal{P}$ selektiert den langsamsten Prozess, dessen Iterationsdauer durch die Summe der I_c-Variablen gegeben ist.

   Die Berücksichtigung des stochastischen Einflusses zeitbehafteter Kommunikation macht stochastische Modelle komplexer, vgl. Gl. 6.2 und 6.3.

Die Maximumbildung über $\mathcal{P}$ in Gl. 6.2 selektiert den "kritischen Prozess", der die Iterationsdauer bestimmt. Hierfür betrachtet man analog zu Variante 4 für jeden Prozess p den Zeitverbrauch in den "**solve**.."-Anweisungen und addiert die maximale Kommunikationszeit der durch p initiierten Nachrichten an Prozesse $p' \neq p$.

In Variante 5 werden alle Daten gleichzeitig verschickt, demgegenüber sind in Variante 6 die Startzeitpunkte in die Berechnungen ("**solve**..") eingebettet und somit unterschiedlich. Deswegen sind in Gl. 6.3 Maximumbildungen und die Summation im Vergleich zu Gl. 6.2 modifiziert. Die innere Maximumbildung vor der Klammer bestimmt die Zeitdauer von Prozess p für die Ausführung von l-vielen "**solve**.."-Anweisungen vermehrt um die maximale Kommunikationszeit der durch p initiierten Nachrichten an Prozesse $p' \neq p$. Dabei verweise $\mathcal{R}_i(p)$ auf das i-te Element der Menge $\mathcal{R}(p)$. Die Reihenfolge der Elemente in $\mathcal{R}(p)$ bzw. die Reihenfolge der "**solve**.."-Anweisungen beeinflusst I. Die äußere Maximumbildung selektiert wiederum den "kritischen Prozess".

3. **Modellanalyse :** Aus der Wahrscheinlichkeitsrechnung bekannte Zusammenhänge ermöglichen die stochastischen Modelle in Gl. 6.1 bis 6.3 zu analysieren, vgl. etwa [61]. Die in Gl. 6.1 bis 6.3 involvierten Zufallsvariablen sind durch Summen- und Maximumoperatoren verknüpft. Die Dichtefunktion der Summe unabhängiger Zufallsvariablen ist die Faltung der individuellen Dichtefunktionen der Zufallsvariablen [61]. Die Verteilungsfunktion des Maximums über Zufallsvariablen ist das Produkt der individuellen Verteilungsfunktionen der Zufallsvariablen [61].

Analyse Gleichung 6.1 Bei der Analyse sind 3 Fälle (a),(b) und (c) zu unterscheiden:

(a) **1 Block je Prozess** ($\forall p \in \mathcal{P} : |\mathcal{R}(p)| = 1$).
In diesem Fall vereinfacht sich Gl. 6.1 zu

$$I = \max(q \cdot I_1, \ldots, q \cdot I_N)$$

und die Verteilungsfunktion von I ist

$$F_I(x) = P[q \cdot I_c \leq x | \forall c = 1 \ldots N] = \prod_{j=1}^{N} F_{I_j}(\frac{x}{q}).$$

(b) **max. 2 Blöcke je Prozess** ($\forall p \in \mathcal{P} : |\mathcal{R}(p)| \leq 2$).
Es sei $2 \cdot N = |\mathcal{P}|$ und der Prozess 1 löse die Blöcke 1 und 2 usw. bis Prozess p löse die Blöcke $2 \cdot p - 1$ und $2 \cdot p$. Dann nimmt die Gl. 6.1 folgende Form an:

$$I = \max(q \cdot (I_1 + I_2), \ldots, q \cdot (I_{2p-1} + I_{2p})).$$

Ein Szenario mit 2 Prozessen ist in Abb. 6.8 veranschaulicht.
Die Dichtefunktion von $I_1 + I_2$ $f_{I_1+I_2}$ ist die Faltung von f_{I_1} und f_{I_1}. Somit gilt

$$\begin{aligned} F_I(x) &= P[q \cdot (I_1 + I_2) \leq x] \cdot \ldots \cdot P[q \cdot (I_{2p-1} + I_{2p}) \leq x] && (6.4) \\ &= \int_0^{\frac{x}{q}} f_{I_1+I_2}(z)\,\mathrm{d}z \cdot \ldots \\ &= \int_0^{\frac{x}{q}} \int_0^{z} f_{I_1}(k) \cdot f_{I_2}(z-k)\,\mathrm{d}k\,\mathrm{d}z \cdot \ldots && (6.5) \end{aligned}$$

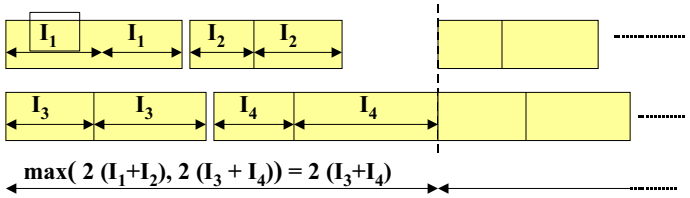

Abbildung 6.8: Visualisierter Trace mit 2 Prozessen, Barrieren-Synchronisation und zeitloser Kommunikation

Die Frage nach der analytischen Behandelbarkeit der Faktoren (=Flächenintegrale) in Gl. 6.5 ist nur in Abhängigkeit von f_{I_1} und f_{I_2} zu beantworten. Falls die I_* Variablen exponentiell verteilt sind, ist $f_{I_1+I_2}$ und $F_{I_1+I_2}$ analytisch (= Dichte- und Verteilungsfunktion der Hypoexponetialverteilung mit zwei Phasen). Es ist vermutlich realistischer anzunehmen, dass die I_* Variablen normal verteilt sind, weil ein innerer Iterationsschritt zahlreiche stochastisch-zeitbehaftete Operationen additiv einschließt. Wenn I_1 und I_2 durch eine Normalverteilung mit $m_1 \triangleq E[I_1]$, $m_2 \triangleq E[I_2]$ und $V[I_1] = V[I_2] = 1$ gegeben sind, dann ist die Faltung der Dichtefunktionen von I_1 und I_2 (inneres Integral in Gl. 6.5) analytisch ausführbar und liefert

$$\int_{-\infty}^{\infty} f_{I_1}(k) \cdot f_{I_2}(z-k)\,\mathrm{d}k = \frac{1}{2\sqrt{\pi}} \exp(-\frac{1}{4}z^2 + z\frac{1}{2}(m_1 - m_2) - \frac{1}{4}(m_1^2 - m_2^2 - 2m_1m_2) \tag{6.6}$$

Die Integralgrenzen in Gl. 6.6 laufen nun von $-\infty$ bis ∞, weil I_* auch negative Werte annehmen kann (Normalverteilung). Die rechte Seite von Gl. 6.6 eingesetzt in Gl. 6.5 ergibt

$$F_I(x) = \frac{1}{2\sqrt{\pi}} \exp(-\frac{1}{4}(m_1^2 - m_2^2 - 2m_1m_2)) \int_0^{\frac{x}{q}} \exp(-\frac{1}{4}z^2 + z\frac{1}{2}(m_1 - m_2))\,\mathrm{d}z \cdot \ldots \tag{6.7}$$

Der Integrand in Gl. 6.7 ist eine Exponentialfunktion mit quadratischer innerer Funktion. Hier ist eine Stammfunktion nicht geschlossen darstellbar, vgl. Verteilungsfunktion der Normalverteilung [61].

(c) **>2 Blöcke je Prozess**
Die Aufgabe ist nun die Berechnung von Wahrscheinlichkeiten der Form

$$P[I_1 + I_2 + I_3 \leq \frac{x}{q}],$$

innerhalb von Gl. 6.4. Ab diesem Punkt muss man auf das Instrumentarium charakteristischer Funktionen zurückgreifen. Die charakteristische Funktion (bzw. Laplace-Transformierte) der Summe endlich vieler unabhängiger Zufallsvariablen ist gleich dem Produkt der charakteristischen Funktionen dieser Zufallsvariablen [61]. Dies bedeutet hier, wenn für I_1, I_2 und I_3 charakteristische Funktionen bekannt sind, ist auch die charakteristische Funktion für $I_1 + I_2 + I_3$ als Produkt dieser charakteristischen Funktionen berechenbar. Andererseits gilt: durch die charakteristische Funktion ist auch die Verteilungsfunktion eindeutig bestimmt [61][3]. Dies bedeutet hier, dass mit

[3]Satz von Levy, siehe Satz 4.5.1 in [61]

der charakteristischen Funktion für $I_1+I_2+I_3$ die Verteilung von $I_1+I_2+I_3$ eindeutig definiert ist. Ob eine Verteilung von $I_1+I_2+I_3$ geschlossen berechenbar ist, kann nur für konkrete Verteilungen von I_1, I_2 und I_3 bzw. der durch sie definierten charakteristischen Funktionen beantwortet werden. Wenn die Verteilungsfunktion von $\sum_{c\in\mathcal{R}} I_c$ bekannt ist, kann die Verteilungsfunktion für I (Maximum) definiert werden.

Analyse Gleichung 6.2 Die Verteilungsfunktion der maximalen Kommunikationsverzögerung (innere Maximumbildung) ist das Produkt der Verteilungsfunktionen $F_{C_{c\to p'}}$. Damit ist die charakteristische Funktion der maximalen Kommunikationsverzögerung definiert und es kann wie im Fall 3c beschrieben vorgegangen werden.

Analyse Gleichung 6.3 Die Gl. 6.3 ist strukturell identisch zu Gl. 6.2. Man hat eine Maximumbildung über Zufallsvariablen, die durch eine Summe definiert sind. Der erste Summand ist eine Summe dedizierter I_* Zufallsvariablen und der zweite Summand ist das Maximum über dedizierte C_* Zufallsvariablen. Deswegen ist die Analyse von Gl. 6.3 analog zu der von Gl. 6.2.

Ein **Fazit**: Für synchronisierte Varianten asynchroner Iterationen aus Tab. 6.8 können stochastische Performance-Modelle für I aufgestellt werden, vgl. Gl. 6.1 bis 6.3. Die Verteilungsfunktion F_I ist in jedem Fall definierbar, ob sie auch geschlossen angegeben werden kann, hängt von den Verteilungs- und Dichtefunktionen der involvierten Zufallsvariablen für die zeitliche Bewertung der Rechen- und Kommunikationszeiten ab. Vorbedingung für alle Analyseansätze ist die Unabhängigkeit involvierter Zufallsvariablen.
Nun soll der Frage nachgegangen werden, wie sich eine "Entsynchronisation" der Implementierungen aus Tab. 6.8 auf die analytische Behandelbarkeit korrespondierender stochastischer Modelle auswirkt. Aus der Theorie der Warteschlangen-Netze ist bekannt, das Synchronisations-Mechanismen die analytische Behandelbarkeit erschweren und deren Aufhebung aus der Sicht der Analyse wünschenswert ist. Andererseits kann - wie nachfolgend gezeigt wird - eine Aufhebung von Synchronisationspunkten neue stochastische Einflussgrößen einführen und so die Analyse erschweren. Konkret besteht die betrachtete "Entsynchronisation" in der Aufhebung der Barrieren-Synchronisation im "**barrier_sync**"-Befehl. Diese Modifikation bewirkt, dass die Varianten 5 und 6 im "**sync** $\mathbf{b}_{[c]}$"-Befehl bis zu unterschiedlichen absoluten Zeiten synchronisieren, vgl. gestrichelte Linien in Abb. 6.9.
Die Variante 4 hat keine Synchronisationspunkte und alle Prozesse iterieren mit individueller Rate. Dies verletzt die Bedingung der "regulated updating sets" (vgl. Bedingung 4.7, Gl. 4.27 auf S. 36). Diese Bedingung ist jedoch essentiell für die Bewertung der Konvergenzrate, vgl. zum Beispiel Abschnitt 5.2.2, Absatz 2 auf S. 46. Deswegen wird für Variante 4 kein stochastisches Modell vorgestellt.

1. **Modellparameter :** Es sei $T_p(t)$ der absolute Startzeitpunkt von Iteration t im Prozess p und es sei $D_p(t)$ die Zeitdauer des Schritts t im Prozess p, vgl. Abb. 6.9. Die Parameter $\mathcal{P}$, N, I_* und $C_{*\to *}$ sind bereits im Absatz "Modellparameter" auf Seite 87 definiert worden.

2. **Modellkonstruktion :** Die Gl. 6.8 und 6.9 repräsentieren stochastische Modelle der Implementierungsvarianten 5 und 6 ohne Barrieren-Symchronisation (vgl. Tab. 6.8) zur Be-

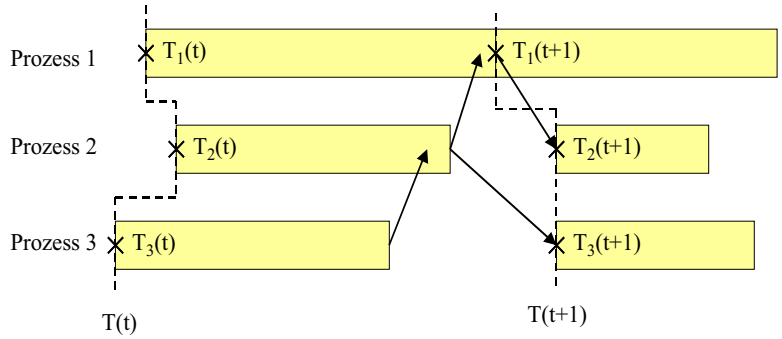

Abbildung 6.9: Visualisierter Trace mit 3 Prozessen ohne Barrieren-Synchronisation

wertung von $D_p(t)$.

$$D_p(t) = \max\left(\mathtt{A}, \max_{p' \neq p}\left(T_{p'}(t) - T_p(t) + \mathtt{B}\right)\right) \text{ und} \tag{6.8}$$

$$D_p(t) = \max\left(\mathtt{A}, \max_{p' \neq p}\left(T_{p'}(t) - T_p(t) + \mathtt{B}\right)\right) \text{ mit} \tag{6.9}$$

$$\mathtt{A} \triangleq q \sum_{c \in \mathcal{R}(p)} I_c \text{ und}$$

$$\mathtt{B} \triangleq \begin{cases} q \sum\limits_{c \in \mathcal{R}(p')} I_c + \max\limits_{c \in \mathcal{R}(p')} C_{c \to p} & \text{für Gl. 6.8} \\ \max\limits_{l=1\ldots|\mathcal{R}(p')|} \left(q \sum\limits_{i=1}^{l} I_{\mathcal{R}_i(p')} + C_{\mathcal{R}_l(p') \to p}\right) & \text{für Gl. 6.9} \end{cases}$$

Die stochastische Modellierung muss zwei Fälle in der algorithmischen Ausführung asynchroner Iterationen unterscheiden:

(a) Ein Prozess p empfängt alle Nachrichten vor Erreichen des "**sync** $\mathbf{b}_{[c]}$"-Befehls. In diesem Fall ist der Aufruf des "**sync** $\mathbf{b}_{[c]}$"-Befehls zeitlos und $D_p(t)$ hängt allein von den I_*-Variablen ab. Dies wird durch den A-Term in Gl. 6.8 und 6.9 erfasst. Im Beispiel-Trace aus Abb. 6.9 zeigt Prozess 1 im t-ten Schritt dieses Verhalten.

(b) Anderenfalls wird $D_p(t)$ durch die Dynamik der "Umgebung" von Prozess p bestimmt, vgl. B-Term. Die innere Maximumbildung über $p' \neq p$ in den Gl. 6.8 und 6.9 selektiert mit dem B-Term den aus Sicht von Prozess p "kritischen Prozess". Ein Prozess p' ist kritisch, wenn er die Ausführungszeit von Prozess p bestimmt. Im Beispiel-Trace aus Abb. 6.9 ist aus der Sicht von Prozess 2 der Prozess 1 kritisch und aus der Sicht von Prozess 3 der Prozess 2. Die B-Terme ähneln Gl. 6.2 und 6.3, die ebenfalls "kritische Prozesse" selektieren. Der wesentliche Unterschied ist der Zusatz der $T_*(t)$-Variablen. Die Korrektur durch die Differenz $T_{p'}(t) - T_p(t)$ berücksichtigt unterschiedliche Startzeitpunkte des aktuellen Schritts t in den Prozessen p und p'.

Die äußere Maximumbildung $\max(\mathtt{A}, \ldots \mathtt{B})$ in Gl. 6.8 und 6.9 selektiert aus zwei nebenläufigen "Prozessen" (lokale Berechnung, Berechnung und Kommunikation der Umgebung) den Prozess, der die lokale Performance im Schritt t bestimmt.

3. **Modellanalyse :** Die Analysis der stochastischen Modelle 6.8 und 6.9 wird durch den Umstand erschwert, dass die Werte der involvierten $T_p(t)$-Variablen implizit sind. $T(t) \triangleq (T_1(t), \ldots, T_{|\mathcal{P}|}(t))$ ist als ein $|\mathcal{P}|$-dimensionaler zeitdiskreter stochastischer Prozess interpretierbar, vgl. auch gestrichelte Linie Abb. 6.9. Seine kontinuierlichen Zustände kodieren eine "Zeitinformation", von der die Dynamik einzelner Iterationsschritte und damit auch $D_p(t)$ abhängt. In die Gleichungen 6.8 und 6.9 gehen Zeit-Differenzen ein. Der $(|\mathcal{P}| - 1)$-dimensionale zeitdiskrete stochastische Prozess

$$\Delta(t) \triangleq \big(T_2(t) - T_1(t), \ldots, T_{|\mathcal{P}|}(t) - T_1(t)\big)$$

kodiert die gleiche Zeitinformation. Aus $\Delta(t)$ sind sowohl Zeit-Differenzen $T_{p'}(t) - T_p(t)$ beliebiger Prozesse p und p', als auch alle absoluten Zeiten $T_p(1), T_p(2), \ldots$ direkt ableitbar.

Eine weitere wichtige Beobachtung ist, dass der stochastische Prozess $T(t)$ bzw. $\Delta(t)$ ein Markov-Prozess ist. Die Verteilung der Zufallsvariablen $T_p(t+1)$ hängt nur vom Zustand $\mathbf{x} \in \mathbb{R}^{|\mathcal{P}|}$ der Zufallsvariablen $T(t)$ ab, denn

$$\mathrm{P}[T_p(t+1) < y] = P[T_p(t) + D_p(t) < y] = P[D_p(t) < y - \mathbf{x}_p \,|T(t) = \mathbf{x}]. \tag{6.10}$$

In Gl. 6.10 sei $\mathbf{x}_p$ die p-te Komponente des Zustand-Vektors $\mathbf{x}$. Es ist wichtig anzumerken, dass die bedingte Verteilung von $D_p(t)$ rechts in Gl. 6.10 von der Bedingung $T(t) = \mathbf{x}$ abhängt und nicht nur etwa von $T_p(t) = \mathbf{x}_p$, weil in die Gl. 6.8 und 6.9 und damit auch in die Verteilung von $D_p(t)$ die Information über $T(t)$ eingeht.

Wegen der Beziehung $T(t+1) = T(t) + D(t)$ mit $D(t) \triangleq (D_1(t), \ldots, D_{|\mathcal{P}|}(t))$ ist T als ein Prozess mit "Zuwächsen" interpretierbar. In der Theorie der zustandskontinuierlichen Markov-Prozesse sind Prozesse mit unabhängigen Zuwächsen bedeutsam, weil sie analytisch behandelbar sind [61]. Leider sind die Zufallsvariablen $D_p(0), D_p(1), \ldots$ hier nicht unabhängig, weil die $D_p(t)$-Verteilung von $T(t) = \sum_{i=1}^{t-1} D(t)$ abhängt.

Ein **Fazit**: Eine "Entsynchronisation" der schwach-asynchronen Iterationen aus Tab. 6.8 durch Aufhebung der Barrieren-Synchronisation führt zu stochastischen Modellen, die von einem Markov-Prozess abhängen, dessen Charakteristik wiederum durch implizite Systemgrößen beeinflusst wird. Eine analytische Lösung ist nicht bekannt.

Kapitel 7

Verfügbarmachung des *ASYNC* Instrumentariums

Das *ASYNC*-Instrumentarium ist über die APNN-Notation (Petri-Netze) der *APNN-Toolbox* und über die *ProC/B*-Schnittstelle benutzerfreundlich verfügbar. Die *ProC/B*-Schnittstelle etabliert eine anwendungsspezifische Schnittstelle für die Modellierung und Analyse logistischer Systeme, vgl. Abb. 1.1. Durch automatisierte Übersetzer ist es möglich, *ProC/B*-Modelle auf Petri-Netz-Modelle in der APNN-Notation abzubilden. Aus diesen kann das analytische Modell für *ASYNC* - die Markov-Kette - abgeleitet werden. Die Ansteuerung von *ASYNC* ist in die graphische Benutzeroberfläche der *APNN-Toolbox* integriert.
Dieses Kapitel ist wie folgt strukturiert: im Abschnitt 7.1 wird die prozess-orientierte Modellierung mit der *ProC/B*-Notation eingeführt. Der Abschnitt 7.2 erläutert die Abbildung von *ProC/B*-Modellen auf Petri-Netz-Modelle in der APNN-Notation. Mit bereits verfügbaren Verfahren kann aus der APNN-Notation eine Markov-Kette mit hierarchischer Kronecker-Algebra generiert werden, vgl. Abschnitt 3.2. Im Abschnitt 7.3 wird diese spezifische Markov-Ketten-Darstellung anhand einer konkreten Systemmodellierung exemplifiziert. Schließlich wird im Abschnitt 7.4 die Ansteuerung von *ASYNC* über die graphische Benutzeroberfläche anhand eines Anwendungsfalls erläutert.

7.1 Prozess-orientierte Modellierung mit *ProC/B*

Auf Prozessketten basierende Notationen wie zum Beispiel Ereignisgesteuerte Prozessketten [89] oder UML-Aktivitätsdiagramme [69] haben sich in der Praxis als Beschreibungsmittel betrieblicher Abläufe etabliert. Die Praxis der prozess-orientierten Modellierung technischer Systeme wird durch zahlreiche Forschungsaktivitäten begleitet, die u.a. zur Gründung des Arbeitskreises "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten" in der Gesellschaft für Informatik (GI)[1] und des SFBs zur "Modellierung großer Netze in der Logistik" [15] führten. Letzterer hat die Entwicklung der *ProC/B*-Notation vorangetrieben.

Die *ProC/B*-Notation [11] unterstützt eine objekt- und prozess-orientierte sowie hierarchische Sichtweise bei der Modellierung und hat sowohl Konzepte des Prozessketten-Paradigmas von Kuhn [72] adoptiert, wodurch Anforderungen und Bedarfe des Anwendungsfeldes berücksichtigt sind, als auch Konzepte des Modellierungs- und Analyse-Tools HIT von Beilner [16], wodurch eine hinreichend formale Beschreibung und eindeutige Interpretation des Modells als Voraussetzung für eine automatisierte Übersetzung in ein Leistungsmodell erreicht wird. Ein Alleinstellungsmerkmal der *ProC/B*-Notation ist, dass sie synergetisch prozess- und objektfluss-orientierte Sichtweisen verzahnt, so dass Nachteile einer rein objektfluss-orientierten Sichtweise (zeitlicher Aspekt wird nur indirekt abgebildet) und prozess-orientierten Sichtweise (statische Aufbaustruktur wird nur indirekt abgebildet) aufgehoben sind. Das *ProC/B*-Tool [11] bietet eine graphisch-basierten Editor zur *ProC/B*-Modellbildung.

In diesem Abschnitt werden die Konzepte der *ProC/B*-Notation erläutert. *ProC/B*-Beispielmodellierungen werden im Kapitel 8.2 behandelt.

Die *ProC/B*-Notation unterscheidet Konstrukte zur Modellierung von Systemstruktur und -verhalten. Das zentrale "strukturierende" Konstrukt ist die `Funktionseinheit` (`FE`). Eine `FE` repräsentiert ein modulares Teilsystem. Jede `FE` inkludiert Modellelemente einschließlich `FE`s (hierarchische Aufbaustruktur durch Enthaltensein-Hierarchie), die gemeinsam eine klar definierte Funktionalität anbieten. Die Funktionalität einer `FE` wird über extern nutzbare Schnittstellen (Dienste) angeboten. Eine `FE` beinhaltet `Prozess-Ketten` (`PK`), `PK-Interfaces`, `Attribute` und unterliegende, "geschachtelte" `FE`s, vgl. Tabelle 7.1. Jede `FE` macht seine Funktionalität durch aufrufbare Dienste zugänglich. Ein Dienst ist formal und im Detail durch eine `PK`-Spezifikation definiert und kann über das `PK-Interface` aufgerufen werden. Die-`PK` Spezifikation definiert eine inhaltlich abgeschlossene, zeitbehaftete und sachlogische Folge von Aktivitäten in Form von `PK-Elementen` (`PKE`), die bei Dienstaufruf auszuführen sind. Die einer `FE` zugeordneten `PKs` lösen somit ein Verhalten aus, das `FE` intern Zustandsänderungen oder `FE` extern bestimmte Reaktionen herbeiführt. Hierbei wird ein hierarchisches Sichtenkonzept realisiert, bei dem eine `FE` nach außen sichtbare `PK-Interfaces` kapselt, die formale Implementierung der jeweiligen Dienste durch die `PK` bleibt für die aufrufende Instanz verborgen. Sie registriert bei Dienstaufruf via dem `PK-Interface` lediglich eine Zeitverzögerung und/oder Werte von Rückgabeparametern. Die Verhaltensmodellierung ist durch eine (ablauf/prozess)-orientierte Sichtweise geprägt und beinhaltet hierfür gängige Konstrukte für die Instanziierung von Prozessen, Nebenläufigkeit, Synchronisation, boolsche und probabilistische Verzweigung und Zusammenführung von Prozessen (Konnektoren) und Schleifen.

Die `PK`-Spezifikation beinhaltet die Disposition (Zuordnung) von Ressourcen zu Aktivitäten, deren Ausführung Ressourcen erfordert. In der Nomenklatur der *ProC/B*-Notation ausgedrückt

[1] http://epk.et-inf.fho-emden.de/index.php

ProC/B Element	Symbol	Semantik
`Funktionseinheit` (`FE`)		modulares Systemteil (Komponenten-Semantik) `FE`-Gesamtheit modelliert hierarchische Aufbaustruktur
`Server`		aktive Ressource (Zeitverbrauch), Bedienstation-Semantik
`Counter`		passive Ressource (Mengenverbrauch), Semaphor-Semantik
`Prozess-Kette` (`PK`)	diverse	ablauflogische Anordnung Aktivitäten (=`PKE`) im Prozess
`Quelle`	$\odot$	virtuelle oder aktive Prozess-Erzeugung (Lastquelle)
`Senke`	$\otimes$	virtuelle oder aktive Prozess-Terminierung
`PK-Interface`	Kringel	Aufrufschnittstelle für `PK`
`PK-Element` (`PKE`)		Basis `PK`-Baustein (Aktivität)
`Delay-PKE`		statischer/stochastischer Zeitverbrauch im Prozess
`Code-PKE`		auszuführender HIT Programm-Code [16]
`Loop-PKE`		kapselt `PKE`s in Schleife, boolsche/stochast. Abbruchbedingung
`AND-SPLIT-Konnektor`		Prozess-Aufsplittung (“fork”)
`AND-JOIN-Konnektor`		Prozess-Synchronisation
`AND-Konnektor`		Kombination `AND-SPLIT/JOIN-Konnektor`
`OR-SPLIT-Konnektor`		stochastische/boolsche Prozessverzweigung (“branch”)
`OR-JOIN-Konnektor`		Prozesszusammenführung (“join”)
`PK-Konnektor`		ähnlich `AND-Konnektor`, erlaubt Erzeugung neuer Prozesse
`FE-Variable`	textuell	Variable mit Geltungsbereich in `FE`
`PK-Parameter`	textuell	Aufrufparameter für `PK` (=Geltungsbereich)

Tabelle 7.1: *ProC/B* Modellelemente (Auswahl): Graphische Darstellung und Semantik

können `PKEs` (=Aktivitäten) unter Zuhilfenahme vordefinierter und parametrisierter Ressourcen wie zum Beispiel `Server` (aktive Ressource mit Bedienstation-Semantik und Zeitverbrauch) oder `Counter` (passive Resource mit Semaphor-Semantik), oder alternativ durch Dienste selbstdefinierter `FEs` ausgeführt werden. Dabei kann in beiden Fällen die Zeit bzw. der Umfang der Inanspruchnahme dynamisch quantifiziert werden. So wird neben der Struktur-Hierarchie eine Verhalten-Hierarchie ermöglicht, die auf Dienstaufrufen basiert (Aufruf-Hierarchie) und die Spezifikation zur Ausführung einzelner `PKEs` (=Aktivitäten) verfeinert (beinhaltet Konzept der Selbstähnlichkeit). Die Struktur-Hierarchie ist durch `Server` bzw. `Counter` abgeschlossen, die keine unterliegenden `FEs` enthalten dürfen.

Aus der “Last-Ressourcen Sicht” agieren `FEs`, `Server` und `Counter` als “Ressourcen”. `PKs` definieren das Interaktionsmuster zur Auferlegung der Last (=Verhalten) und sind innerhalb der `FE`s von der Ressourcenbeschreibung deutlich abgegrenzt (Sichtenkonzept).

Eine Modellbildung logistischer Systeme mit Petri-Netzen ist prinzipiell möglich [54, 57]. Jedoch hat die Erfahrung im Umgang mit Gestaltern logistischer Systeme gezeigt, dass eine anwendungsspezifische Notation wie *ProC/B* zu erheblich höherer Akzeptanz einer modellbasierten Systemanalyse führt und Systemgestalter befähigt, Wissen und Informationen über logistische Systeme in den Prozess der Modellbildung einzubringen. Im nachfolgenden Abschnitt 7.2 wird verdeutlicht, dass Petri-Netze insofern bedeutsam sind, als sie als Bindeglied zwischen *ProC/B* und analytischen Modellen fungieren.

7.2 Modell-Transformation von *ProC/B* nach Petri Netzen

Eine übliche Methodik der modellbasierten Analyse besteht darin, die manuelle Modellbildung durch eine hochsprachliche Notation zu unterstützen, die in ihrer Semantik hinreichend formal und eindeutig ist, um aus ihr automatisiert ein analytisches Modell ableiten zu können, das als Eingabe der Analyseverfahren dient. Das hier angestrebte analytische Modell ist eine Markov-Kette mit hierarchischer Kronecker-Algebra (vgl. Abschnitt 3.2). P. Buchholz und P. Kemper haben bereits Abbildungen von *SGSPN*s in dieses analytische Modell konzipiert und implementiert [33, 32]. Deswegen ist es naheliegend, auf eine vergleichsweise aufwendige Re-Implementierung dieser Methodik speziell für *ProC/B* zu verzichten und anstelle dessen *ProC/B*-Modelle zunächst auf *SGSPN*s abzubilden und dann auf die verfügbare Methodik zurückzugreifen. Dieses Vorgehen ist in der Abb. 7.1 dargestellt. Das gestrichelte Rechteck zeigt, wie *GSPN*s eingesetzt werden, um

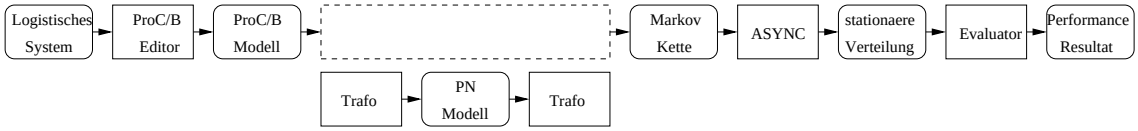

Abbildung 7.1: Modellbildung logistischer Systeme und die Anbindung von *ASYNC*

die Lücke zwischen der hochsprachlichen *ProC/B* Modellbildung und dem analytischen Modell zu schließen. Die *APNN-Toolbox* integriert die Methodik, die in Abb. 7.1 rechts vom *ProC/B*-Modell dargestellt ist. Das *ProC/B*-Toolset unterstützt die *ProC/B*-Modellbildung logistischer Systeme und die Ansteuerung verschiedener Analysewerkzeuge (linker Teil Abb. 7.1).
Der konzeptionelle Entwurf und die Implementierung der Abbildung von *ProC/B*-Modellen auf *GSPN*-Modelle wurden in Kooperation mit P. Kemper, C. Tepper und Z. Wu realisiert [58, 59]. Der Definitionsbereich der Abbildung ist eine wohldefinierte Teilmenge von *ProC/B*-Modellen. Er resultiert aus Vorgaben des angestrebten analytischen Modells (Markov-Kette) und Einschränkungen des Analyseverfahrens (Größe behandelbarer Zustandsraum). So müssen alle Zeitverbräuche im *ProC/B*-Modell durch Exponentialverteilungen spezifiziert sein und datenabhängiges Verhalten (Prozess-Parameter, Variablen und deren Manipulation und Abfrage, boolscher `AND-Konnektor` etc.) kann nicht oder nur sehr eingeschränkt abgebildet werden. In Tab. 7.2 wird aufgezählt, welche Ausprägungen von *ProC/B*-Konstrukten durch die Abbildung unterstützt werden, nur bedingt (approximativ) abbildbar sind oder nicht behandelt werden können.
Die grundlegende Idee der Abbildung ist, für alle Konstrukte der *ProC/B*-Notation (siehe Tab. 7.1) verhaltensäquivalente Petri-Netz Beschreibungen zu definieren (=lokale Ersetzung) und diese gemäß ihrer ablauflogischen Reihenfolge korrekt zu verknüpfen. Auf die Details der Abbildung soll nicht weiter eingegangen werden und es sei hierfür auf [58, 59] verwiesen.
Es sei angemerkt, dass durch die Abbildung auf *PN*s nicht nur analytische Markov-Ketten Modelle erschlossen, sondern auch ein breites Spektrum von Analyseverfahren (Invarianten etc), die direkt auf *PN*s agieren.
Ein Defizit der Abbildung ist, dass kein strukturiertes *SGSPN*-Modell, sondern lediglich ein *GSPN* automatisiert generierbar ist. Eine Strukturierung in Komponenten muss manuell vorgenommen werden.
Technisch ist die *APNN-Toolbox* an das *ProC/B*-Instrumentarium durch Übersetzer angebunden, die mehrstufig über 4 Schnittstellen *ProC/B*-Modelle in Petri-Netz-Modelle (*GSPN*) über-

ProC/B Element	Abbildbarkeit auf Petri-Netz
`FE`	Strukturierungselement ohne Einfluss auf Modell-Dynamik
`Server`	⊕ "Random" Bediendisziplin, Mehrbediener ⊗ "Infinite Server" Semantik durch Mehrbediener approximiert ⊖ FCFS, PS Bediendisziplin, Bedienprioritäten, zustandsabh. Bediengeschwindigkeit
`Counter`	⊕ ein- und mehrdimensionale Zähler mit konstanter (In/De)krementierung ⊖ dynamische, datenabhängige Zählermanipulation
`Quelle`	⊕ Poisson-artige und singuläre Prozesserzeugung (auch "bulk"-Prozesse) ⊖ boolsche (datenabhängige) Prozesserzeugung
`Senke`	⊗ "offene" Prozesse werden via Kurzschluss durch "geschlossene Prozesse" ersetzt
`PK`	beinhaltet (potentiell) alle unten beschriebene Konstrukte
`PK-Interface`	⊖ alle Typen von Aufrufparametern
`Delay-PKE`	⊕ konstante Verzögerung mit Markov'schen Zeitverbrauch ⊗ "Infinite Server" Semantik durch Mehrbediener approximiert ⊖ datenabhängige Zeitverbräuche
`Code-PKE`	⊕ HiSlang Code [16] zur Addition/Subtraktion von Konstanten auf/von `FE-Variable` ⊖ sonstiger HiSlang Code [16]
`Loop-PKE`	⊕ Endlosschleife, boolsche Abbruchbedingung (Test auf `FE-Variablen` Werte) probabilistische Abbruchbedingung (geometrische Verteilung der Durchläufe) ⊗ boolsche Abbruchbedingung durch probabilistische Abbruchbedingung approxim. ⊖ komplexe boolsche Ausdrücke als Abbruchbedingung
`AND-Konnektor`	⊕ "Fork"-Operator zur Erzeugung von Parallel-Prozessen ⊗ "Join-Operator" rekombiniert+synchronisiert jeweils terminierte Parallel-Prozesse, die aus u.U. verschiedenen "Fork" hervorgehen ("Min-Match" Semantik)
`OR-Konnektor`	⊕ probabilistischer "Branch"-Operator, boolsche Variante für einfache Ausdrücke wie Test auf `FE-Variablen` Werte ⊖ komplexe boolsche Ausdrücke für "Branch"-Auswahl
`PK-Konnektor`	⊕ Synchronisation, Erzeugung und Replikation von Prozessen
`FE-Variable`	⊕ Integer-Variablen mit endlichem Definitionsbereich (auch mehrdimensional) ⊖ Variablen ≠ Integer-Typ
`PK-Parameter`	⊖ alle Ausprägungen

Tabelle 7.2: *ProC/B* Modellelemente (Auswahl) und ihre Abbildbarkeit auf Petri Netze: Ausprägungen von *ProC/B*-Konstrukten, die ohne Einschränkung abbildbar sind (= ⊕) , bedingt (approximativ) abbildbar sind (= ⊗) und nicht behandelbar sind (= ⊖).

setzen. Ein Parser liest eine textuelle Spezifikationen des *ProC/B*-Modells (1. Schnittstelle) und erzeugt eine C++ Datenstruktur (2. Schnittstelle), die auf einem C++ Klassendiagramm für *ProC/B*-Modellkonstrukte basiert. Die Konzeption des Parsers und der Schnittstellen geht auf A. van Almsick und F. Bause zurück, die Implementierung wurde durch M. Eickhoff, D. Köpp und D. Sawitzki geleistet. Die *ProC/B* C++ Datenstruktur wird danach in eine Petri-Netz C++ Datenstruktur (3. Schnittstelle) übersetzt. Die Konzepte dieser Abbildung wurden oben rekapituliert. Ein Parser der *APNN-Toolbox* liest die Petri-Netz C++ Datenstruktur aus und erzeugt die APNN-Notation (4. Schnittstelle) - das textuelle Austauschformat für Petri-Netze der *APNN-Toolbox*. Die Konzeption und Implementierung dieses Parsers geht auf P. Kemper und C. Tepper zurück.

7.3 Hierarchische Kronecker-Darstellung der Markov-Kette

In diesem Abschnitt wird für das *SGSPN*-Modell des `MSMQ`-Systems (Abb. 2.1, Bsp. 2.3) eine hierarchische Kronecker-Darstellung der Markov-Kette (Abschnitt 3.2) schrittweise explizit dargestellt. Das `MSMQ`-System ist zwar kein logistisches System, es ist aber wegen der geringen Modellgröße gut geeignet, die Konzepte der Kronecker-Darstellung zu exemplifizieren.
Das *SGSPN*-Modell ist wie folgt parametrisiert: $J = 4$ Stationen (=Komponenten), $S = 1$ Bediener und $C_j = 1, (1 \leq j \leq J)$ als Kapazitäten der Warteräume der einzelnen Stationen. Die lokalen Zustandsräume $\mathcal{S}^j$ können vollständig isoliert von der Umgebung oder in partieller Isolation unter Ausnutzung maximaler Kapazitäten der Stellen des *GSPN*s, welche aus globalen Invarianten resultieren, generiert werden [70]. Im letzteren Fall haben die $\mathcal{S}^j$ jeweils $n_j = 5$ Zustände (zeitlose Zustände sind bereits eliminiert), die zugehörigen Markierungen der Stellen und aktivierte Transitionen sind in Tab. 7.3 aufgeführt.

Zustand	M(p_1)	M(p_2)	M(p_3)	M(p_4)	M(p_5)	t_Buf	t_Proc	t_Sw_j	t_Sw_j+1
0	1	0	0	0	0	×		×	
1	0	1	0	0	0			×	
2	1	0	0	0	1	×		×	×
3	0	0	1	0	0		×	×	
4	0	1	0	0	1			×	×

Tabelle 7.3: Zustände aus $\mathcal{S}^j$, zugehörige Markierungen $M(p_*)$ der Stellen p_* und aktivierte Transitionen t_* (×) für das *GSPN*-Modell aus Abb. 2.1.

Die Matrix $\mathbf{Q}_l^j$ in Gl. 7.1 erfasst alle Raten der Zustandübergänge in $\mathcal{S}^j$, die durch **lokale, zeitbehaftete Transitionen** $t_{Buf}, t_{Proc} \in TL^j$ des *GSPN*-Modells ausgelöst werden. Mit der Reihenfolge der Zustände, wie sie in der Tab. 7.3 aufgelistet sind, hat die Matrix $\mathbf{Q}_l^j$ folgende Einträge:

$$\mathbf{Q}_l^j = \lambda_j[\begin{pmatrix} \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} - \mathbf{D}^j_{\texttt{t\_Buf}}] + \mu_j[\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} - \mathbf{D}^j_{\texttt{t\_Proc}}]$$

$$= \begin{array}{c|cc|ccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & -\lambda_j & \lambda_j & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline 2 & \cdot & \cdot & -\lambda_j & \cdot & \lambda_j \\ 3 & \cdot & \cdot & \mu_j & -\mu_j & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \triangleq \begin{pmatrix} \mathbf{Q}_l^j[0,0] & \mathbf{Q}_l^j[0,1] \\ \mathbf{Q}_l^j[1,0] & \mathbf{Q}_l^j[1,1] \end{pmatrix}. \tag{7.1}$$

Die Blockstruktur von $\mathbf{Q}_l^j$ bzw. die Partitionierung von $\mathcal{S}^j$ in sogenannte lokale Äquivalenzklassen $\mathcal{S}^j[\cdot]$ bzw. **lokale Makrozustände** (*LMZ*)

$$\mathcal{S}^j = \mathcal{S}^j[0] \,\dot\cup\, \mathcal{S}^j[1] \;\text{ mit Indexmenge } \mathcal{Z}^j = \{0,1\} \tag{7.2}$$

resultiert aus einer Äquivalenzrelation, die über das Erreichbarkeitskriterium 7.3 definiert ist [23]. Zwei Zustände $s_1, s_2 \in \mathcal{S}^j$ gehören genau dann einer Äquivalenzlasse an, wenn sie in Kombination

mit den gleichen Zuständen env ihrer Umgebung $\mathcal{ENV}^j \triangleq \times_{\substack{i=1 \\ i\neq j}}^{J} \mathcal{S}^i$ erreichbar sind, d.h. wenn

$$\forall env \in \mathcal{ENV}^j : (s_1, env) \text{ ist erreichbar } \Leftrightarrow (s_2, env) \text{ ist erreichbar} \tag{7.3}$$

gilt. Die lokalen Zustände 2 und 4 aus $\mathcal{S}^j[1]$ (für alle j) repräsentieren genau die Zustände, in denen der Bediener in Komponente j ist (genau eine der Stellen `p_3`, `p_4` und `p_5` hat ein Token). Die Zustände 0 und 1 können bei Abwesenheit des Bedieners eintreten. Die Anzahl der Äquivalenzklassen ist minimal. Zustände aus $\mathcal{S}^j[0]$ und $\mathcal{S}^j[1]$ können nicht zusammengefasst werden, weil der Bediener immer in genau einer Warteschlange residiert.
Die globalen Äquivalenzklassen $\mathcal{S}[\cdot]$ des globalen Zustandsraums $\mathcal{S}$ bzw. die **globalen Makrozustände** (GMZ) sind das Kreuzprodukt lokaler Äquivalenzklassen bzw. LMZs, wobei aus der Menge kombinatorisch möglicher Kreuzprodukte stets nur GMZs erzeugt werden, die ausschließlich erreichbare Zustände beschreiben. Potentiell können somit $2^4 = 16$ GMZ entstehen, im Beispiel sind es nur $N = 4$ GMZs. Weil der Bediener immer in genau einer Warteschlange residiert, ist $\mathcal{S}^j[1]$ genau einmal in GMZs involviert. Die GMZ definieren somit die Verteilung der S Bediener (Population) über den J Komponenten. Rein kombinatorisch gibt es hierfür $\binom{J+S-1}{S}$ Möglichkeiten. Für die Benennung von GMZ ist es hilfreich, Indexmengen für lokale Makrozustände ($\mathcal{Z}^{j>0}$) und globale Makrozustände ($\mathcal{Z}^0$) zu definieren. Jeder GMZ $X \in \mathcal{Z}^0$ ist als Kreuzprodukt von $J = 4$ LMZs

$$\mathcal{Z}^0 = \mathcal{Z}^1 \times \mathcal{Z}^2 \times \mathcal{Z}^3 \times \mathcal{Z}^4,$$

und somit als ein 4-Tupel $X = (X^1, X^2, X^3, X^4)$ identifizierbar. Im Beispiel ist $\mathcal{Z}^0 = \{0, 1, 2, 3\}$ ($\Rightarrow N = 4$) und $\mathcal{Z}^j = \{0, 1\}$ für $j = 1 \ldots 4$ und die GMZs ausgedrückt durch die Indizes involvierter LMZs

$$\mathcal{Z}^0 = \{ \underbrace{(1\,0\,0\,0)}_{GMZ \text{ Index } 0}, \underbrace{(0\,1\,0\,0)}_{GMZ \text{ Index } 1}, \underbrace{(0\,0\,1\,0)}_{GMZ \text{ Index } 2}, \underbrace{(0\,0\,0\,1)}_{GMZ \text{ Index } 3} \}.$$

Alternativ sind GMZs durch explizite Angabe involvierter Zustände lokaler Zustandsräume $\mathcal{S}^j$ beschreibbar

GMZ Index	$\mathcal{S}^1$	$\mathcal{S}^2$	$\mathcal{S}^3$	$\mathcal{S}^4$	Zustände
0	$2 \ldots 4$	$0 \ldots 1$	$0 \ldots 1$	$0 \ldots 1$	$3 \cdot 2 \cdot 2 \cdot 2 = 24$
1	$0 \ldots 1$	$2 \ldots 4$	$0 \ldots 1$	$0 \ldots 1$	$2 \cdot 3 \cdot 2 \cdot 2 = 24$
2	$0 \ldots 1$	$0 \ldots 1$	$2 \ldots 4$	$0 \ldots 1$	$2 \cdot 2 \cdot 3 \cdot 2 = 24$
3	$0 \ldots 1$	$0 \ldots 1$	$0 \ldots 1$	$2 \ldots 4$	$2 \cdot 2 \cdot 2 \cdot 3 = 24$

bzw.

$$\begin{aligned} \mathcal{S} &= \mathcal{S}[0] \cup \mathcal{S}[1] \cup \mathcal{S}[2] \cup \mathcal{S}[3] \\ &= \mathcal{S}^1[1] \times \mathcal{S}^2[0] \times \mathcal{S}^3[0] \times \mathcal{S}^4[0] \cup \\ &\quad \mathcal{S}^1[0] \times \mathcal{S}^2[1] \times \mathcal{S}^3[0] \times \mathcal{S}^4[0] \cup \\ &\quad \mathcal{S}^1[0] \times \mathcal{S}^2[0] \times \mathcal{S}^3[1] \times \mathcal{S}^4[0] \cup \\ &\quad \mathcal{S}^1[0] \times \mathcal{S}^2[0] \times \mathcal{S}^3[0] \times \mathcal{S}^4[1]. \end{aligned} \tag{7.4}$$

Die Generatormatrix $\mathbf{Q}_l$ lokaler Transitionen ist wie $\mathbf{Q}$ blockstrukturiert und es sei

$$\mathbf{Q}_l \triangleq \begin{pmatrix} \mathbf{Q}_l[0,0] & \mathbf{Q}_l[0,1] & \mathbf{Q}_l[0,2] & \mathbf{Q}_l[0,3] \\ \mathbf{Q}_l[1,0] & \mathbf{Q}_l[1,1] & \mathbf{Q}_l[1,2] & \mathbf{Q}_l[1,3] \\ \mathbf{Q}_l[2,0] & \mathbf{Q}_l[2,1] & \mathbf{Q}_l[2,2] & \mathbf{Q}_l[2,3] \\ \mathbf{Q}_l[3,0] & \mathbf{Q}_l[3,1] & \mathbf{Q}_l[3,2] & \mathbf{Q}_l[3,3] \end{pmatrix} \in \mathbb{R}^{96\times 96}. \tag{7.5}$$

Für die generische Beschreibung der Matrizen $\mathbf{Q}_l[\cdot,\cdot]$ können die Operatoren der Kronecker-Algebra verwendet werden, vgl. Def. 3.6 auf S. 21. Lokale Transitionen im *GSPN* definieren Zustandsübergänge, bei denen Start- und Zielzustand entweder im selben *LMZ* sind (*GMZ* bleibt unverändert) oder einen *LMZ*-Übergang induzieren (*GMZ* ändert sich in genau einer Komponente). Im Beispiel verändern sich bei einem *GMZ*-Übergang jedoch stets genau 2 involvierte *LMZ*s, z.B. $(1\,0\,0\,0) \rightarrow (0\,1\,0\,0)$. Demzufolge können Übergänge zwischen *GMZ*s nicht durch lokale Transitionen ausgelöst werden und alle Nicht-Diagonalblöcke in $\mathbf{Q}_l$ sind gleich 0. Die Hauptdiagonalblöcke, welche Zustandsübergänge innerhalb von *LMZ*s erfassen, haben folgende Gestalt:

$$\begin{aligned}
\mathbf{Q}_l[0,0] &= \mathbf{Q}_l^1[1,1] \oplus \mathbf{Q}_l^2[0,0] \oplus \mathbf{Q}_l^3[0,0] \oplus \mathbf{Q}_l^4[0,0] \\
&= \begin{pmatrix} -\lambda_1 & 0 & \lambda_1 \\ \mu_1 & -\mu_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_2 & \lambda_2 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_3 & \lambda_3 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_4 & \lambda_4 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{24\times 24}
\end{aligned}$$

$$\begin{aligned}
\mathbf{Q}_l[1,1] &= \mathbf{Q}_l^1[0,0] \oplus \mathbf{Q}_l^2[1,1] \oplus \mathbf{Q}_l^3[0,0] \oplus \mathbf{Q}_l^4[0,0] \\
&= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_2 & 0 & \lambda_2 \\ \mu_2 & -\mu_2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_3 & \lambda_3 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_4 & \lambda_4 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{24\times 24}
\end{aligned}$$

$$\begin{aligned}
\mathbf{Q}_l[2,2] &= \mathbf{Q}_l^1[0,0] \oplus \mathbf{Q}_l^2[0,0] \oplus \mathbf{Q}_l^3[1,1] \oplus \mathbf{Q}_l^4[0,0] \\
&= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_2 & \lambda_2 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_3 & 0 & \lambda_3 \\ \mu_3 & -\mu_3 & 0 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_4 & \lambda_4 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{24\times 24}
\end{aligned}$$

$$\begin{aligned}
\mathbf{Q}_l[3,3] &= \mathbf{Q}_l^1[0,0] \oplus \mathbf{Q}_l^2[0,0] \oplus \mathbf{Q}_l^3[0,0] \oplus \mathbf{Q}_l^4[1,1] \\
&= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_2 & \lambda_2 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_3 & \lambda_3 \\ 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} -\lambda_4 & 0 & \lambda_4 \\ \mu_4 & -\mu_4 & 0 \\ 0 & 0 & 0 \end{pmatrix} \in \mathbb{R}^{24\times 24}
\end{aligned}$$

Diese Darstellung zeigt, dass nur 5 Einträge (Hauptdiagonaleinträge nicht mitgezählt) die Matrizen $\mathbf{Q}_l[0,0] \ldots \mathbf{Q}_l[3,3]$ mit 52 Einträgen generieren. Diese Matrizen erzeugen generisch die Matrix $\mathbf{Q}_l[1,1] =$

$$\begin{pmatrix}
\cdot & \lambda_4 & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \lambda_4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mu_2 & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_2 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_3 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \lambda_3 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_3 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \lambda_4 & \cdot
\end{pmatrix}$$

Abschließend sollen die Matrizen $\mathbf{Q}_t^j$ ($t \in TS = \{\texttt{Sw\_1}, \texttt{Sw\_2}, \texttt{Sw\_3}, \texttt{Sw\_4}\}$) betrachtet werden. Transitionen aus TS bewirken Zustandsübergänge in mindestens zwei Komponenten des *SGSPN*s. Beispielsweise $\texttt{Sw\_2} \in TS$ induziert einen Übergang in Komponente 2 (Ankunft Bediener) und in Komponente 1 (Abgang Bediener). Der Zustand in Komponente 3 und 4 bleibt unverändert und man setzt $\mathbf{Q}^3_{\texttt{Sw\_2}} = \mathbf{I}_{n_3}$ und $\mathbf{Q}^4_{\texttt{Sw\_2}} = \mathbf{I}_{n_4}$ ($n_3 = n_4 = 5$).

$$\mathbf{Q}^1_{\texttt{Sw\_2}} = \begin{array}{c|cc|ccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline 2 & 1 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & 1 & \cdot & \cdot & \cdot \end{array} \triangleq \begin{pmatrix} \mathbf{Q}^1_{\texttt{Sw\_2}}[0,0] & \mathbf{Q}^1_{\texttt{Sw\_2}}[0,1] \\ \mathbf{Q}^1_{\texttt{Sw\_2}}[1,0] & \mathbf{Q}^1_{\texttt{Sw\_2}}[1,1] \end{pmatrix} \tag{7.6}$$

$$\mathbf{Q}^2_{\texttt{Sw\_2}} = \begin{array}{c|cc|ccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & \cdot & \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & 1 & \cdot \\ \hline 2 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \triangleq \begin{pmatrix} \mathbf{Q}^2_{\texttt{Sw\_2}}[0,0] & \mathbf{Q}^2_{\texttt{Sw\_2}}[0,1] \\ \mathbf{Q}^2_{\texttt{Sw\_2}}[1,0] & \mathbf{Q}^2_{\texttt{Sw\_2}}[1,1] \end{pmatrix}. \tag{7.7}$$

Es sei $\mathbf{Q}_{\texttt{Sw\_2}}[X,Y]$ der (additive) Beitrag von Transition $\texttt{Sw\_2} \in TS$ zu $\mathbf{Q}[X,Y]$, der alle Übergänge von *GMZ* $X \in \mathcal{Z}^0$ nach $Y \in \mathcal{Z}^0$ erfasst. Im Beispiel ist

$$\mathbf{Q}_{\texttt{Sw\_2}}[X,Y] \begin{cases} \neq \mathbf{0} & \text{falls } X = 0 \text{ und } Y = 1 \\ = \mathbf{0} & \text{sonst,} \end{cases}$$

weil die Transition Sw_2 nur einen Übergang vom *GMZ* mit Index 0 in den *GMZ* mit Index 1 erzeugt. Die Matrix $\mathbf{Q}_{\texttt{Sw\_2}}[0,1] \in \mathbb{R}^{24\times 24}$ hat die folgende Gestalt:

$$\mathbf{Q}_{\texttt{Sw\_2}}[0,1] = \omega_1 \cdot \mathbf{Q}^1_{\texttt{Sw\_2}}[1,0] \otimes \mathbf{Q}^2_{\texttt{Sw\_2}}[0,1] \otimes \mathbf{Q}^3_{\texttt{Sw\_2}}[0,0] \otimes \mathbf{Q}^4_{\texttt{Sw\_2}}[0,0]$$

$$= \omega_1 \cdot \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Ähnlich verhält es sich mit den restlichen Transitionen Sw_j aus TS, sie induzieren folgende *GMZ* Übergänge ($\xrightarrow{\texttt{Sw\_j}}$):

$$\underbrace{\underbrace{(1000)}_{GMZ\ 0} \xrightarrow{\texttt{Sw\_2}} \underbrace{(0100)}_{GMZ\ 1}}_{\Rightarrow \mathbf{Q}_{\texttt{Sw\_2}}[0,1] \neq \mathbf{0}} \xrightarrow{\texttt{Sw\_3}} \underbrace{(0010)}_{GMZ\ 2} \xrightarrow{\texttt{Sw\_4}} \underbrace{(0001)}_{GMZ\ 4} \xrightarrow{\texttt{Sw\_1}} \underbrace{(2111)}_{GMZ\ 0}.$$

Sie beschreiben Zustandstransitionen auf der oberen Ebene der hierarchischen Darstellung. Auf gleiche Weise werden die Matrizen $\mathbf{Q}_{\texttt{Sw\_1}}$, $\mathbf{Q}_{\texttt{Sw\_3}}$ und $\mathbf{Q}_{\texttt{Sw\_4}}$ erzeugt, in denen jeweils genau der Anteil $\mathbf{Q}_{\texttt{Sw\_1}}[3,0]$, $\mathbf{Q}_{\texttt{Sw\_3}}[1,2]$ und $\mathbf{Q}_{\texttt{Sw\_4}}[2,3]$ ein Block mit Einträgen ungleich 0 ist.

Die Gesamtdarstellung der hierarchischen und strukturierten Generatormatrix in ihrer allgemeinen Form aus Gl. 3.9 nimmt für das Beispiel die Form

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_l[0,0] & \mathbf{Q}_{\mathtt{Sw\_2}}[0,1] & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_l[1,1] & \mathbf{Q}_{\mathtt{Sw\_3}}[1,2] & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_l[2,2] & \mathbf{Q}_{\mathtt{Sw\_4}}[2,3] \\ \mathbf{Q}_{\mathtt{Sw\_1}}[3,0] & \mathbf{0} & \mathbf{0} & \mathbf{Q}_l[3,3] \end{pmatrix} + \mathbf{D} \tag{7.8}$$

an. Die nicht explizit angegebene Diagonalmatrix $\mathbf{D}$ erzeugt die "passenden" Diagonaleinträge.

7.4 Integration *APNN-Toolbox*

Das gesamte *ASYNC*-Instrumentarium ist in die *APNN-Toolbox* integriert und von der graphischen Benutzeroberfläche aus aufrufbar. Die Ansteuerung des *ASYNC*-Instrumentariums ist in 4 Schritte unterteilt: 1. Generierung Markov-Kette, 2. Vorbereitung und Einrichtung des numerischen Lösers, 3. Aufruf des numerischen Lösers und 4. Auswertung der Ausführung des numerischen Lösers und Aufbereitung der Markov-Ketten Analyseergebnisse. Alle Schritte sind einzeln und sequentiell in obiger ablauflogischer Reihenfolge von einer graphischen Benutzeroberfläche aus aufrufbar, vgl. Abb. 7.2, Fenster oben. Auf oberster Ebene sind die Schritte

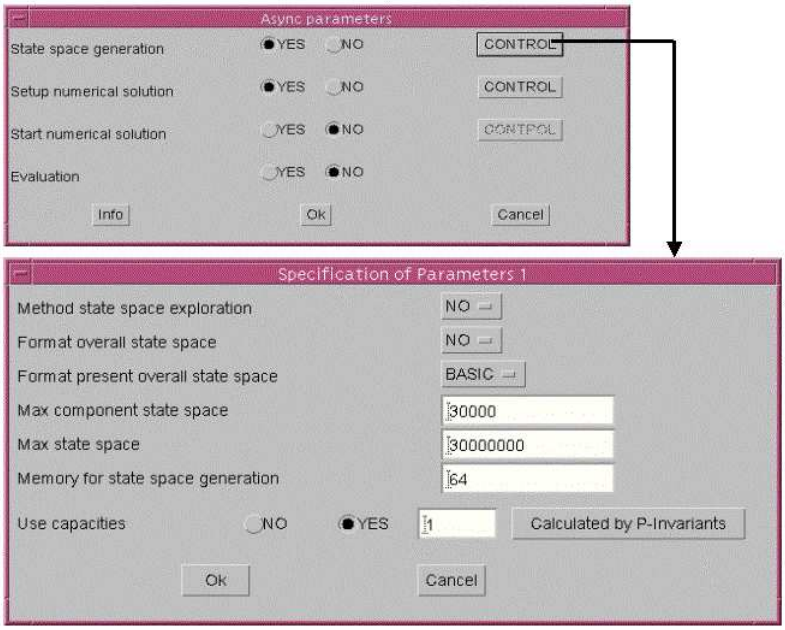

Abbildung 7.2: *APNN-Toolbox*: *ASYNC*-Ansteuerung (Generierung der Markov-Kette)

durch Shell-Skripte implementiert, die ihrerseits Ansteuerungen und Aufrufe ausführbarer Programmdateien beinhalten. Informationen, die in den Shell-Skripten benötigt werden, müssen - unterstützt durch eine graphische Benutzeroberfläche - spezifiziert werden. Im Einzelnen beinhaltet dies:

1. **Generierung Markov-Kette**: Das Shell-Skript kapselt alle Programmaufrufe zur Generierung der Markov-Kette (hierarchische Kronecker-Darstellung) aus der Spezifikation eines Petri-Netzes in der "Abstract Petri Net Notation". Es wurde von P. Buchholz und P. Kemper implementiert, vgl. Abschnitte 2.1.1, 3.2, 7.2 und 7.3. Die Abb. 7.2 zeigt unten die zugehörige Benutzerschnittstelle, über die voreingestellte Werte gewöhnlich unverändert übernommen werden können.

2. **Vorbereitung numerischer Löser**: Das Shell-Skript prüft die Vollständigkeit und Konsistenz der Dateien, die die Markov-Kette beschreiben. Das Shell-Skript erzeugt diverse Ansteuerungsdateien zur Berechnung der Lastverteilung und für den numerischen Löser und ruft benutzerdefiniert *Chaco* 2.0 oder *ParMETIS* zur Berechnung der Lastverteilung auf, vgl. Abschnitt 5.5. Desweiteren konfiguriert das Skript eines der Kommunikations-Tools (*PVM* oder *MPI*). Die Abb. 7.3 links unten zeigt die zugehörige Benutzerschnittstelle. Ein Bestandteil der Benutzerschnittstelle dient der Spezifikation der Rechner, die für die verteilte Ausführung des numerischen Lösers benutzt werden sollen, vgl. Abb. 7.3 rechts unten.

3. **Aufruf numerischer Löser**: Das Shell-Skript testet die Verfügbarkeit von *PVM* bzw. *MPI* und liest Parameter des numerischen Lösers ein. Die Belegung der Parameter erfordert teilweise eine Neuübersetzung des Quellcodes. Der Aufruf des numerischen Lösers resultiert in der Berechnung einer (approximativen) stationären Verteilung, die in einer Datei gespeichert wird. Des Weiteren werden Informationen zum Laufzeitverhalten und über den dynamischen Ablauf des Algorithmus in Dateien abgelegt, vgl. Modul `CONTROL` in Abschnitt 5.6.1.

4. **Auswertung**: Das Shell-Skript ruft ein Evaluator-Programm auf (von P. Kemper), das aus der zuvor berechneten stationären Verteilung die Durchsätze der Transitionen und die Tokenpopulationen der Stellen des Petri-Netzes ermittelt, vgl. Abschnitt 2.1.1. Des Weiteren ruft das Shell-Skript ein Programm auf, dass aus den protokollierten Informationen zum Laufzeitverhalten und zum dynamischen Ablauf des Algorithmus unter Zuhilfenahme von `gnuplot` und `LaTex` ein postscript-Dokument erzeugt, indem alle Informationen integriert und für eine ex-post Analyse geeignet aufbereitet sind.

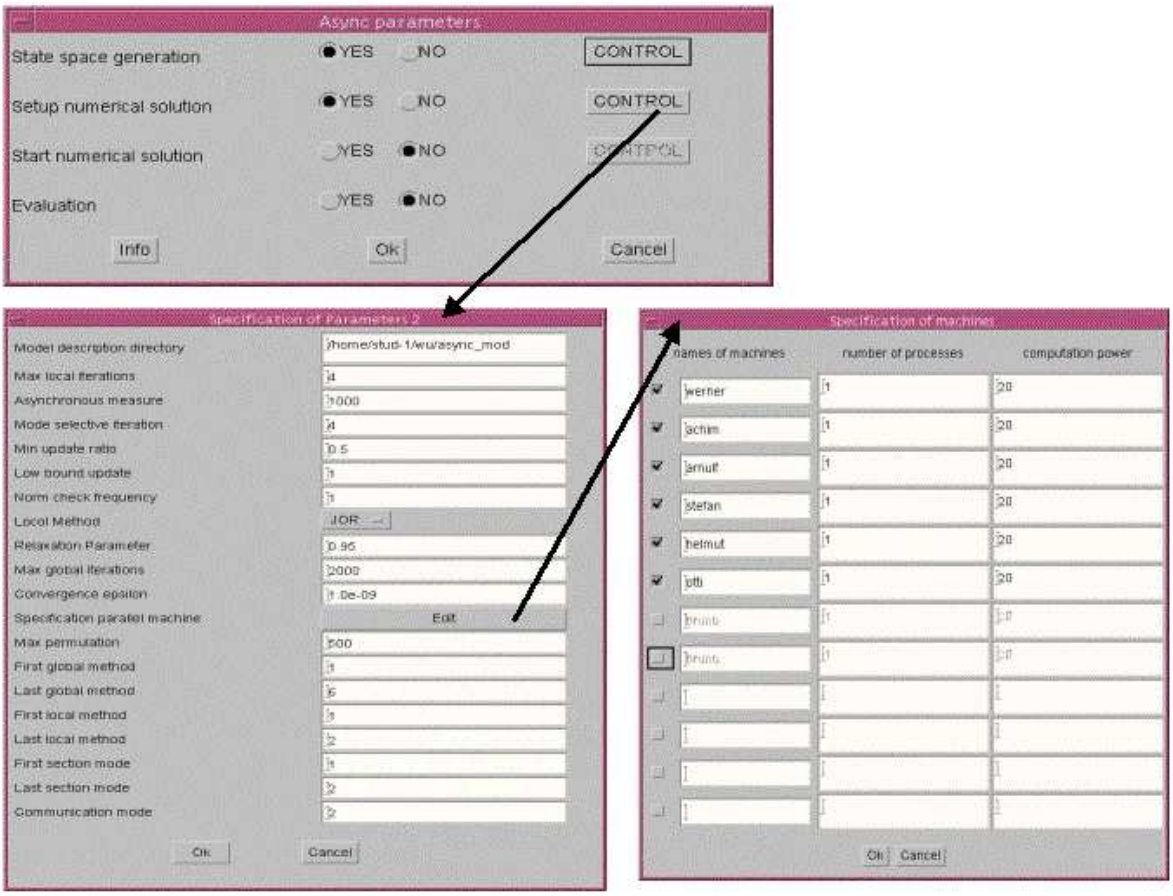

Abbildung 7.3: *APNN-Toolbox*: *ASYNC*-Ansteuerung (Vorbereitung numerischer Löser)

Kapitel 8

Modellbildung und Analyse logistischer Systeme

In diesem Kapitel wird die Anwendbarkeit und Nützlichkeit des *ASYNC*-Instrumentariums und der begleitenden Modellierungs-Software aus der Perspektive der Logistik-Domäne untersucht und bewertet. Der Ausgangspunkt ist die Interpretation logistischer Systeme als *DEDS* und deren Modellierung mit der *ProC/B*-Notation. Ein Alleinstellungsmerkmal der *ProC/B*-Notation ist, dass sie Anforderungen (prozess-orientierter) logistischer Systeme berücksichtigt, die Ableitung von formalen Performance-Modellen erlaubt und ein breites Spektrum von *DEDS*-Analyseverfahren erschließt. Insbesondere werden Markov-Ketten als analytisches Modell zur Performance-Analyse logistischer Systeme erschlossen. Das Markov-Ketten-Instrumentarium wird über den Zwischenschritt der Petri-Netz Modellwelt erreicht.
Das Kapitel ist wie folgt strukturiert: im Abschnitt 8.1 wird die Methodik der Modellbildung und Analyse logistischer Systeme eingeführt und im Abschnitt 8.2 anhand konkreter Beispiele demonstriert. Für eine Stückgut-Umschlaghalle wird der Durchsatz bedienter LKWs, die Auslastung involvierter Ressourcen und die Wahrscheinlichkeit kritischer Systemzustände für unterschiedliche Modell-Konfigurationen in einer Analyse-Serie bewertet, vgl. Abschnitt 8.2.1. Für ein Teilsystem einer Lieferkette (Ein- und Auslagerungsprozesse am Lager) werden die Attribute eines Aggregats (Fluss-äquivalenter Bediener) quantifiziert, vgl. Abschnitt 8.2.2. Schließlich werden Prozesse mit ausfallbehafteten Ressourcen betrachtet und ein parametrisiertes Zuverlässigkeitsmodell hinsichtlich der Frage bewertet, mit welcher Rate eingesetzte Ressourcen gewartet werden müssen (Modellparameter), um ihre Verfügbarkeit zu maximieren. Im Abschnitt 8.3 werden Einsichten und Erfahrungen, die bei der Modellierung und Markov-Ketten-Analyse gewonnen wurden, resümiert und diskutiert.

8.1 Methodik der Modellbildung und Analyse

Logistische Systeme Die **Logistik** beschäftigt sich mit der Gestaltung der Material-, Wert- und Informationsflüsse innerhalb unterschiedlicher Unternehmensbereiche wie Beschaffung, Lagerung, Produktion und Transport [4]. Hierbei hat sich ein **prozess-orientierter Ansatz** etabliert, der nicht auf einzelne, oft künstlich isolierte Unternehmensbereiche fokussiert, sondern **Prozesse logistischer Systeme** bereichsübergreifend betrachtet. Die Planung neuer oder die Reorganisation bestehender logistischer Systeme zielt auf eine **Optimierung logistischer Kennzahlen** wie Durchlaufzeiten, Liefertreue, Kapazitätsauslastung, Bestände und Kosten ab. Obwohl die Planung vielen Randbedingungen unterliegt (existierende, unveränderliche Strukturen, gesetzlich festgelegte Richtlinien, dokumentierte "Best-Practices") ergeben sich insbesondere bei der Dimensionierung und Disposition von Betriebsmitteln zahlreiche Freiheitsgrade bzw. Planungsszenarien. Für die Optimierung müssen Planungsszenarien bewertet werden. Eine modellbasierte Bewertung vermeidet teure Eingriffe in existierende Systeme (Messung) bzw. eine zeitaufwendige, prototypische Implementierung neuer Systeme.

Modellbildung Vorhandene Software mit graphischen Benutzeroberflächen zur Modellbildung, Fortschritte in der Analysemethodik und die Verfügbarkeit hoher Rechenkapazitäten fördern die Akzeptanz einer rechnergestützten und modellbasierten Systemanalyse mit formalen Methoden in der Industrie. In der betrieblichen Praxis ist nicht nur die modellbasierte Ermittlung logistischer Kennzahlen, sondern auch deren Optimierung bedeutsam. Gegenwärtig werden logistische Systeme oft durch statische Modelle abgebildet. Dies hat aus Sicht der Optimierungsverfahren den Vorteil, dass Auswertungen der Zielfunktion (=Analysen des statischen Modells mit variierenden Attributwerten) einfach und preiswert sind. Allerdings wird die komplexe Dynamik logistischer Systeme nicht adäquat durch statische Funktionen beschrieben. Erst dynamische Modelle erlauben eine realitätsnahe Modellbildung.
Im Vorfeld der eigentlichen Modellbildung müssen typische Systemeigenschaften identifiziert werden. Logistische Systeme sind oft als **diskrete ereignis-gesteuerte dynamische Systeme** (*DEDS*) interpretierbar, weil auftretende Größen diskret sind (Lagerbestand, Bestellmenge) und Ereignisse diskret und asynchron auftreten (Einlagerung, Auftragseingang), vgl. Kapitel 2. Die Akzeptanz dieser Interpretation wird u.a. durch das "Handbuch Logistik" [4] belegt, in dem auf bedientheoretische Ansätze und Diskrete Ereignis-gesteuerte Simulation (die beide eine *DEDS*-Interpretation voraussetzen) als gängige Techniken zur Analyse logistischer Systeme verwiesen wird. Eine *DEDS*-Interpretation ist in verwandten technischen Systemen, wie zum Beispiel Rechen- und Telekommunikationssystemen [20, 36] und Fertigungssystemen [3] weit verbreitet.
Für die Reproduktion logistischer Systeme (Lieferketten, Geschäftsprozesse etc.) in Modellen sind zahlreiche hochsprachliche Notationen bekannt, die durch eine objekt-orientierte, prozessorientierte und/oder hierarchische Methodik (mit Anleihen aus UML [69]) an Bedarfe und Denkweisen des jeweiligen Anwendungsgebiets adaptieren. UML (Unified Modeling Language) Aktivitätsdiagramme [69], EPC^{TM} (Event-driven Process Chains, IDS-Scheer) [89], Process Diagrams (Casewise Inc. und TIBCO), Rational RequisitePro^{TM} (IBM), Rational Rose Use Case Diagrams^{TM} (Rational) und einige mehr haben sich in der Praxis als Beschreibungsmittel betrieblicher Abläufe etabliert und sind in kommerziellen Tools für BPR/M (Business Process Reengineering/Management), ERP (Enterprise Resource Planning) und WFM (Workflow Management) verfügbar, vgl. etwa das ARIS-Toolset^{TM} (IDS-Scheer), MS Visio^{TM} (Microsoft) und

Corporate Modeler (Casewise Inc.). Basierend auf dem Prozessketten-Paradigma von Kuhn [72] (Fraunhofer Institut für Materialfluss und Logistik, Dortmund) ist ein MS-Visio^{TM} basiertes Tool und das *ProC/B*-Toolset (Informatik Lehrstuhl iV, Dortmund, vgl. Abschnitt 7.1) entstanden. Alle Tools ermöglichen eine graphische Modellbildung und unterstützen damit das Explizieren von Konzepten zur Planung, zum Management und zur Steuerung logistischer Abläufe. Die Ableitung eines formalen Leistungsmodells, das eine rechnergestützte Berechnung logistischer Kennzahlen ermöglicht, ist mit Ausnahme des ARIS-Toolset^{TM} (nur Simulation) und des *ProC/B*-Toolset nicht möglich. Eine Gartner-Studie[1] resümiert, dass Simulationsmodelle in Zukunft verstärkt eingesetzt werden.
Die ***ProC/B*-Notation** aus Abschnitt 7.1 ist durch Konzepte des Prozessketten-Paradigmas von Kuhn [72] inspiriert (graphische Darstellung etc), wodurch sie Anforderungen des Anwendungsfeldes berücksichtigt. Die *ProC/B*-Notation adoptiert darüber hinaus einige Konzepte des Modellierungs- und Analyse-Tools HIT von Beilner [16], wodurch eine hinreichend formale Beschreibung und eindeutige Interpretation des Modells als Voraussetzung für eine automatisierte Übersetzung in ein Performance-Modell erreicht wird. Die *ProC/B*-Notation unterstützt die Methodik der objekt- und prozessorientierten sowie hierarchischen Modellbildung. Diese Methodik wurde bereits Mitte der siebziger Jahre in HIT angeregt, sie fand aber erst durch die Verbreitung UML-basierter Modellierungstools breite Anwendung, zuerst im Informatik-Bereich (Software-Entwurf, Datenbanken), später auch im betriebswirtschaftlichen Bereich. Ein Alleinstellungsmerkmal der *ProC/B*-Notation ist, dass sie synergetisch prozessorientierte und objektflussorientierte Sichtweisen verzahnt, so dass Nachteile einer rein objektflussorientierten Sichtweise (zeitlicher Aspekt wird nur indirekt abgebildet) und prozessorientierten Sichtweise (statische Aufbaustruktur des Systems wird nur indirekt abgebildet) aufgehoben sind. Zudem realisiert die *ProC/B*-Notation gängige Konzepte der Hierarchiebildung in Struktur- und Verhaltensmodellierung, vgl. Abschnitt 7.1. Das *ProC/B*-Tool [11] ergänzt die aus HIT importierte Methodik durch eine neue graphische Modellierungsumgebung und neue syntaktische Konstrukte, die aus spezifischen Anforderungen der Logistik-Domäne resultieren. Im Unterschied zur rein deskriptiven Modellbildung, deren Intention nur das Explizieren von Informationen ist, ohne dabei implizite Logistikkennzahlen des modellierten Systems ermitteln zu können, ist die Ausdrucksmächtigkeit der *ProC/B*-Notation im Vergleich zu der von UML- Aktivitätsdiagrammen eingeschränkt. Zum Beispiel erlaubt die *ProC/B* Notation keine asynchronen Dienstaufrufe.

Analyse Eine gängige Methodik der rechnergestützten, modellbasierten Analyse ist, die manuelle Modellbildung durch eine hochsprachliche Notation wie zum Beispiel *ProC/B* zu unterstützen, die in ihrer Semantik hinreichend formal und eindeutig ist, um aus ihr automatisiert ein **analytisches Modell** ableiten zu können, das als Eingabe des Analyseverfahrens fungiert. Die *ProC/B*-Modellbildung zielt nicht auf ein spezifisches analytisches Modell bzw. Analyseverfahren ab und es existieren Übersetzer in die HIT-, Warteschlangennetzwerk- und Petri-Netz-Modellwelt [11], vgl. Abb. 1.1.
In der vorliegenden Arbeit wird lediglich die Abbildung auf *PN*s betrachtet, weil sie die Voraussetzung für die Anbindung des Markov-Ketten Instrumentariums schafft, vgl. Abschnitt 7.2. *PN*s sind als Mittler zwischen einer prozess-orientierten Notation wie *ProC/B* und Markov-Ketten gut geeignet, weil gängige Ablaufkonzepte von Prozessen (Nebenläufigkeit, Synchronisation, boolesche/probabilistische Verzweigung und Vereinigung) und deren Dynamik (geteilte Ressourcen,

[1] `http://mediaproducts.gartner.com/reprints/idsscheer/119964.html`

Indeterminismus) gut beschreibbar sind. Konzeptionell markieren Token instanziierte Prozessobjekte. *PN*-Stellen haben zwei Funktionen, sie zeigen den aktuellen Zustand der Prozessobjekte an (Aufenthaltsort, Bearbeitungsstufe etc.) und bilden den Zustand der Ressourcen ab (Verfügbarkeit etc.). Alternativ ist das Farbenkonzept der Token für die Zustandskodierung anwendbar. Dies ist insbesondere dann interessant, wenn Farben lediglich als Speicher von Informationen dienen, deren Visualisierung durch Stellen das Verständnis der Prozessabläufe behindert, z.B. bei Schleifen. Gewöhnlich existieren mehrere Token gleichzeitig, um Nebenläufigkeit, mehrfach instanziierte Prozessobjekte etc. abzubilden. *PN*-Transitionen modellieren Aktionen der Prozesse und Zustandsveränderungen der Ressourcen (Benutzung↔Freigabe, Ausfall↔Reparatur).

8.2 Anwendungsbeispiele

In den nachfolgenden Unterabschnitten werden drei Anwendungsbeispiele logistischer Systeme vorgestellt, anhand derer die Anwendbarkeit und Nützlichkeit des *ASYNC*-Instrumentariums untersucht werden soll.

8.2.1 Stückgut-Umschlaghalle

Die Stückgut-Umschlaghalle ist ein zentraler Bestandteil eines Güterverkehrszentrums (`GVZ`). Ein `GVZ` etabliert eine Schnittstelle für den Warenumschlag zwischen unterschiedlichen Verkehrsträgern (Verkehrsträgerwechsel), die Verkehrs- und Dienstleistungsunternehmen zur Gewinnung von Synergien zusammenführt, vgl. auch [4]. Ein `GVZ` ist als ein Knoten innerhalb eines größeren logistischen Netzwerks interpretierbar. Deswegen ist die Modellierung und Analyse von `GVZ`s eine wichtige Aufgabe innerhalb des SFB 559 “Modellierung großer Netze in der Logistik”. Nicht-simulative Analysetechniken sind hierfür anwendbar und nützlich, vgl. [5, 53, 54, 57].

Systembeschreibung In einer Stückgut-Umschlaghalle werden Güter von LKWs angeliefert und abgeholt und bei Bedarf zwischengelagert. Der LKW-Abfertigungsbereich ist organisatorisch in Terminals unterteilt. Jedes Terminal hat mehrere Rampen, die von LKWs für deren Abfertigung angefahren werden. Sobald ein LKW an einer von ihm belegten Rampe wartet, wird ein Gabelstapler angefordert. Ein Gabelstapler ist entweder bereits am Terminal oder er fährt von einem zentralen Gabelstapler-Pool zum Terminal, vgl. Abb. 8.1. Sobald ein Gabelstapler verfügbar ist, startet der eigentliche LKW-Umschlagprozess. Hierfür wird zusätzlich zum Gabelstapler Personal (=aktive Ressource) beansprucht, um Frachtpapiere zu prüfen und den Umschlag zu steuern und zu überwachen. Das Personal ist jedem Terminal exklusiv zugeordnet. Nach dem Umschlag verläßt der LKW die Rampe. Der benutzte Gabelstapler ist darüber hinaus noch für eine gewisse Zeit nicht verfügbar (Einsatzplanung, Fahrt zum Pool bzw. nächsten Einsatzort). Hierbei sind zwei Fälle zu unterscheiden. Der Gabelstapler bleibt im aktuellen Terminal, wenn dort bereits ein neuer LKW auf seine Abfertigung wartet, anderenfalls fährt der Gabelstapler in einen zentralen Pool, um dort auf zukünftige Einsätze zu warten. Weil die Gabelstapler-Einsatzplanung im zentralen Pool erfolgt, müssen Gabelstapler zum Pool zurück kehren und können nicht direkt ein neues Terminal anfahren. Die Einsatzplanung der Gabelstapler ist somit lastabhängig gesteuert, vgl. Abb. 8.1. Bei der Analyse des Systems wird ein System mit insgesamt 5 Terminals betrachtet. In der Abb. 8.1 sind aus Gründen der Übersichtlichkeit nur 2 Terminals dargestellt, die prinzipielle Organisation der Abläufe bleibt von der Erhöhung

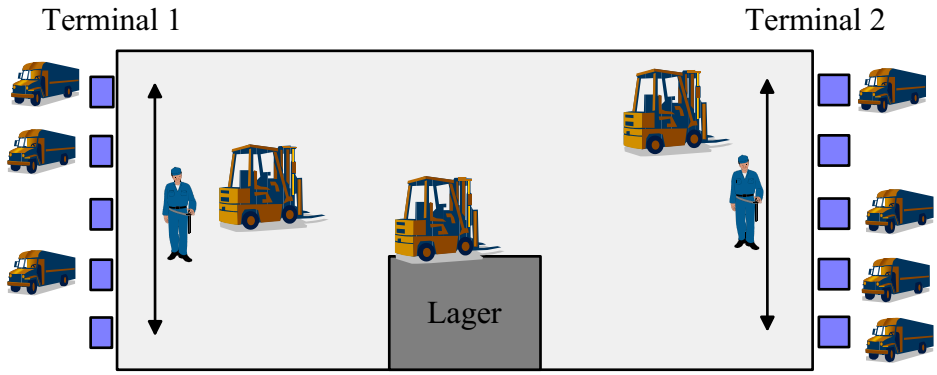

Abbildung 8.1: Struktur der Stückgutumschlaghalle mit 2 Terminals und 5 Rampen je Terminal

der Anzahl der Terminals unberührt.
Die Tab. 8.1 gibt eine Übersicht der System-Parameter. Nach Kuhn [72] sind "Ressourcen" (knappe) logistische Betriebsmittel, "Prozesse" systemübergreifende Verkettungen sachlich zusammenhängender Aktivitäten einzelner Systeme und "Strukturen" sind Anordnungen der Betriebsmittel einschließlich der Aufbau-Organisation des Systems. Systemparameter sind als Größen zu interpretieren, die a-priori bekannt sind und interessierende Zielgrößen beeinflussen. Typischerweise quantifizieren Systemparameter a-priori bekannte Zeitverbräuche und Dimensionierungen eingesetzter Ressourcen.

Ressourcen	Gabelstapler (1...7 je nach Modellvariante), Terminal (5), Rampen je Terminal (5), Personal je Terminal (1)
Prozesse	Ankunft und Abfertigung der LKWs am Terminal (siehe Abb. 8.2, Prozesse `Truck0` und `Truck1`) Gabelstapler Disposition (siehe Abb. 8.2, Prozess `ForkLift`)
Struktur	Anordnungstruktur, vgl. Abb. 8.1
Systemparameter	Dimensionierung der Betriebsmittel Fahrzeiten der Gabelstapler Be- und Entladezeit der LKWs
Systemlast	LKW-Ankunftsprozesse an Terminals
Fragestellung	Durchsatz bedienter LKWs, Auslastung Personal und Wahrscheinlichkeit kritischer Systemzustände

Tabelle 8.1: Systembeschreibung der Stückgutumschlaghalle

Modellbildung Die Stückgut-Umschlaghalle ähnelt einem komplexen Bediensystem, in dem unabhängige Ankunftströme (=LKWs an Terminals) an Bedienstationen mit endlichem Warteraum (=Rampen je Terminal) von Bedienern (Gabelstapler) verarbeitet werden, die zwischen den Bedienstationen lastgesteuert und zeitbehaftet zirkulieren. Jeder Gabelstapler-Bedienprozess beansprucht geschachtelt eine weitere aktive Ressource (=Personal). Aufgrund des komplexen

Verhaltens der Bediener und des eigentlichen Bedienprozesses ist eine analytisch-algebraische Lösung des gesamten Systems nicht möglich.
Der Modellbildung liegen folgende System-Annahmen zugrunde:

1. **Größe Lager:** Das Lager als Ressource ist ausreichend dimensioniert, und es treten keine Engpässe bzw. Wartezeiten beim Zugriff auf. Deswegen ist das Lager im Modell nicht erfasst.

2. **LKW-Ankunft** Das Ankunftsverhalten der LKWs an den Terminals ist unabhängig voneinander. Deswegen initieren die Quellen im Modell unabhängig voneinander LKW-Objekte. Die Zwischenankunftszeiten der LKWs können durch eine Exponentialverteilung approximiert werden, d.h. der LKW-Ankunftsprozess ist ein stochastischer Poisson-Prozess [20]. Wenn alle Rampen belegt sind (es sind 5 Rampen vorhanden), werden ankommende LKWs in einen separaten Bereich umgeleitet. Diese LKWs werden im Modell nicht betrachtet, weil der Durchsatz von LKWs ermittelt werden soll, die an der Stückgut-Umschlaghalle abgefertigt werden.

3. **LKW-Typen:** Alle LKWs haben die gleiche Ausprägung und werden nicht hinsichtlich ihrer Ladungsmenge, Ladungstyp, Umschlagzeit etc. unterschieden.

4. **Gabelstapler-Routing:** Gabelstapler fahren Terminals an, wenn dort mindestens ein LKW auf seine Bedienung wartet. Wenn LKWs an mehreren Terminals warten, wird ein Terminal stochastisch-gleichverteilt ausgewählt (lastabhängig-stochastisches Routing).

5. **Relation Gabelstapler-LKW:** Es wird nur ein Gabelstapler zur Abfertigung eines LKWs eingesetzt.

Die Abb. 8.2 zeigt das *ProC/B*-Modell der Stückgut-Umschlaghalle. Das Modell besteht aus 2 `Prozess-Ketten` (`PK`), die das Verhalten der LKWs an 2 Terminals beschreiben (gelbe Stränge) und eine `PK`, die das Verhalten der Gabelstapler erfasst (grüner Strang). Zwecks besserer Lesbarkeit sind in Abb. 8.3 beide `PK`s zusätzlich separat und vergrößert angegeben.
Die Gabelstapler-`PK` beinhaltet einen öffnenden, boolschen `OR-Konnektor`, der die lastabhängige Einsatzplanung der Gabelstapler an Terminals modelliert. Das `Code-PKE` vor dem `OR-Konnektor` berechnet eine boolsche Funktion für das nachfolgende boolsche Routing am `OR-Konnektor`. Hierfür wird die globale Variable T gelesen, über die die LKW-`PK`s ihre Gabelstapler-Anfragen kommunizieren. Die Variable T speichert die Anzahl wartender LKWs. Der HIT-Code [16] der `Code-PKE` und die Deklaration von T ist links unten im Modell angegeben, vgl. Abb. 8.2.
Gabelstapler- und LKW-Prozesse kommunizieren über die globale Variable T und synchronisieren an `AND-Konnektoren`. Der erste `AND-Konnektor` (links) synchronisiert die Ankunft des Gabelstaplers am Terminal mit dem Start des LKW-Umschlagprozesses. Der zweite `AND-Konnektor` (rechts) synchronisiert die Beendigung des LKW-Umschlagprozesses mit der Freigabe und der Einsatzplanung der Gabelstapler. Beide Synchronisationspunkte liegen im Rumpf einer Schleife mit boolscher Abbruchbedingung, deren Start bzw. Ende durch `Loop-PKE`s links bzw. rechts der `AND-Konnektoren` angezeigt ist. Die Schleife wird abgebrochen, wenn $T = 0$ erfüllt ist (kein LKW wartet am Terminal).
Weil die Gabelstapler unveränderlich und permanent existieren, werden sie in der `Quelle` einmalig erzeugt und verlassen nie eine Endlosschleife, deren Start direkt nach der `Quelle` und

deren Ende direkt vor der `Senke` liegt, vgl. Abb. 8.3 unten. Ebenso ist der LKW-Strom kurzgeschlossen, vgl. Abb. 8.3 oben. Dadurch wird erreicht, dass LKWs im Fall einer Überbelegung der Rampen in einen separaten Bereich abgewiesen und dort separat bearbeitet werden (nicht im Modell-Fokus).
In der Tab. 8.2 sind alle *ProC/B*-Konstrukte der LKW-`PK` in ihrer ablauflogischen Reihenfolge angegeben.

Modell-Konstrukt	Bezeichner	Bedeutung
`Quelle`	-	erzeugt einmalig (zur Zeit 0) 5 LKW-Prozesse
`PK-Interface`	Trucks	Prozessname
`Loop-PKE`	Start	äußere Schleife für permanente LKW-Prozess-Objekte
`Delay-PKE`	Arrival	modelliert Zwischenankunftszeiten der LKWs
`Code-PKE`	Register	Zähler-Inkrementierung, signalisiert Gabelstapler-Bedarf
`AND-Konnektor`	-	Synchronisation mit Gabelstapler-Prozess
`Code-PKE`	UpdateWIP	Zähler-Dekrementierung: kein Gabelstapler-Bedarf
`PKE.Server`	ReqStaff	Anforderung Personal (=aktive Ressource)
`Delay-PKE`	Handling	Verbringen der Ladung
`AND-Konnektor`	-	Synchronisation mit Gabelstapler-Prozess: Freigabe Gabelstapler
`Loop-PKE`	Fin	siehe `Loop-PKE` oben
`Senke`	-	markiert `PK` Ende

Tabelle 8.2: LKW-`Prozess-Kette` und involvierte *ProC/B*-Konstrukte

Modell-Konstrukt	Bezeichner	Bedeutung/Aktivität
`Quelle`	-	erzeugt einmalig (zur Zeit 0) 2 Gabelstapler-Prozesse
`PK-Interface`	ForkLift	Prozessname und -variablen
`Loop-PKE`	Start	äußere Schleife für permanente Gabelstapler-Prozess-Objekte
`Code-PKE`	Disposition	Routing Gabelstapler zu Terminal festlegen
`OR-Konnektor`	-	Routing Gabelstapler zu Terminal ausführen
`Delay-PKE`	Dispatch	Fahrt zum Terminal
`Loop-PKE`	Start	Start innere Schleife für mehrfache LKW-Bedienung am Terminal
`AND-Konnektor`	-	Synchronisation mit LKW-Prozess: Start LKW-Umschlag
`AND-Konnektor`	-	Synchronisation mit LKW-Prozess: Ende LKW-Umschlag
`Delay-PKE`	LocDispatch	Rüstzeit nach LKW-Bedienung
`Loop-PKE`	Fin	Ende/Fortsetzung innere Schleife für mehrfache LKW-Bedienung
`Delay-PKE`	Dispatch	Fahrt zum Pool
`Loop-PKE`	Fin	siehe `Loop-PKE` oben
`Senke`	-	markiert `PK` Ende

Tabelle 8.3: Gabelstapler-`Prozess-Kette` und involvierte *ProC/B*-Konstrukte

Ein **kritischer Systemzustand** sei die Situation, dass mindestens 4 Gabelstapler (von maximal 7) ausgehend vom Terminal 1 gleichzeitig Ladung verbringen wollen. Im *ProC/B*-Modell aus Abb. 8.2 bedeutet dies, dass mindestens 4 Prozesse gleichzeitig im `Delay-PKE` "Handling" sind. Im realen System ist dieser Zustand als kritischer Zustand einzustufen, wenn man annimmt, dass sich Gabelstapler gegenseitig behindern. Der *ProC/B*-Modellbildung liegt die Annahme zugrunde, dass Gabelstapler unabhängig voneinander Ladung verbringen und einen zustandsunabhängigen Zeitverbrauch benötigen. Deswegen ist die Modellbildung unter der Annahme des geschilderten kritischen Systemszustands eigentlich nur eine Approximation. Trotzdem soll die

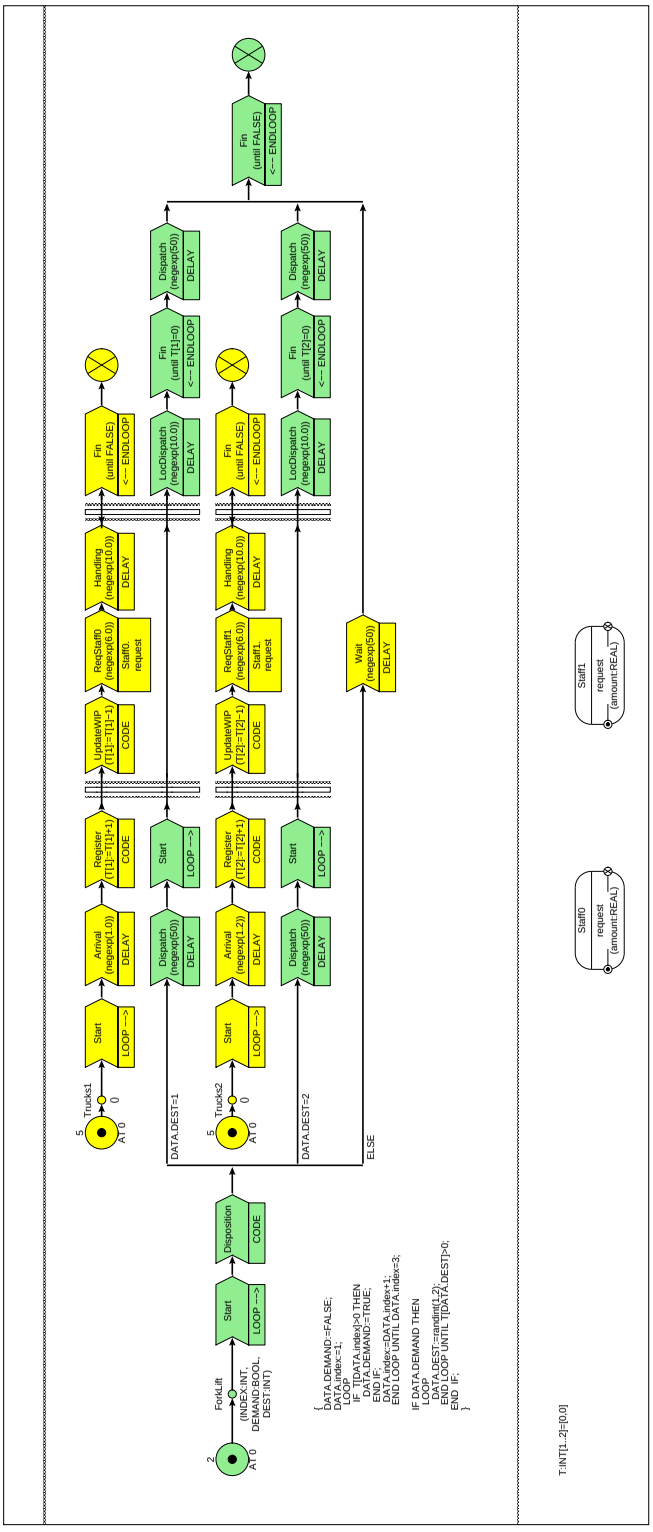

Abbildung 8.2: *ProC/B* Modell der Stückgutumschlaghalle

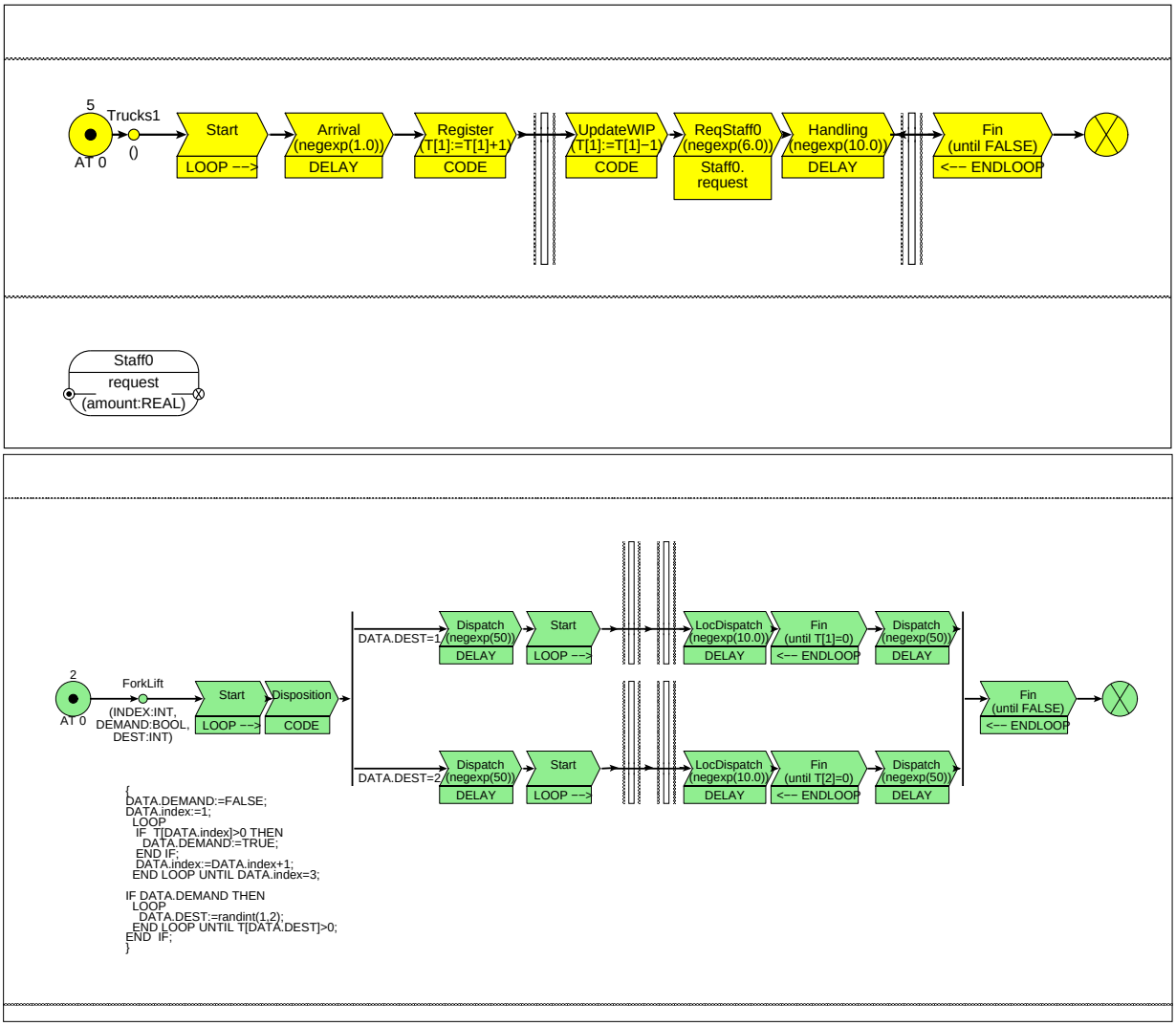

Abbildung 8.3: Prozess-Sichten des `SUH1`-Modells: LKW-`Prozess-Kette` (oben) und Gabelstapler-`Prozess-Kette` (unten)

vorliegende *ProC/B*-Modellbildung beibehalten werden und anhand dieser die Wahrscheinlichkeit für das Auftreten dieses kritischen Systemzustands bewertet werden.

Analyseergebnisse Der LKW-Durchsatz und die Personal-Auslastung jeweils in Abhängigkeit vom Terminal (Spalten 1..5) und in Abhängigkeit von der Anzahl eingesetzter Gabelstapler (Zeilen) k ist in den Tabellen 8.4 und 8.5 angegeben. Der LKW-Durchsatz und die Personal-Auslastung ist mit *ASYNC* berechnet. Die Werte in eckigen Klammern geben Konfidenzintervalle einer simulativen Auswertung an.
Die LKW-Durchsätze erreichen mit Zunahme von k die Werte $0.4, 0.5, 0.6, 0.8$ bzw. 1.0. Diese Grenzwerte sind durch die LKW-Ankunftsraten je Terminal festgelegt (im Modell vorgegeben). Der maximal mögliche LKW-Durchsatz wird bereits für $k = 4$ Gabelstapler erreicht.

k	LKW-Durchsatz an Terminals				
	1	2	3	4	5
1	0.369 [0.368,0.370]	0.444 [0.442,0.443]	0.512 [0.510,0.511]	0.634 [0.631,0.633]	0.743 [0.737,0.739]
2	0.399 [0.399,0.401]	0.499 [0.498,0.500]	0.597 [0.595,0.598]	0.792 [0.792,0.795]	0.983 [0.982,0.985]
3	0.400 [0.399,0.404]	0.499 [0.496,0.501]	0.600 [0.593,0.599]	0.798 [0.796,0.803]	0.996 [0.992,0.999]
4	0.400 [0.399,0.405]	0.500 [0.494,0.500]	0.600 [0.594,0.600]	0.799 [0.796,0.804]	0.998 [0.993,1.001]
7	0.400 [0.399,0.405]	0.500 [0.493,0.500]	0.600 [0.595,0.602]	0.800 [0.795,0.804]	0.999 [0.992,1.002]

Tabelle 8.4: LKW-Durchsatz an Terminals

k	Personal-Auslastung [%]				
	1	2	3	4	5
1	7.9	9.7	11.3	14.3	16.7
2	8.0	10.0	11.9	15.8	19.7
3	8.0	10.0	12.0	16.0	19.9
4	8.0	10.0	12.0	16.0	20.0
7	8.0	10.0	12.0	16.0	20.0

Tabelle 8.5: Personal-Auslastung an Terminals

Die Personal-Auslastung steigt erwartungsgemäß mit Zunahme von k, weil der LKW-Durchsatz steigt. Die Personal-Auslastung ist auch bei maximaler Belastung ($k \geq 4$) moderat, vgl. Tab. 8.4.
Alle Performance-Ergebnisse, die mit *ASYNC* ermittelt wurden, sind durch Petri-Netz Simulationen verifizierbar. Hierfür wurde ein Java-basierter Petri-Netz Simulator benutzt, der in der *APNN-Toolbox* bereits verfügbar ist. Die Konfidenzintervalle der simulativen Bewertung in Tab. 8.4 sind auf ein Konfidenz-Level von 90% bezogen. Die numerisch ermittelten Werte liegen im Konfidenzintervall oder, mit wenigen Ausnahmen bedingt durch die stochastische Ungenauigkeit der Simulation, knapp daneben. Sofern es die Größe des Zustandsraums zulässt, wurden auch konventionelle sequentielle numerische Löser zur Validierung der *ASYNC*-Ergebnisse benutzt.
Neben der Validierung der Performance-Ergebnisse ermöglicht die simulative Auswertung einen Effizienzvergleich der Methoden. Die Simulationszeiten schwanken zwischen 100 Sekunden und 900 Sekunden, wobei die Simulationszeit mit zunehmender Anzahl der Gabelstapler k sinkt. Demgegenüber steigen die *ASYNC*-Lösungszeiten[2] mit zunehmendem k, weil die Zustandsraumgröße steigt, vgl. Tab. 8.6. *ASYNC* schneidet im Methodenvergleich für Modelle mit $k \geq 4$ (>

k:	1	2	3	4	5	6	7
Zeit [s] *ASYNC*:	17	1 025	412	1 606	6 089	18 650	63 560

Tabelle 8.6: *ASYNC*-Lösungszeiten

20 Millionen Zustände) trotz verteilter Lösung deutlich schlechter ab. Die Lösungszeiten liegen deutlich über denen der Simulation (100 Sekunden - 900 Sekunden). Eine simulative Auswertung der LKW-Durchlaufzeiten und Gabelstapler-Auslastungen ist effizienter, weil Ereignisse, die zur Bewertung dieser Performance-Kennzahlen beitragen, hochfrequent auftreten.

[2]Für k=1,2 wurden 2 *Worker*-Prozesse und für k=3,..,7 wurden 6 *Worker*-Prozesse verwendet.

Erfahrungsgemäß können numerische Löser mit der Simulation konkurrieren, wenn seltene Ereignisse wie zum Beispiel kritische Systemzustände (Paket-Verlustraten in mobiler Kommunikation oder Ausfall von Ressourcen) zu bewerten sind. Hierfür sei $X(t)$ die Anzahl von Gabelstaplern, die zum Zeitpunkt t ausgehend vom Terminal 1 gleichzeitg Ladung verbringen. Oben wurde erläutert, warum $X(t) \geq 4$ einen kritischen Systemzustand beschreibt. Das Ziel ist die stationäre Wahrscheinlichkeit $\lim_{t\to\infty} P[X(t) \geq 4]$ für einen kritischen Systemzustand zu bewerten. In Tab. 8.7 sind die durch *ASYNC* ermittelten Wahrscheinlichkeiten für $k = 4 \ldots 7$ eingesetzte Gabelstapler angegeben. Erwartungsgemäß steigen die Wahrscheinlichkeiten mit zunehmender Anzahl eingesetzter Gabelstapler.

k	0..3	4	5	6	7
$\lim_{t\to\infty} P[X(t) \geq 4]$	0	3.526×10^{-8}	6.406×10^{-8}	7.994×10^{-8}	8.804×10^{-8}

Tabelle 8.7: Wahrscheinlichkeiten für das Auftreten kritischer Systemzustände

Für alle k-Werte wurden *PN*-Simulationen durchgeführt und nach 15000 Sekunden abgebrochen. Nach 15000 Sekunden konnte für alle k-Werte noch kein vertrauenswürdiges Konfidenzintervall ermittelt werden. Demgegenüber liefert *ASYNC* mit 2 *Worker*-Prozessen exakte Resultate in deutlich weniger als 15000 Sekunden für Modellkonfigurationen mit $k \leq 5$, vgl. Tab. 8.6.

8.2.2 Aggregat-Berechnung für Lieferketten

Dekomposition und Aggregierung (DA) [42] ist ein bekannter Ansatz, um den Aufwand einer modellbasierten Analyse technischer Systeme zu reduzieren. Die grundlegende Idee hierbei ist, dedizierte Teilmodelle in Isolation von ihrer Umgebung vorzuanalysieren, daraus Attribute einer vereinfachten Beschreibung abzuleiten (=Aggregat) und schließlich Teilmodelle durch ihre Aggregate zu ersetzen. DA zielt daruf ab, die Dynamik des Gesamtmodells zu vereinfachen (kleinere Zustandsräume), um Simulationszeiten zu verkürzen oder zustandsraumbasierte Analysen zu ermöglichen, oder die Struktur derart zu modifizieren, dass sie einer analytisch-algebraischen Analyse zugänglich ist (keine Synchronisationsmechanismen). Der Preis der Modellreduktion ist, dass ein Aggregat nur bestimmte Eigenschaften des Teilmodells exakt imitiert. Somit wird in der Interaktion des Aggregats mit seiner Umgebung in diese ein Aggregierungsfehler induziert, der durch die vereinfachte Konstruktion des Aggregats verursacht ist und Analyseergebnisse verfälschen kann.

Ein weit verbreiteter Aggregat-Typ ist der Fluss-äquivalente Bediener. Seine Konstruktion basiert auf der Annahme, dass die Aufenthaltsdauer einer Entität (Kunde, Prozessobjekt etc) im Teilmodell nur von der aktuellen Entität-Population im Teilmodell abhängt. Es abtrahiert vom Aufenthaltsort und vom Zustand der Entitäten. In die Sprache der *ProC/B*-Welt übersetzt bedeutet dies: die Dauer eines `FE`-Dienstaufrufs bzw. der Fortschritt eines Prozess-Objekts durch die `PK` hängt nur von der Anzahl der Prozess-Objekte ab, die aktuell in der `PK` residieren, der genaue Aufenthaltsort der Prozess-Objekte in der `FE` (z.B. im `PKE` oder am `Konnektor`) wird ignoriert. Die in der *ProC/B*-Notation vorliegende hierarchische Strukturierung der Modelle in Teilmodelle (`FE`), deren Interaktion über wohldefinierte Schnittstellen beschrieben ist (`FE`-Dienste), unterstützt den DA-Ansatz sehr gut.

Nachfolgend wird die numerische Voranalyse eines *ProC/B*-Teilmodells und die Ableitung eines "Fluss-äquivalenter Bediener"-Aggregats demonstriert. Das Ziel der Voranalyse ist es, seriell für

alle Prozess-Objekt-Populationen des *ProC/B*-Teilmodells mittlere Durchlaufzeiten zu ermitteln. Rein technisch ist ein Fluss-äquivalenter Bediener vollständig durch Funktionen beschrieben (je `FE`-Dienst), die Prozess-Objekt-Populationen auf Bediengeschwindigkeiten abbildet. Hierfür wird das stationäre Verhalten der Teilmodelle im Kurzschluss betrachtet. Kurzschluss heisst, dass terminierte Prozess-Objekte zeitlos an den Startpunkt im *ProC/B*-Teilmodell gelenkt werden, wodurch eine konstante Population erreicht wird. In die Sprache der *ProC/B*-Welt übersetzt bedeutet dies: es wird eine rückführende Verbindung zwischen `Senke` und `Quelle` implementiert.

Systembeschreibung Lieferketten (engl. supply chains) beschreiben alle Aktivitäten, die zwischen einer Kundenbestellung und daraufhin gelieferten Produkten, Dienstleistungen oder Informationen auftreten. Dies umfasst Prozesse involvierter Akteure wie Zulieferer (Beschaffung von Rohmaterialien oder vorgefertigter Teilprodukte), Produzenten (Produktion, Zusammenbau), Zwischenhändler (Lagerhaltung) und Händler (Auftragsbearbeitung, Verkauf) und deren Verknüpfung und Steuerung durch Transport- und Lenkungsprozesse [4]. Hybride, auf Markov-Ketten (Aggregat-Berechnung) und Warteschlangen basierende quantitative Bewertungen von Lieferketten sind in [6] dokumentiert. Nachfolgend wird lediglich die Lagerhaltung innerhalb einer Lieferkette betrachtet und für diese ein Aggregat berechnet.
Das zu aggregierende Teilsystem besteht aus einem Ein- und einem Auslagerungsprozess. Beide Prozesse beanspruchen jeweils simultan die Ressourcen "Personal" und "Gabelstapler". Die Anzahl der ein- bzw. auszulagernden Teile kann variieren. Die Tab. 8.8 gibt eine Übersicht der System-Parameter.

Ressourcen	Gabelstapler und Personal
Prozesse	Ein- und Auslagerung, vgl. Abb 8.4
	Prozesse `Put` und `Get`
Struktur	keine (es wird nur ein Teilsystem betrachtet)
Systemparameter	Anzahl Ein- und Auslagerungsprozesse
Systemlast	Rate und Umfang der Ein- und Auslagerungsprozesse
Fragestellung	mittlere Dauer Ein- und Auslagerungsprozesse

Tabelle 8.8: Systembeschreibung Ein- und Auslagerung

Modellbildung Die Abb. 8.4 zeigt das *ProC/B*-Modell mit zwei `Prozess-Ketten` (`PK`) jeweils eine für die Einlagerung (oben) und für die Auslagerung (unten). Der Modellbildung liegen folgende System-Annahmen zugrunde:

1. **Größe und Bestand Lager:** Das Lager als Ressource ist ausreichend dimensioniert und es treten keine Engpässe bzw. Wartezeiten beim Zugriff auf. Deswegen ist das Lager bzw. sein Bestand im Modell nicht erfasst.

2. **Start Ein- bzw. Auslagerungsprozess:** Ein- und Auslagerungen treten unabhängig voneinander auf. Deswegen initieren die Quellen im Modell unabhängig voneinander Prozess-Objekte. Die Zwischenankunftszeiten der Ein- und Auslagerungen können durch eine Exponentialverteilung approximiert werden, d.h. der Ankunftsprozess ist ein stochastischer Poisson-Prozess [20].

3. **Ladungsmenge:** Die variable Ladungsmenge ist durch eine (diskrete) geometrische Verteilung [20] approximierbar. Im Modell in Abb. 8.4 ist die Abbruchbedingung der Schleife probabilistisch und die Anzahl der stochastisch-vielen Schleifendurchläufe modelliert die Ladungsmenge. In der Put-`PK` ist die Abbruchbedingung der Schleife mit der Wahrscheinlichkeit 0.8, in der Get-`PK` mit der Wahrscheinlichkeit 0.6 erfüllt.

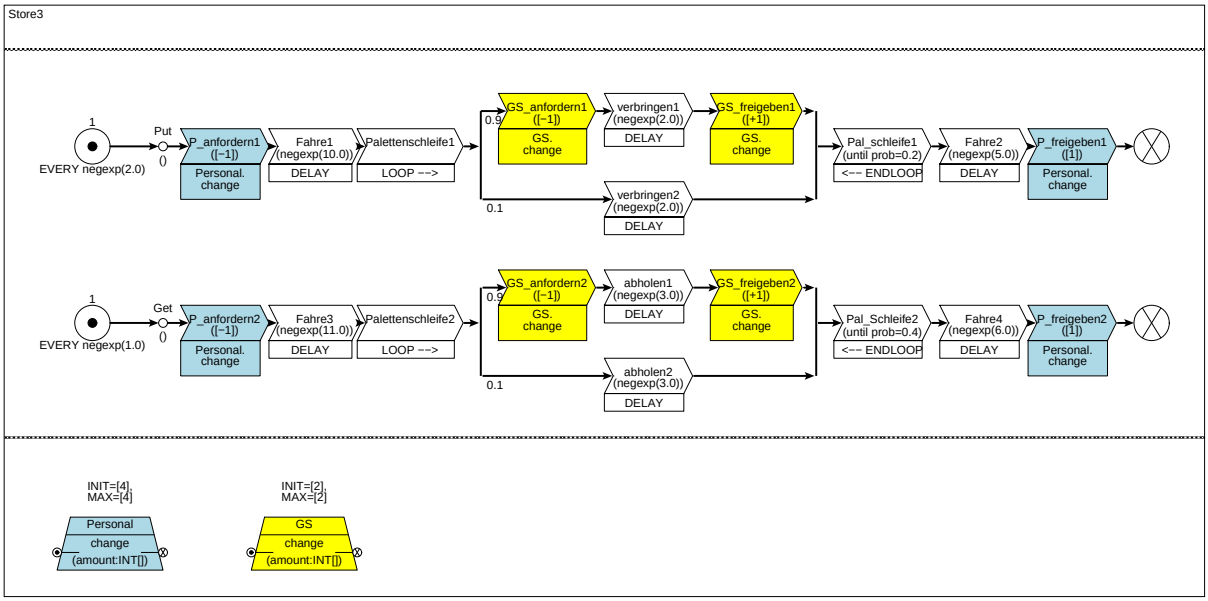

Abbildung 8.4: *ProC/B* Modell Ein- und Auslagerung

In der Tab. 8.9 sind alle *ProC/B*-Konstrukte der LKW-`PK` in ihrer ablauflogischen Reihenfolge angegeben.
Die Abb. 8.5 zeigt das Petri-Netz aus der *APNN-Toolbox*, das automatisiert aus dem *ProC/B*-Modell in Abb. 8.4 generiert wurde. Das Petri-Netz hat insgesamt 32 Stellen und 64 Transitionen (28 sind sichtbar). An diesem Beispiel wird der Vorteil einer automatisierten Abbildung deutlich. Eine direkte Modellierung eines fehlerfreien Petri-Netz Modells dieser Größe dauert 1-2 Stunden. Demgegenüber ist das (relativ kleine) *ProC/B*-Modell aus Abb. 8.4 gut in 20 Minuten erstellbar und kann binnen Sekunden automatisiert in ein Petri-Netz übersetzt werden.

Analyseergebnisse Die Abb. 8.6 zeigt die mittlere Dauer der Einlagerung (links) und der Auslagerung in Abhängigkeit ausgewählter Prozess-Objekt Populationen (hier von $(1,1)$ bis $(5,5)$. Erwartungsgemäß steigt die Dauer für beide Prozesse mit Zunahme der Population (=Last). Aus den visualisierten Werten in Abb. 8.6 sind die populationsabhängigen Bedienraten des Fluss-äquivalenten Bedieners (= Attribute des Aggregats) nach dem "Gesetz von Little" direkt ableitbar. Das berechnete Aggregat des *ProC/B*-Modells für die Ein- und Auslagerung kann in ein übergeordnetes Modell eingebettet werden.

Modell-Konstrukt	Bezeichner	Bedeutung
`Quelle`	-	erzeugt Einlagerungs-Prozesse
`PK-Interface`	Put	Prozessname
`PKE.Counter`	P_anfordern	Personal anfordern
`Delay-PKE`	Fahre1	Personal fährt zum Einsatzort
`Loop-PKE`	Palettenschleife1	Beginn Schleife für Ladungsmenge
`OR-Konnektor`	-	10% der Ladungsmenge benötigt keinen Gabelstapler
`PKE.Counter`	GS_anfordern1	Gabelstapler anfordern
`Delay-PKE`	verbringen1/2	Ladung einlagern
`PKE.Counter`	GS_freigeben1	Gabelstapler freigeben
`OR-Konnektor`	-	s.o.
`Loop-PKE`	Pal_Schleife1	Ende Schleife für Ladungsmenge
`Delay-PKE`	Fahre2	Personal verläßt Einsatzort
`PKE.Counter`	P_freigeben1	Personal frei geben
`Senke`	-	markiert `PK` Ende

Tabelle 8.9: Einlagerung-`Prozess-Kette` und involvierte *ProC/B*-Konstrukte

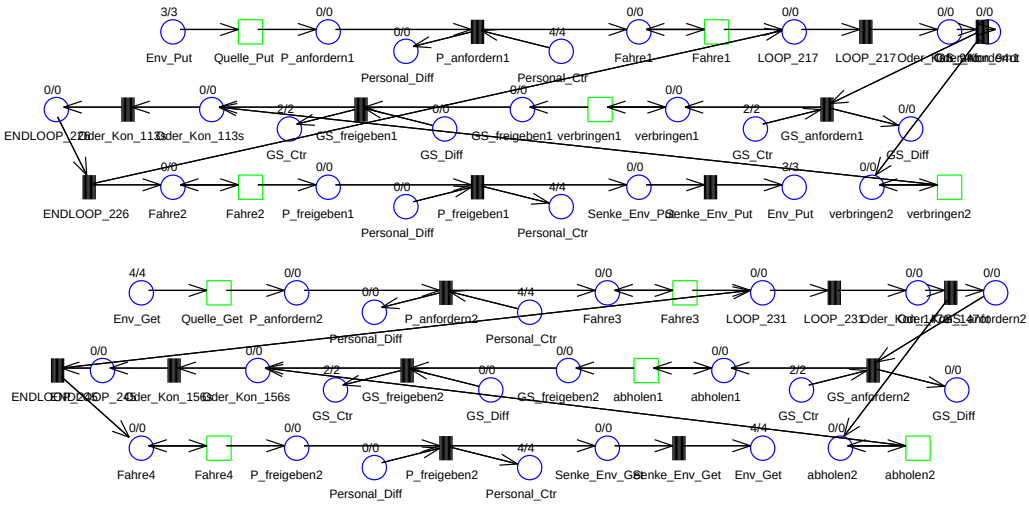

Abbildung 8.5: Petri-Netz Modell Ein- und Auslagerung

8.2.3 Prozesse mit ausfallbehafteten Ressourcen

Die Performance logistischer Systeme ist abhängig von der Last, die den Systemen durch ihre Umwelt auferlegt wird. Die Performance ist ebenfalls abhängig von der Anzahl verfügbarer Ressourcen, die zur Verarbeitung der Last beansprucht werden. In vielen technischen Systemen sind Ressourcen ausfall-, ersetzungs- und wartungsbedingt nicht permanent verfügbar.
Der Ansatz der "Performability"-Modellierung [67] berücksichtigt diesen Umstand und zielt darauf ab, implizite Performance-Charakteristika im Zusammenspiel mit impliziten Zuverlässigkeits-Charakteristika modellbasiert zu bewerten. Das Akronym "Performability" deutet an, dass

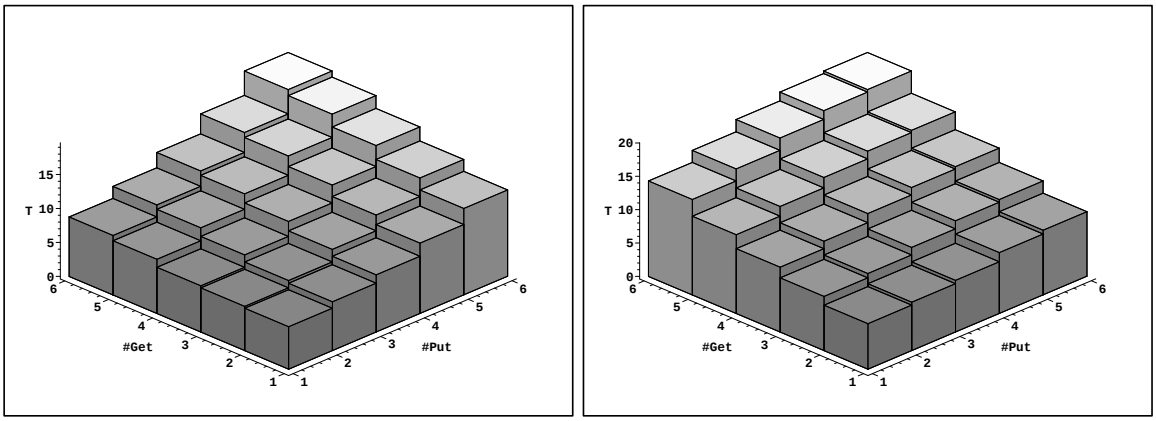

Abbildung 8.6: Mittlere Aufenthaltszeit Ein- und Auslagerungsprozesse

Aspekte der "Depend*ability*" (Zuverlässigkeit) und der "*Perform*ance" integrativ untersucht werden. Das Gesamtmodell ist in zwei sich wechselseitig beeinflussende Modelle unterteilbar: ein *Performance-Modell* (P-Modell) und ein *Zuverlässigkeits-Modell* (Z-Modell).

Die Modell-Interaktion drückt sich darin aus, dass die Performance R_c von der Ressourcen-Verfügbarkeit $c \in \mathcal{C}$ abhängt. Die Menge $\mathcal{C}$ beschreibt alle möglichen Ressourcen-Konfigurationen. Jede Konfiguration c kodiert eindeutig, welche Ressourcen verfügbar sind und welche nicht. Wenn beispielsweise Gabelstapler an 4 verschiedenen Orten agieren, beschreibt $c = (2, 2, 2, 10)$ die Konfiguration, in der an den ersten drei Orten 2 Gabelstapler und am vierten Ort 10 Gabelstapler verfügbar sind. Umgekehrt hängt die Verfügbarkeit auch von der Systemlast ab, die im P-Modell definiert ist. Beispielsweise steigt die Ausfallrate benutzter Maschinen mit erhöhter Beanspruchung. Die wechselseitige Beeinflussung drückt sich somit im zeitlichen Verhalten aus.

Die Kopplung von P- und Z-Modellen führt zu sehr großen Zuständsräumen des "Performability"-Modells, weil gewöhnlich die Größen der beiden Teilmodelle faktoriell eingehen. Die wechselseitige Beeinflussung drückt sich im zeitlichen Verhalten, nicht aber in eingeschränkter Erreichbarkeit von Zuständen aus. Die Zustände des Z-Modells (= Ressourcen-Konfigurationen) sind weitestgehend unabhängig von den Zuständen des P-Modells erreichbar. Beispielsweise kann eine Maschine gewöhnlich in jedem Zustand der "beanspruchenden Umgebung" (P-Modell) ausfallen und in einen Zustand "nicht verfügbar" wechseln. Umgekehrt sind die Zustände des P-Modells (= Aufenthaltsort, Bearbeitungsstufe etc. der Prozessobjekte) weitestgehend unabhängig von den Zuständen des Z-Modells. Solange der Zustand des Z-Modells die prinzipielle Operationalität der Prozessabläufe nicht einschränkt, bleiben alle Zustände des P-Modells erreichbar, lediglich das zeitliche Verhalten (Prozess-Dauer etc.) variiert.

Die Größe des Performability-Modell-Zustandsraums macht eine numerische, zustandsraumbasierte Analyse unmöglich. Zeitskalenunterschiede zwischen P- und Z-Modell rechtfertigen eine Analyse, die auf einer Zeitskalen-Dekomposition basiert. Die Zeitskalen-Dekomposition beruht auf der Annahme, dass Transitionen zwischen Ressourcen-Konfigurationen (Fehler treten selten auf, Wartungsraten sind gering) hinreichend lang sind im Verhältnis zu Transitionen im P-Modell (Bedienraten an den Ressourcen). Unter dieser Annahme erreicht das System zwischen Ressourcen-Konfigurationen Transitionen einen quasi-stationären Zustand. Somit kann für alle Ressourcen-Konfigurationen $c \in \mathcal{C}$ unabhängig voneinander eine stationäre Performance R_c ermittelt werden. Typischerweise werden P-Modelle simulativ oder analytisch-algebraisch

hinsichtlich R_c bewertet. Anschließend werden die bedingten stationären Performance-Maße R_c zu einer unbedingten stationären Performance-Kennzahl $R = \sum_{c \in \mathcal{C}} p_c \cdot R_c$ aggregiert. Hierbei ist p_c die stationäre Wahrscheinlichkeit für die Ressourcen-Konfiguration c. Unter der vereinfachenden Annahme, dass das zeitliche Verhalten des Z-Modells unabhängig vom P-Modell ist, sind die p_c-Werte durch eine stationäre Analyse des Z-Modells ermittelbar. Die Anwendung von Markov-Ketten ist im Kontext der Zuverlässigkeitsanalyse weit verbreitet, vgl. etwa [67].
Nachfolgend wird ein einfaches Z-Modell für Gabelstapler innerhalb eines Lagers vorgestellt und numerisch analysiert. In [12] wird dieses Z-Modell im Zusammenspiel mit einer simulativen Auswertung eines *ProC/B*-Performance-Modells betrachtet und insbesondere untersucht, wie der Aufwand der Simulation durch eine Steuerung basierend auf den p_c-Werten reduziert werden kann.

Systembeschreibung Die Abbildung 8.7 zeigt eine Lager-Organisation im Zusammenspiel mit einer Produktions-Linie. Die schematische Darstellung der Abläufe soll verdeutlichen, dass

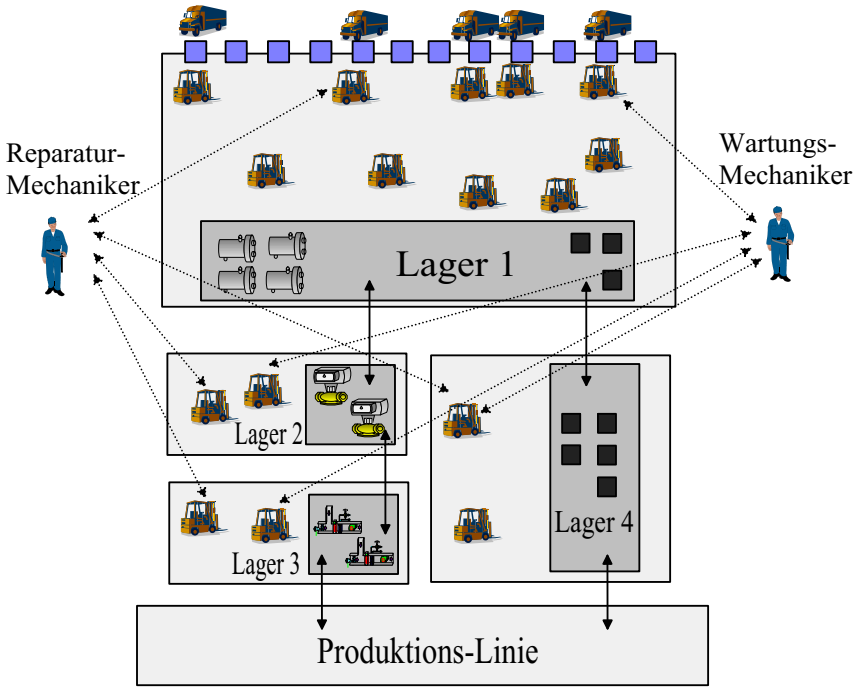

Abbildung 8.7: Lagerstruktur mit 4 Puffern und 10 bzw. 2 Gabelstaplern je Puffer

ein- und ausgehende Material- bzw. Güterflüsse über 4 Lager abgewickelt werden. Jedem Lager ist eine feste Anzahl ausfallbehafteter Gabelstapler statisch zugeordnet, vgl. Abb. 8.7. Ein Reparatur-Mechaniker ist für alle Gabelstapler zuständig und repariert ausgefallene Gabelstapler. Ein Wartungs-Mechaniker ist ebenfalls für alle Gabelstapler zuständig und führt in vorab definierten Zeitabständen Wartungen an Gabelstaplern durch. Nach einer Wartung ist die zu erwartende Zeit bis zum Auftreten eines Fehlers "zurückgesetzt" (Annahme: perfekte Wartung) und während der Wartung sind Gabelstapler nicht betriebsbereit.
Die Tab. 8.10 zählt die System-Parameter auf, die Zuverlässigkeitsaspekte betreffen. Die Frage-

stellung lautet, welche Wartungsrate hinsichtlich der Verfügbarkeit der Gabelstapler optimal ist. Eine hohe Verfügbarkeit der Gabelstapler ist notwendig für performante Material- und Güterflüsse (=Prozesse im P-Modell). Ein hoher Wartungsaufwand vermindert die Wahrscheinlichkeit für fehlerbedingte Nichtverfügbarkeit, allerdings auf Kosten einer erhöhten Nichtverfügbarkeit verursacht durch Wartung. Insofern ist die Fragestellung nicht trivial. Des Weiteren soll die Auslastung der Mechaniker in Abhängigkeit von der Wartungsrate ermittelt werden.

Ressourcen	Reparatur-Mechaniker, Wartungs-Mechaniker
Prozesse	Gabelstapler-Zyklen:
	verfügbar → in Reparatur → verfügbar
	verfügbar → in Wartung → verfügbar
Struktur	Anordnungstruktur vgl. Abb. 8.7
Systemparameter	Anzahl Mechaniker
	Ausfallzeit, Reparaturzeit und Wartungszeit für Gabelstapler
	Wartungsrate
Systemlast	Wartungsrate und Ausfallzeit
Fragestellung	Auslastung Wartungs- und Reparatur-Mechaniker sowie
	Verfügbarkeit Gabelstapler jeweils in Abh. von Wartungsrate

Tabelle 8.10: Systembeschreibung der Lagerhaltung aus Abb. 8.7

Modellbildung Die *ProC/B*-Schnittstelle etabliert eine Schnittstelle zur Modellierung und Analyse logistischer Systeme, vgl. Kapitel 7. In den vorherigen Abschnitten 8.2.1 und 8.2.2 wurde demonstriert, wie ausgehend von händisch-erstellten *ProC/B*-Modellen, Petri-Netze bzw. Markov-Ketten automatisch erzeugt werden können und somit *ASYNC* anwendbar wird. In leichter Abwandlung dieser Methodik wird nun ein Z-Modell direkt als Petri-Netz modelliert, weil die *ProC/B*-Notation für die Beschreibung von Zuverlässigkeitsaspekten defizitär ist. Dennoch hat die *ProC/B*-Notation eine große Bedeutung bei der Modellierung des Systems aus Abb. 8.7, sie beschränkt sich aber auf die P-Modellbildung (hier nicht betrachtet, vgl. [12] für Details). Bevor die Defizite näher erläutert werden, soll das Petri-Netz Z-Modell vorgestellt werden, vgl. Abb. 8.8 (von F. Bause aus [12]) und Abb. 8.9. In der Modellbildung wird angenommen, dass die Ausfallzeit (=Zeit bis zum Auftreten eines Fehlers), die Reparaturzeit und die Wartungszeit durch je eine Erlang-2 Verteilung [20] beschreibbar sind. Das Modell besteht aus 3 Teilen (von unten nach oben):

Verfügbarkeit und Reparatur (Stellen $p_1, \ldots, p_6$, Transitionen $t_1, \ldots, t_5$) Die Stellen p_1 und p_2 modellieren verfügbare Gabelstapler und die Transitionen t_1 und t_2 modellieren die Ausfallzeit. Ein Token auf der Stelle p_3 zeigt einen ausgefallenen Gabelstapler an. Ein Token auf der Stelle p_6 zeigt einen verfügbaren Reparatur-Mechaniker an. Die Transitionen t_4 und t_5 modellieren die Reparaturzeit.

Wartungsrate und -Triggerung (Stellen $p_{11}, \ldots, p_{13}$, Transitionen $t_9, \ldots, t_{13}$) Die Transitionen t_9 und t_{10} modellieren die Wartungsrate. Ein Token auf der Stelle p_{13} aktiviert die Transitionen t_{11} und t_{12}, die in Nullzeit alle Token der Stellen p_1 und p_2 auf die Stelle p_7 verschieben. Ein neuer Wartungszyklus beginnt (Transition t_{13} ist aktiviert), sobald die Markierung der Stellen p_1 und p_2 leer ist (alle in Wartung).

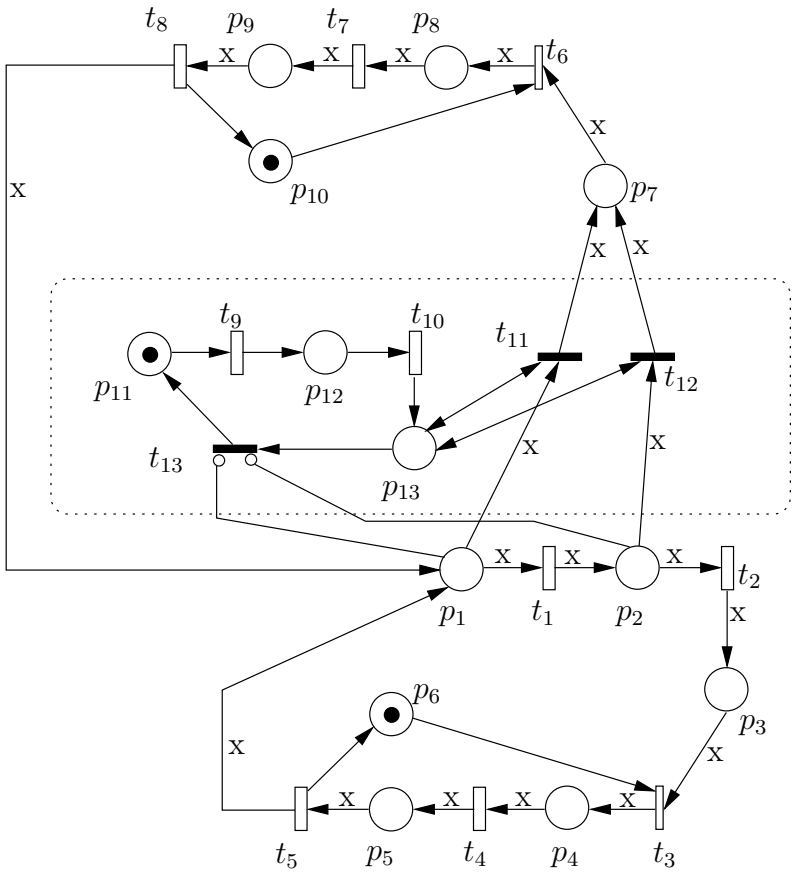

Abbildung 8.8: Petri-Netz Zuverlässigkeitsmodell

Ausführung der Wartung (Stellen $p_7, \ldots, p_{10}$, Transitionen $t_6, \ldots, t_8$) Die Token auf der Stelle p_7 repräsentieren Gabelstapler, die gewartet werden, sobald ein Wartungs-Mechaniker verfügbar ist. Ein Token auf der Stelle p_{10} zeigt einen verfügbaren Wartungs-Mechaniker an. Die Wartungszeit wird durch die Transitionen t_7 und t_8 modelliert.

Das Modell ist wie folgt quantifiziert: die Ausfallzeit (Transitionen t_1 und t_2) beträgt 20000 Minuten, die Reparaturzeit 2000 Minuten und die Wartungszeit 80 Minuten (jeweils Mittelwerte einer Erlang-2-Verteilung). Wie bereits oben angemerkt, erfasst das Z-Modell Gabelstapler an 4 verschiedenen Lokationen mit 10 (Lager 1) und 2 (Lager 2 bis Lager 4) Gabelstaplern.

Das Ausfall-, Wartungs- und Reparaturverhalten ist für alle Gabelstapler identisch. Deswegen kann es kompakt durch ein farbiges Petri-Netz modelliert werden. Die Kanten-Annotation "x" in Abb. 8.8 zeigt farbige Stellen-Markierungen und korrespondierende Transitions-Modi an.

Die Abb. 8.9 zeigt das Petri-Netz, wie es in der *APNN-Toolbox* modelliert ist. Das Petri-Netz in Abb. 8.9 ist trotz einer veränderten graphischen Darstellung (keine Farben, zusätzliche Mess-Stellen und "fusion"-Stellen) verhaltensäquivalent zum Petri-Netz in Abb. 8.8. Das Petri-Netz hat insgesamt 37 Stellen und 62 Transitionen (30 Transitionen sind sichtbar). In Abb. 8.9 sind 4 strukturell identische Teilnetze zu erkennen. Sie erfassen das Verhalten der Gabelstapler an den Lagern. Links oben ist ein kleines Teilnetz, das die Wartungsrate modelliert. Die Transitionen `Sync1` bis `Sync4` synchronisieren betriebsbereite Gapelstapler mit dem Start der Wartungsphase.

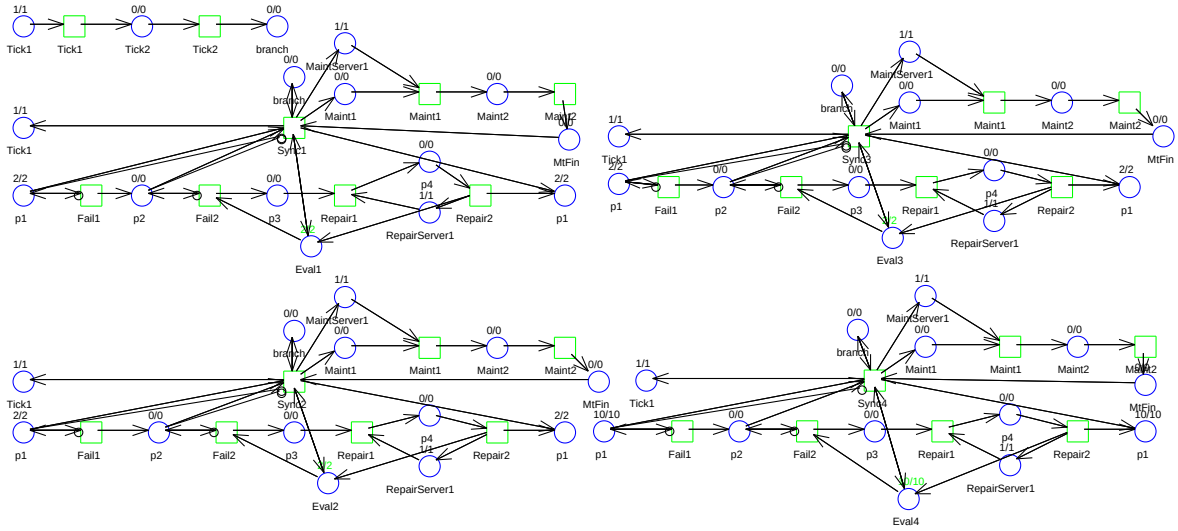

Abbildung 8.9: Petri-Netz Zuverlässigkeitsmodell aus *APNN-Toolbox*

Die Token-Population in den Mess-Stellen `Eval1` bis `Eval4` repräsentiert die Verteilung verfügbarer Gabelstapler am Lager. Die Auslastung der Mechaniker ist an den Stellen `MaintServ1` bzw. `RepairServer1` messbar.

Bestimmte Eigenschaften des Petri-Netzes können mit der *ProC/B*-Notation nicht "direkt" abgebildet werden. Im Petri-Netz in Abb. 8.8 haben die Transitionen t_1 und t_{11} sowie t_2 und t_{12} gemeinsame Eingangs-Stellen p_1 bzw. p_2. Es existieren Petri-Netz Markierungen, in denen die Transitionen t_1 und t_{11} bzw. t_2 und t_{12} simultan aktiviert sind (Petri-Netz Terminologie: es treten Konflikte auf). Im vorliegenden Petri-Netz wird der Konflikt durch den Umstand aufgehoben, dass die zeitlosen Transitionen t_{11} und t_{12} per Definition höherpriorisiert sind. Dieses Konstrukt garantiert, dass alle betriebsbereiten Gabelstapler gewartet werden. Implizit sind jedoch nur zeitbehaftete Transitionen in den (asymmetrischen) Konflikt involviert, denn aktivierte Transitionen t_1 bzw. t_2 sind in Markierungen deaktiviert, die einer Feuerung von t_{10} (ebenfalls zeitbehaftet) nachfolgen. Derartige Konflikte mit zeitbehafteten Transitionen bzw. allgemein mit zeitbehafteten Aktivitäten können nicht "direkt" durch *ProC/B*-Konstrukte abgebildet werden. Ein `OR-SPLIT-Konnektor` bildet zwar einen Konflikt in der Auswahl einer Prozessverzweigung ab, die Auswahl ist jedoch zwingend zeitlos. Das Verhalten der restlichen *ProC/B*-Konstrukte beschreibt keine Konfliktsituationen. Dies ist auch daran zu erkennen, dass korrespondierende verhaltensäquivalente Petri-Netz Modelle keine Transitionen mit gemeinsamen Eingangs-Stellen besitzen. Mit `CODE-PKE`s können textuell HIT-Befehle in *ProC/B*-Modelle eingefügt werden. Dadurch ist es prinzipiell möglich, auch zeitbehaftete Konflikte zu modellieren, allerdings schränkt dieses Vorgehen die automatisierte Abbildbarkeit von *ProC/B*-Modellen auf Petri-Netz Modelle ein. Der Übersetzer kann originäre *ProC/B*-Konstrukte auswerten und übersetzen, die ausdrucksstarke HIT-Syntax wird jedoch nur sehr eingeschränkt unterstützt. Deswegen ist es im betrachteten Fall notwendig, die *ProC/B*-Schnittstelle bei der Modellierung von Zuverlässigkeitsaspekten zu umgehen und anstelle dessen Petri-Netz Modelle zu erstellen. Trotz dieses Defizits können andere Zuverlässigkeitsaspekte, wie zum Beispiel ausfallbehaftete

aktive Ressourcen oder Timeout-Mechanismen für unsichere Dienste auch ohne HIT-Code in *ProC/B* abgebildet werden, vgl. [56].
Die vorgestellte Modellbildung ist somit hybrid und es liegt ein Petri-Netz Z-Modell (s.o.) und ein *ProC/B* P-Modell (hier nicht betrachtet, Details in [12]) vor. Der hybride Ansatz trägt dem Umstand Rechnung, dass es nicht eine universal anwendbare und adäquate Modellierungs-Notation gibt, sondern Notationen ggf. im Hinblick auf gewisse Defizite in ihrer Ausdrucksmächtigkeit oder im Hinblick auf durch die Analyse auferlegte Restriktionen sinnvoll kombiniert werden können und müssen.

Analyseergebnisse Das Untersuchungsziel ist die Bestimmung der Auslastung der Mechaniker und die Verfügbarkeit der Gabelstapler in Abhängigkeit von der Wartungsrate. Die *ASYNC*-Lösungszeit der Markov-Kette mit 12 246 960 Zuständen ist sehr sensitiv bzgl. der eingestellten Wartungsrate. Die Lösungszeit mit 2 Worker-Prozessen liegt zwischen 810 Sekunden (Wartungsrate $20\,000^{-1}$ Minuten) und 17 067 Sekunden (100^{-1} Minuten).
Die Abb. 8.10 zeigt die Auslastung der Mechniker. Die Auslastung wurde für Wartungsraten von $20\,000^{-1}$, $10\,000^{-1}$, $8\,000^{-1}$, $6\,000^{-1}$, $4\,000^{-1}$, $2\,000^{-1}$, $1\,000^{-1}$, 500^{-1}, 250^{-1} und 100^{-1} Sekunden^{-1} ermittelt und für dazwischenliegende Werte linear approximiert. Erwartungsgemäß fällt die Auslastung des Reparatur-Mechanikers mit Zunahme der Wartungsrate, ebenso steigt die Auslastung des Wartungs-Mechanikers mit Zunahme der Wartungsrate. Für sehr kleine Wartungsraten geht die Grenzauslastung des Reparatur-Mechanikers gegen 0.57. Die mittlere Auslastung beider Mechaniker wird für eine Wartungsrate von etwa $2\,000^{-1}$ Sekunden^{-1} minimal, vgl. mittlere Kurve in Abb. 8.10.

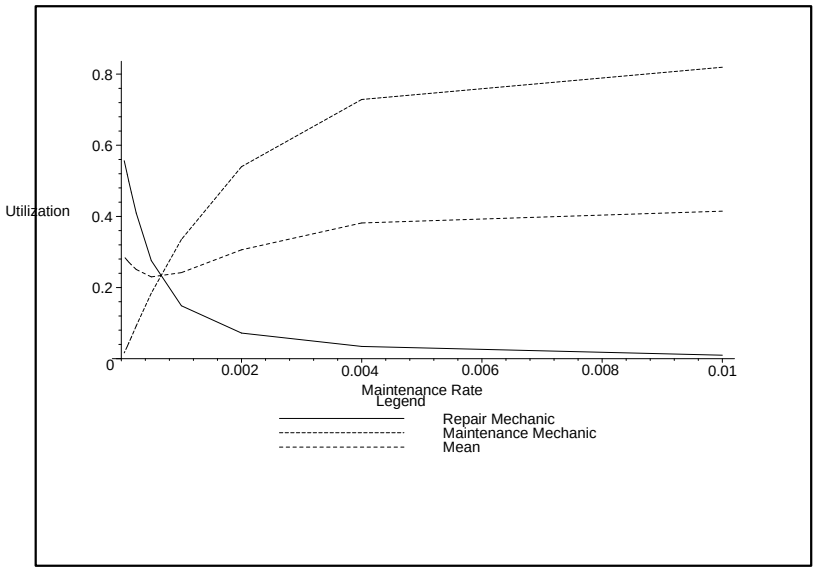

Abbildung 8.10: Verfügbarkeits-Verteilung Gabelstapler in Abhängigkeit von Wartungs-Rate

Die Abb. 8.11 zeigt exemplarisch die Verteilung der verfügbaren Gabelstapler im Lager 1 in Abhängigkeit von der Wartungsrate (0..10 Gabelstapler sind verfügbar, vgl. Abb. 8.7). Das Montonieverhalten der Verteilungen variiert deutlich mit der Wartungsrate. Für die Werte $10\,000^{-1}$, $8\,000^{-1}$ und $6\,000^{-1}$ ist wird das Maximum der Verteilung für weniger als 10 Gabelstapler er-

100	250	500	1 000	2 000	4 000	6 000	8 000	10 000	20 000
2.3402	6.478	8.2274	8.8179	8.8249	8.4096	8.0818	7.8588	7.7013	7.3234

Tabelle 8.11: Kehrwert Wartungsrate (oben) und mittlere Anzahl verfügbarer Gabelstapler (unten) im Lager 1

reicht, für die Werte $4\,000^{-1}$, $2\,000^{-1}$ $1\,000^{-1}$ und 500^{-1} ist die Verteilung im betrachteten Ausschnitt streng monoton steigend und erreicht ihr Maximum bei 10 Gabelstaplern. Bei einer Wartungsrate von etwa 250^{-1} kippt das Monotonierverhalten recht schnell und die Verteilung für eine Wartungsrate von 100^{-1} ist streng monoton fallend. Bei einer Wartungsrate von 100^{-1} tritt das "Worst-Case"-Szenario (kein Gabelstapler verfügbar) sogar mit der höchsten Wahrscheinlichkeit ein.

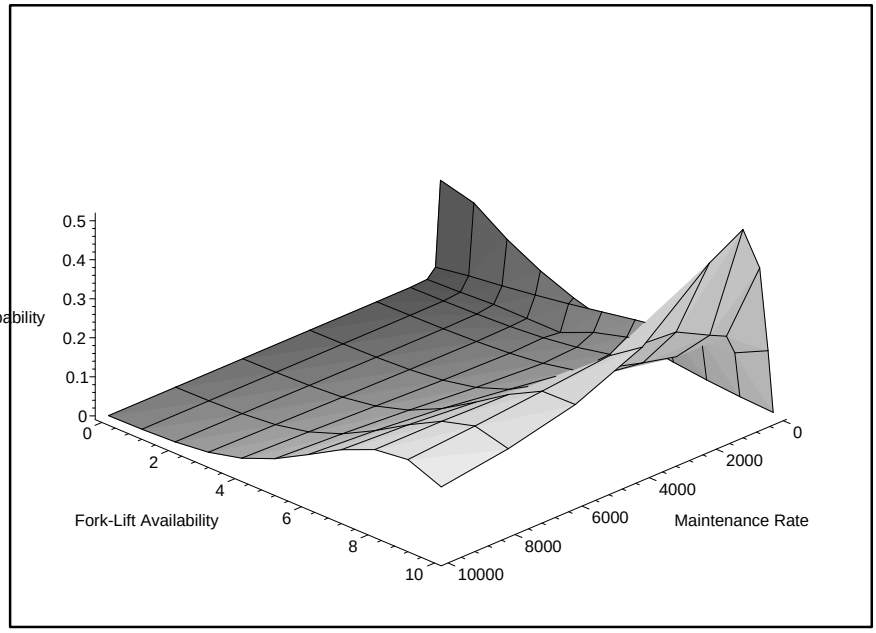

Abbildung 8.11: Verfügbarkeits-Verteilung Gabelstapler in Abhängigkeit von Wartungs-Rate

In Tab. 8.11 ist die mittlere Anzahl verfügbarer Gabelstapler im Lager 1 angegeben. Obwohl die Wahrscheinlichkeit dafür, dass alle Gabelstapler verfügbar sind, bei einer Wartungsrate von $1\,000^{-1}$ am höchsten ist (rechter Peak in Abb. 8.11), ist die mittlere Verfügbarkeit dort nicht maximal. Der Mittelwert wird bei einer Wartungsrate von etwa 2000^{-1} maximal.
Zusammen mit den Verfügbarkeits-Verteilungen der Gabelstapler an den anderen Lagern (hier nicht dargestellt) erhält man die Verteilung über allen Ressourcen-Konfigurationen $\mathcal{C}$ (p_c-Werte, s.o.). Diese Verteilung ist hinsichtlich der Durchlaufzeiten involvierter Ein- und Auslagerungsprozesse (erfasst im P-Modell) für eine Wartungsrate von c.a. $5\,000^{-1}$ optimal, Details siehe [12]. Die Abweichung ist ein Indiz dafür, dass die Gabelstapler im Lager 1 keinen Engpass darstellen, weil die optimale Performance der Prozesse bei einer - aus der Sicht der Gabelstapler im Lager 1 - nicht optimlen Einstellung der Wartungsrate erzielt wird.
Die Analyse des P-Modells im Zusammenspiel mit den obigen Ergebnissen der Z-Modell-Analyse

wird an dieser Stelle nicht weiter betrachtet, weil das P-Modell einer numerischen Analyse nicht zugänglich ist. Die Ergebnisse einer simulativen Auswertung des P-Modells basierend auf den Ergebnissen der Z-Modell-Analyse sind in [12] dokumentiert.

8.3 Diskussion

Es ist gelungen, die Methodik der Markov-Ketten-Analyse benutzerfreundlich in die *ProC/B*-Modellierungsumgebung zu integrieren. Nachfolgend werden die Erfahrungen und Einsichten, die bei der Modellierung und Analyse logistischer Systeme mit Markov-Ketten gewonnen wurden, zusammenfasst.

Bedeutung von Petri-Netzen *PN*s und diverse Erweiterungen, die auf eine tokenbasierte Modellierung gängiger Ablaufkonzepte abzielen (u.a. Aktivitätsdiagramme in UML 2), sind in der Modellbildung prozess-orientierter technischer Systeme verbreitet [65] und haben sich auch in dieser Arbeit bewährt. Auch wenn eine Modellbildung logistischer Systeme mit *PN*s prinzipiell möglich ist [54, 57], hat die Erfahrung im Umgang mit Gestaltern logistischer Systeme gezeigt, dass eine anwendungsspezifische Notation wie *ProC/B* zu einer erheblich höheren Akzeptanz einer modellbasierten Systemanalyse führt und Systemgestalter befähigt, Wissen und Informationen über logistische Systeme in den Prozess der Modellbildung einzubringen. Eine direkte Gegenüberstellung verhaltensäquivalenter *ProC/B*- und *PN*-Modelle verdeutlicht, dass eine *ProC/B*-Notation prozess-orientierter Abläufe erheblich kompakter und suggestiver ist, vgl. Abb. 8.4 (*ProC/B*-Modell von Lagerprozessen) und Abb. 8.5 (*PN*-Modell).
Die Methodik, zuerst *ProC/B*-Modelle händisch zu erstellen und dann daraus *PN*s automatisiert zu generieren, muss abgewandelt werden, wenn die *ProC/B*-Notation bzgl. der Beschreibung gewisser Systemeigenschaften defizitär ist. Im Abschnitt 8.2.3 wurden bestimmte Zuverlässigkeitsaspekte identifiziert, deren Abbildung in *ProC/B* problematisch ist. Hier ist eine direkte (manuelle) *PN*-Modellierung besser. Ansonsten erfüllen *PN*s als interne Zwischen-Notation eine bedeutsame Vermittlungsfunktion zwischen der *ProC/B*-Welt und der Markov-Ketten-Welt.

Endlicher Zustandsraum In dieser Arbeit werden nur Markov-Ketten mit endlichem Zustandsraum betrachtet. Deswegen muss die Kapazität eingesetzter Ressourcen endlich, die Anzahl gleichzeitig existierender Prozess-Objekte beschränkt und die Anzahl in Prozesse involvierter Aktivitäten ebenfalls endlich sein. Es hat sich gezeigt, dass logistische Systeme häufig mit offener Systemlast modelliert werden, in denen die Anzahl instanziierter und gleichzeitig existierender Prozess-Objekte nicht beschränkt ist. Ein Grund hierfür ist, dass obere Schranken nicht bekannt sind bzw. einer Beschränkung im Modell kein Einfluss auf die interessierenden Analyseergebnisse beigemessen wird. Andererseits ist in logistischen Systemen die Population der Prozess-Objekte endlich, insbesondere in Situationen, in denen die Systemlast durch den Systemzustand kontrolliert wird (Abweisung bzw. Umleitung von Prozessen, Kanban-Steuerung) oder in denen Aktivitäten unter Zuhilfenahme endlicher Ressourcen ausgeführt werden. Insofern ist eine Modellierung mit endlicher Population - wie sie hier vorausgesetzt wird - in vielen Fällen realitätsnäher.

Stochastisches Modell vs. Determinimus Logistische Systeme weisen oft gemischt stochastisches und deterministisches Verhalten auf. Beispiele sind Lastgeneratoren, die durch (ideali-

siert deterministische) Fahrpläne getriggert sind, Reparaturzeiten und Lieferzeiten, die vertraglich zugesichert sind und getaktetes Verhalten in Fertigungslinien. Die Abbildung des Determinismus in einem rein stochastischen Modell - wie es Markov-Ketten sind - kann nur approximativ erfolgen. Hierdurch wird die Anwendbarkeit des Markov-Ketten-Instrumentariums sehr eingeschränkt. Für stochastische Modelle spricht der Umstand, das Determinimus im System oft eine idealisierte Annahme darstellt und die Annahme stochastischen Verhaltens eigentlich adäquater wäre (z.B. Fahrpläne).

Stochastik im zeitlichen Verhalten In Markov-Ketten sind Zeitverbräuche durch Exponentialverteilungen spezifiziert. Die Erfahrung im Umgang mit logistischen Systemen zeigt, dass die Exponentialverteilung oft "zu stochastisch" ist und anstelle dessen Verteilungen mit geringerem Variationskoeffizienten adäquater sind. Bekanntlich können weniger stochastische Zeitverbräuche durch Phasenverteilungen approximiert werden, die aus Exponentialverteilungen konstruierbar sind. Allerdings führt dies zu zusätzlichen Zuständen. Beliebige Verteilungen führen zur Klasse (Generalisierter) Semi-Markov Prozesse, die prinzipiell auch analysierbar sind, allerdings wird die Erweiterung der Modellklasse durch einen erheblichen Mehraufwand bei der analytischen Betrachtung erkauft, weil der unterliegende stochastische Prozess partiell nicht gedächtnislos ist.

Größe Zustandsraum Obwohl *ASYNC* leistungsfähige Rechen- und Platzressourcen erschließt, überschreitet der Zustandsraum logistischer Systeme oft die Grenzen, innerhalb derer eine numerische Analyse des Zustandsraums praktizierbar ist. Deswegen liegt das Haupteinsatzgebiet von *ASYNC* in der Voranalyse dedizierter Teilmodelle, zum Beispiel mit dem Ziel, vereinfachte Beschreibungen für diese abzuleiten, vgl. Abschnitt 8.2.2 zur Aggregat-Berechnung.
In logistischen Systemen existieren zahlreiche "Treiber" großer Zustandsräume. Ein Treiber ist die Diversifikation von Prozess-Objekt-Ausprägungen. Wenn neben der essentiellen Kodierung von Ressourcenzuständen und Aufenthaltsorten der Prozess-Objekte auch (als Faktor eingehende) Merkmale einzelner Prozess-Objekte im Zustandsraum kodiert werden müssen, ist die Größe des Zustandsraums oft nicht behandelbar.
Eine ähnliche Zustandsraumexplosion tritt ein, wenn das Verhalten der Prozess-Objekte nicht gedächtnislos ist, z.B. wenn Prozess-Objekte repetitiv Teilprozesse durchlaufen (Prozess-Objekt merkt sich Anzahl der Schleifendurchläufe) oder bei der Aufspaltung und Rekombination von Prozessen (erzeugte Teilprozesse merken sich Zusammengehörigkeit). In diesen Fällen muss zustands- bzw. datenabhängiges Verhalten durch Randomisierung approximiert werden (probabilistische Anzahl von Schleifendurchläufen im *ProC/B*-Modell in Abb. 8.4).
Ein weiterer Treiber für große Zustandsräume sind strukturierte Systeme, in denen die Zustandsräume der Teilsysteme faktoriell zur Größe des gesamten Zustandsraums beitragen. Dies trifft zum Beispiel für Systeme mit ausfallbehafteten Resourcen zu, vgl. Abschnitt 8.2.3.

Performance-Modellierung versus Performance-Messung Die Performance-Modellierung spielt im (Geschäfts)-Prozess-Management, wie es gegenwärtig in der betrieblichen Praxis anzutreffen ist, nur eine untergeordnete Rolle. Der Versuch, Prozesse kontinuierlich zu verbessern und veränderten Rahmenbedingungen anzupassen, basiert auf der Erfassung bzw. Messung der Performance-Daten am realen System sowie einer geeigneten Aufbereitung entscheidungsrelevanter Mess-Daten in Data Warehouses oder Process Warehouses. In der Terminologie

unterstützender Software werden hierfür Business-Warehouse (BW) Lösungen und Business-Intelligence (BI) Lösungen verwendet. Ein BW ist eine Datenbank (Daten-Modellierung), in der beobachtete Abläufe (sogenannte Bewegungsdaten) und Performance-relevante Zusatzinformationen (was passiert wann und wo) gespeichert werden. BI ist grob gesagt eine Umschreibung aller Aktivitäten, die eine Analyse und Aufbereitung der Informationen aus dem BW für menschliche Entscheider umfasst. Dies beinhaltet sogenannte Data-Mining- und Process-Mining-Techniken, mit denen eine Prozess-Sicht und prozessorientierte Performance-Kennzahlen aus dem BW rekonstruiert bzw. extrahiert wird.

Eine Messung interessierender Performance-Kennzahlen wird also durch den Umstand begünstigt, dass Prozesse in logistischen Systemen zunehmend IT-gestützt ablaufen, so dass die Erfassung bzw. das Abgreifen von Daten kostengünstig und automatisiert möglich ist.

Eine Performance-Modellierung wird durch die Vorgabe erschwert, dass hauptsächlich die Performance kundennaher Prozesse (B2C und B2B) interessiert und für das Management entscheidungsrelevant ist. Die Performance derartiger "Top-Level" Prozesse hängt nicht nur von der Organisation der Geschäftsprozesse selbst, sondern insbesondere auch von der Performance unterliegender IT-Integrations-Ebenen (Web-services etc.) und von der Performance der IT-Systeme ab. Die Berücksichtigung aller Aspekte (IT, IT-Integration, Geschäftsprozesse) in einem Performance-Modell führt zwangsläufig zu sehr großen, gegenwärtig nicht behandelbaren Modellen.

Eine Performance-Modellierung ist interessant, wenn die Performance nicht messbar ist (Planung neuer Systeme) oder ein "Trial-and-Error" Ansatz basierend auf BW und BI risikobehaftet ist, weil die Ursache-Wirkungs-Beziehungen unklar sind.

Kapitel 9

Fazit und Ausblick

Durch die Kombination einer asynchronen und hierarchischen Block-Jacobi-Iteration mit einer Kronecker-Darstellung der Generatormatrix und der Nutzung einer parallelen Rechnerarchitektur mit verteiltem Speicher (Rechner-Cluster/Netzwerk) können Markov-Ketten-Modelle mit nahezu 1 Milliarde Zuständen gelöst werden. Diese Leistungsfähigkeit wird durch Synergieeffekte dieser speziellen Konstellation ermöglicht:

1. Die Kronecker-Darstellung der Generatormatrix profitiert von einer parallelen Rechnerarchitektur, weil verteuerte Rechenoperationen auf der Generatormatrix (bedingt durch das generische Datenformat) bei einer verteilten Ausführung der Iterationsschritte zusätzliche Rechenressourcen erschließen. Zudem wird in einem Rechnernetzwerk ein großer, wenn auch verteilter Arbeitsspeicher verfügbar gemacht. Eine Verteilung des Iterationsvektors ist vorteilhaft, weil der Platz-Flaschenhals (= Speicherung des Iterationsvektors) behoben wird.

2. Die Kronecker-Darstellung liefert eine Blockstruktur der Generatormatrix, an der blockorientierte numerische Lösungsverfahren adaptieren können. Des Weiteren ist die Blockstruktur eine geeignete Dekomposition des Berechnungsproblems, auf dem ein verteiltes Lösungsverfahren aufsetzen kann.

3. Eine verteilte Block-Jacobi-Iteration profitiert von der Kronecker-Darstellung, weil in jedem Arbeitsspeicher die vollständige Generatormatrix als Duplikat gehalten werden kann, wodurch Datenabhängigkeiten verringert und gleichzeitig die Kommunikation flexibilisiert wird.

4. Verteilte, asynchrone Iterationen sind unabhängig von der Verfügbarkeit "aktueller" kommunizierter Daten und deswegen geeignet für Kommunikationsmedien mit geringer bzw. lastabhängig-schwankender Bandbreite. Umgekehrt profitiert das Kommunikationsmedium von verteilten, asynchronen Iterationen, die zu einer gleichmäßigen Belastung des Mediums führen, weil Rechenphasen (geringe Belastung) und Kommunikationsphasen (hohe Belastung) nicht alternierend, sondern zeitlich überlappend ablaufen.

5. Asynchrone und hierarchische Iterationen sind flexibel ausführbar und können an Performance-Charakteristika des Kommunikationsmediums angepasst werden.

Weitere Ergebnisse der vorliegenden Arbeit nach Themen sortiert lauten:

Engpass-Analyse Experimente mit *ASYNC* auf Kommunikationsressourcen, dessen theoretische Bandbreite auf 10 MBit/s limitiert ist, haben gezeigt, dass bereits für kleine Markov-Ketten-Modelle mit wenigen Millionen Zuständen eine vollständig entkoppelte Kommunikation Engpässe und Instabilitäten in der Berechnung erzeugt. Diese können vermieden werden, wenn als Datenkonsumenten agierende Rechner bei Zugriff aus kommunizierte Daten ihren Aktualisierungsbedarf beim Produzenten anmelden. In diesem Fall ist die Rate, mit der Daten ausgetauscht werden, durch den Konsumenten partiell gesteuert, vgl. Def. 5.10, S. 57. Messungen auf der Ebene eines Kommunikationsnetzwerks haben gezeigt, dass bei Markov-Ketten-Modellen mit mehr als 10 Millionen und einer Verteilung auf 4 Rechner der Kommunikationsaufwand bereits so hoch ist, dass eine 10 MBit/s-Verbindung trotz konsumenten-gesteuerter Kommunikation nicht ausreichend Kapazität bietet, vgl. Abschnitt 6.1.1. Erfreulicherweise sind verfügbare 1 GBit/s-Verbindungen (Switch im Rechner-Cluster) für alle behandelbaren Problemgrößen ausreichend. Das größte gelöste Markov-Ketten-Modell hat knapp 897 Millionen Zustände. Die *ASYNC*-Lösung mit 14 *Worker*-Prozessen auf 7 Dualprozessoren benötigt etwas über 55 Stunden, vgl. Tab. 6.5. Während der *ASYNC*-Lösung werden 1.2 Millionen Vektoren mit einem Datenvolumen von insgesamt 8.5 Tera Bytes transferiert, vgl. Tab. 6.6. Es sind wenigstens 14 *Worker*-Prozesse notwendig, um den Platzaufwand je Prozess auf unter 3 GB zu drücken (durch Betriebssystem vorgegebene Platzschranke).
Das zweitgrößte gelöste Modell hat 424 Millionen Zustände (`SUH1`-Modell mit k=6, vgl. Tab. 3.2) und verursacht einen Platzaufwand je Prozess von 6.34 GB in einer 1-Prozess-Konfiguration und 2.30 GB in einer 4-Prozess-Konfiguration, vgl. Tab. 6.7. Dieses Modell ist für eine 1-Prozess-Konfiguration zu gross (s.o.). Das `SUH1`-Modell mit k=7 hat bereits 1.3 Milliarden Zustände und erzeugt einen Platzengpass. Eine Erhöhung der Anzahl verfügbarer Rechner von aktuell 6 auf 18 (mindestens) würde den Platzengpass beheben. Möglicherweise verlagert sich dann der Engpass wieder auf das Kommunikationsmedium, weil je Iterationsschritt ca. 28 GB Daten transferiert werden müssen.

Auswertung des dynamischen Laufzeitverhaltens *ASYNC* bietet eine Experimentierumgebung, in der unterschiedliche Varianten asynchroner Iterationen zu Testzwecken auswählbar sind, vgl. Tab. 5.4. In Trace-Dateien können zur Laufzeit Ereignisse und Zustände protokolliert werden. Die statistische und visuelle Auswertung dieser Trace-Dateien ist sehr hilfreich und gibt Einblick in den dynamischen Ablauf von Berechnung und Kommunikation, vgl. Abb. 6.3. Experimente mit dem `FMS`-Modell haben gezeigt, dass schwach asynchrone Iterationen die Performance total asynchroner Iterationen erreichen können, wenn die Last gut balanziert ist und wenig Datenabhängigkeiten (verursachen potentiell Synchronisation) bestehen, vgl. Tab. 6.1. Die Visualisierung von Trace-Dateien hat gezeigt, dass sich nach einigen Iterationsschritten stationäre Rechen- und Kommunikationsmuster einstellen, die in wiederholten Ausführungen nicht reproduzierbar sind.

Beschleunigung und Effizienz *ASYNC* erreicht durch die Verteilung der Berechnung eine aus praktischer Sicht akzeptable Effizienz zwischen 0.5 bis 0.75. Beispielsweise erreicht *ASYNC* für das `SUH1`-Modell im Vergleich zu einer *ASYNC*-1-Prozess-Lösung eine Beschleunigung von 1.47 (2 Prozesse), 2.61 (4 Prozesse) und 3.52 (6 Prozesse), vgl. Tab. 6.3. *ASYNC* kann schon

allein deswegen keine (optimalen) lineare Beschleunigung erzielen, weil die Asynchronität in der verteilten Berechnung einen signifikanten numerischen Mehraufwand gemessen an Iterationsschritten bewirkt. Beispielsweise erhöht sich der numerische Aufwand beim Übergang von einer 1-Prozess-Lösung auf eine 2-Prozess-Lösung um ca. 36%. Zudem ist jede gemessene Beschleunigung als stochastische Variable zu interpretieren, weil das Rechen- und Kommunikationsmuster stochastisch ist (hat Einfluss auf die Effektivität asynchroner Iterationen). In der 6-Prozess-Lösung werden Beschleunigungen zwischen 3.39 (schlechteste) und 3.63 (beste) bei unveränderter Experimentumgebung gemessen.

Asynchronität und Lösungszeit In den Experimenten erbringen total asynchrone Iterationen keine signifikante Performance-Veränderung im Vergleich zu schwach asynchronen Iterationen. Diese Aussage basiert auf Experimenten, in denen das `FMS`-Modell mit verschiedenen Implementierungsvarianten schwach asynchroner Iteration gelöst wird (vgl. Tab. 6.1) und in denen das `SUH2`-Modell mit partiell asynchronen Iterationen gelöst wird (vgl. Tab. 6.2).

Implementierung Für *ASYNC* ist eine Kommunikationsschnittstelle entwickelt worden, die Funktionalitäten für verteilt und asynchron auszuführende Iterationsschritte bietet. Dies beinhaltet Sende- und Empfangsbefehle mit asynchroner Semantik neben anderen, vgl. Abschnitt 5.6.1. Die für eine asynchrone Berechnung notwendige Unabhängigkeit von der Verfügbarkeit kommunizierter Daten wird durch eine Pufferung gewährleistet. Hierfür sind unterschiedliche technische Puffer-Realisationen verfügbar, vgl. Abschnitt 5.6.3. Die Kommunikationsschnittstelle abstrahiert von allen technischen Details, insbesondere auch davon, ob die Funktionalität auf *PVM* oder wahlweise auf *MPI* aufsetzt, vgl. Abschnitt 5.6.2. So ist eine maximale Flexibilität und Portabilität garantiert.

Modellbasierte Performance-Bewertung synchroner und asynchroner Iterationen Stochastische Modelle zur zeitlichen Bewertung asynchroner Iterationen im Zusammenspiel mit theoretischen Abschätzungen der Konvergenzrate sind problematisch. Eine analytische Behandelbarkeit stochastischer Performance-Modelle und gewisse Anforderungen mathematischer Modell erzwingen, asynchrone Iterationen mit vielen Restriktionen zu betrachten. Bereits schwach-asynchrone Iterationen führen zu komplexen stochastischen Modellen, vgl. Abschnitt 6.2.1.

Anwendbarkeit und Nützlichkeit Performance-Kennzahlen logistischer Systeme können basierend auf Markov-Ketten und dem numerischen Lösungsverfahren aus *ASYNC* ermittelt werden. *ASYNC* ist benutzerfreundlich in die *APNN-Toolbox* integriert und kann von der graphischen Benutzeroberfläche aus aufgerufen werden, vgl. Abschnitt 7.4. Die entwickelte Anbindung des Markov-Ketten-Instrumentariums an das *ProC/B*-Modellierungs-Interface via automatisierter Modellübersetzer ermöglicht logistische Systeme einfach mit *ASYNC* zu analysieren, vgl. Abschnitt 7.2. Die Anwendbarkeit und Nützlichkeit des *ASYNC*-Instrumentariums wird durch 3 Anwendungsbeispiele demonstriert. Dies beinhaltet die Bewertung von Performance-Kennzahlen (Durchsatz bedienter LKWs, Auslastung involvierter Ressourcen) und die Risikoabschätzung für kritische Systemzustände in einer Stückgutumschlaghalle, die Berechnung eines Aggregats für Lager-Prozesse in einer Lieferkette und eine Zuverlässigkeitsanalyse von Prozessen mit ausfallbehafteten Ressourcen, vgl. Abschnitt 8.2.

Steuerung Die Asynchronität und Hierarchisierung der *ASYNC*-Iteration bieten zahlreiche Parameter, die das stochastische Laufzeitverhalten und die Konvergenz der numerischen Berechnung beeinflussen und als Steuerparameter zur Erzielung einer verbesserten Konvergenz und Laufzeit dienen können. Allerdings hat es sich als schwierig herausgestellt, den Einfluss zu bewerten. Die Aussagekraft beobachtbarer Wirkzusammenhänge aus einzelnen Experimenten ist gering, weil sie - bedingt durch die inhärente Stochastik des Laufzeitverhaltens - nicht reproduzierbar und somit nicht allgemeingültig sind. Auch der Test von einfachen "Reinforcement-Learning"-Strategien, um beispielsweise ein optimales Aufwand-Nutzen Verhältnis für innere Iterationen in der hierarchischen Iteration zu "lernen", hat keinen signifikanten Vorteil erbracht. Lediglich eine Steuerung in der *ASYNC*-Kommunikation konnte erfolgreich zur Vermeidung von Engpässen im Kommunikationsmedium angewendet werden.

Ausblick Aus praktischer Perspektive wäre der Test noch größerer Rechner-Cluster oder anderer paralleler Rechnerarchitekturen interessant. Eine Portierung von *ASYNC* auf andere, noch leistungsfähigere Architekturen sollte einfach möglich sein und die Lösung noch größerer Markov-Ketten-Modelle erlauben. Auch wenn Rechenzeiten von mehreren Tagen einen Zeitengpass in der Lösung anzeigen, so lag der unmittelbare Engpass im zur Verfügung stehenden Arbeitsspeicher (3 GB je Prozess). Abhilfe können hier platzeffiziente (symbolische, u.U. approximative) Darstellungen für die Diagonaleinträge in der Generatormatrix (hier explizit gespeichert) oder des Iterationsvektors schaffen, die bekannt sind, in dieser Arbeit für die hierarchische und asynchrone *BJAC*-Iteration aber nicht implementiert wurden.
ASYNC realisiert die verteilte und asynchrone Ausführung einer hierarchischen *BJAC*-Iteration, in der die Intra-Block-Lösung eine (einfache) *SOR*-Iteration ist. Darüber hinaus gibt es zahlreiche weitere blockorientierte und hierarchische Fixpunkt-Iterationen, die bereits mit mehr "sophisticated" problemspezifischen Anpassungen kombiniert wurden (z.B. Auswahl des Verfahrens zur Intra-Block-Lösung). In diesem Punkt könnte *ASYNC* erweitert werden.
Es wäre interessant, Problem-Instanzen zu identifizieren, für die Asynchronität im Vergleich zu synchronen oder schwach asynchronen Ausführungen iterativer Verfahren einen Performance-Vorteil erzielt, der in den hier betrachteten Problem-Instanzen durch einen numerischen Mehraufwand (Anzahl notwendiger Iterationen) verhindert wird.

Literaturverzeichnis

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transaction on Computer Systems*, 2(1), 1984.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets*. Wiley, New York, USA, 1995.

[3] T. Altiok. *Performance Analysis of Manufacturing Systems*. Springer, 1997.

[4] D. Arnold, H. Isermann, A. Kuhn, and H. Tempelmeier. *Handbuch Logistik*. VDI Verlag, 2002.

[5] M. Arns, M. Fischer, and P. Kemper. *Anwendung nicht-simulativer Techniken zur Analyse eines dezentralen Güterverkehrszentrums*. Technischer Bericht 03017, SFB 559, ISSN 1612-1376, 2003.

[6] M. Arns, M. Fischer, P. Kemper, and C. Tepper. Supply chain modelling and its analytical evaluation. *Journal of the Operational Research Society*, 53:885–894, 2002.

[7] J. M. Bahi. Asynchronous iterative algorithms for nonexpansive linear systems. *J. of Parallel and Distributed Computing*, 60:92–112, 2000.

[8] H. Bartsch and P. Birkenbach. *Supply Chain Management with SAP 'Advanced Planning Optimization' package*. Galileo Press, 2001.

[9] G.M. Baudet. Asynchronous iterative methods. *J. ACM*, 25:226–244, 1978.

[10] F. Bause. Queueing petri nets - a formalism for the combined qualitative and quantitative analysis of systems. *5th Int. Workshop on Petri Nets and Performance Models*, pages 14–23, 1993.

[11] F. Bause, H. Beilner, M. Fischer, P. Kemper, and M. Völker. The ProC/B toolset for the modelling and analysis of process chains. In T. Field, P.G. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation, Modelling Techniques and Tools*, Springer LNCS 2324, pages 51–70, 2002.

[12] F. Bause, H. Buchholz, M. Fischer, and P. Kemper. Hybrid performability analysis of logistic networks. In *Parallel and Distributed Simulation*, IEEE Computer Society, pages 131–138, 2004.

[13] F. Bause and P. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg, 1996.

[14] F.B. Beidas and G.P. Papavassilopoulos. Convergence analysis of asynchronous linear iterations with stochastic delays. *Parallel Computing*, 19:281–302, 1993.

[15] H. Beilner, R. Jansen, E. Jehle, A. Kuhn, and M. ten Hompel (Vorstand). *SFB 559 'Modellierung großer Netze in der Logistik' der Deutschen Forschungsgemeinschaft*. im Internet: `http://www.sfb559.uni-dortmund.de/index.php`, seit 1998.

[16] H. Beilner, J. Mäter, and C. Wysocki. The hierarchical evaluation tool HIT. 7th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 1994.

[17] A. Bell. Disk-based and distributed generation and analysis of large stochastic models. GI/Dagstuhl Research Seminar on Validation of Stochastic Systems, 2002.

[18] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Science*. SIAM, 1994.

[19] D.P. Bertsekas and D. Tsitsiklis. *Parallel and Distributed Computing - Numerical Methods*. Prentice-Hall, 1989.

[20] G. Bolch, S. Greiner, H. Meer de, and K. Trivedi. *Queueing networks and Markov chains*. J. Wiley, 1998.

[21] R. Bru, V. Migallón, J. Penadés, and D.B. Szyld. Parallel, synchronous and asynchronous two-stage multisplitting methods. *Electronic Transaction on Numerical Analysis*, 3:24–38, 1995.

[22] P. Buchholz. An adaptive aggregation/disaggregation algorithm for hierarchical markovian models. *J. of Operational Research*, 116(3):545–564, 1999.

[23] P. Buchholz. Hierarchical structuring of superposed GSPNs. *IEEE Transactions on Software Engineering*, 25(2):166–181, 1999.

[24] P. Buchholz. Projections methods for the analysis of stochastic automata networks. *3rd Int. Workshop on the Numerical Solution of Markov Chains (NSMC'99), in [86]*, pages 149–168, 1999.

[25] P. Buchholz. Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics*, 31(4):375–404, 1999.

[26] P. Buchholz. Multilevel solutions for structured Markov chains. *SIAM J. Matrix Anal. Appl.*, 22(2):342–357, 2000.

[27] P. Buchholz. An adaptive decomposition approach for the analysis of stochastic Petri nets. *Performance Evaluation*, 56:23–52, 2004.

[28] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations and sparse matrices with applications to the solution of Markov chain models. *INFROMS Journal on Computing*, 12(3):203–222, 2000.

[29] P. Buchholz, M. Fischer, and P. Kemper. Distributed steady state analysis using Kronecker algebra. *3rd Int. Workshop on the Numerical Solution of Markov Chains (NSMC'99), in [86]*, pages 76–95, 1999.

[30] P. Buchholz, M. Fischer, P. Kemper, and C. Tepper. New features in the APNN-Toolbox. In P. Kemper (ed.), editor, *Tools at Int. Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, Technical Report 760, University of Dortmund, pages 62–68, 2001.

[31] P. Buchholz and P. Kemper. On generating a hierarchy for GSPN analysis. *ACM Performance Evaluation Review*, 26(2):5–14, 1998.

[32] P. Buchholz and P. Kemper. A toolbox for the analysis of discrete event dynamic systems. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification*, volume 1633 of *Springer LNCS*, pages 483–486, 1999.

[33] P. Buchholz and P. Kemper. Kronecker based matrix representations for large Markov models. In *Validation of Stochastic Systems*, volume 2925 of *Springer LNCS*, pages 256–295, 2004.

[34] S.L. Campbell and C.D. Meyer. *Generalized Inverse of Linear Transformation*. Dover (reprint), New York, USA, 1991.

[35] H. Casanova, M.G. Thomason, and J.D. Dongarra. Stochastic performance prediction for iterative algorithms in distributed environments. *J. of Parallel and Distributed Computing*, 58:68–91, 1999.

[36] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer, 1999.

[37] A. Chan, B. Gropp, and R. Lusk. Performance visualization of parallel programs. `http://www-unix.mcs.anl.gov/perfvis/`, 2000.

[38] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and its Application*, 2:199–222, 1969.

[39] G. Ciardo and K. S. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18(3):37–59, 1993.

[40] Y. Cotronis and J. Dongarra (Eds.). *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer LNCS 2131, 2001.

[41] Supply Chain Council. *SCOR: Supply-Chain Reference Model*. Version 4.0, 2000.

[42] P. J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, 1977.

[43] C. G. Cullen. *Matrices and Linear Transformations*. Dover Publications, N.Y., 1990.

[44] T. Dayar and W.J. Stewart. Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains. *SIAM J. Scientific Computing*, 21(5):1691–1705, 2000.

[45] N.J. Dingle and W.J. Knottenbelt. Distributed solution of large markov models using asynchronous iterations and graph partitioning. In *18th UK Performance Engineering Workshop (UKPEW 2002)*, pages 27–34, 2002.

[46] S. Donatelli. Superposed generalized stochastic petri nets: definition an efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets*, LNCS 815, pages 258–277. Springer, 1994.

[47] S. Donatelli and P. Kemper. Integrating synchronization with priority into a kronecker representation. *Performance Evaluation*, 44:73–96, 2001.

[48] J. Dongarra, S. Huss-Ledermann, S.W. Otto, M. Snir, and D.W. Walker. *MPI - The Complete Reference*. MIT Press, 1996.

[49] W.J. Stewart (Ed). *Numerical Solution of Markov Chains*. Marcel Dekker, 1990.

[50] W.J. Stewart (Ed). *Computations with Markov Chains*. Kluwer, 1995.

[51] A. Geist et. al. *PVM - Parallel Virtual Machine*. MIT Press, 1994.

[52] M. Fischer. *Numerische Verfahren zur quantitativen Analyse von ProC/B und Petri-Netz Modellen*. Technischer Bericht 03012, SFB 559, ISSN 1612-1376, 2003.

[53] M. Fischer and C. Dilling. *Analytisch-numerische Techniken zur Lagerbestandanalyse unter Berücksichtigung einer zeitlich variierenden Last*. Technischer Bericht 03013, SFB 559, ISSN 1612-1376, 2003.

[54] M. Fischer and P. Kemper. Modelling and analysis of a freight terminal with stochastic Petri nets. *Proc. of 9th IFAC Symposium Control in Transportation Systems 2000*, 2:295–300, 2000.

[55] M. Fischer and P. Kemper. Distributed numerical Markov chain analysis. *Euro PVM/MPI Conference, in [40]*, pages 272–279, 2001.

[56] M. Fischer and P. Kemper. Perspectives on performability evaluation in the ProC/B toolset. *Proc. 6th Int. Workshop on Performability Modeling of Computer and Communication Systems*, pages 35–38, 2003.

[57] M. Fischer, P. Kemper, and Ch. Möller. Markov-Ketten Analyse des Umschlag-Terminals eines Güterverkehrszentrums mit Petri-Netzen. *DHF-Journal*, 4:18–22, 2000.

[58] M. Fischer, P. Kemper, C. Tepper, and Z. Wu. Abbildung von ProC/B nach Petri netzen - version 2. Technischer Bericht 03011, SFB 559, ISSN 1612-1376, 2003.

[59] M. Fischer and C. Tepper. GSPNs to support aggregation in the ProC/B toolset. *In P. Kemper (ed): Workshop on Stochastic Petri nets and related formalisms, Technischer Bericht 780, Universität Dortmund, Fachbereich Informatik*, pages 26–46, 2003.

[60] M. Fischer and Z. Wu. ASYNC Software-Entwurf . `http://ls4-www.informatik.uni-dortmund.de/home/fischer/PhD/Design/index.html`, 2002.

[61] M. Fisz. *Wahrscheinlichkeitsrechnung und mathematische Statistik.* Verlag der Wissenschaften, Berlin, 1989.

[62] A. Frommer and D.B. Szyld. Asynchronous two-stage iterative methods. *Numerische Mathematik*, 69:141–153, 1994.

[63] A. Frommer and D.B. Szyld. Asynchronous iterations with flexible communication for linear systems. *J. Calculateurs Paralleles*, 210:421–429, 1998.

[64] A. Frommer and D.B. Szyld. On asynchronous iterations. *J. of Computational and Applied Mathematics*, 23:201–216, 2000.

[65] C. Girault and R. Valk. *Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications.* Springer, 2003.

[66] A. Graham. *Kronecker Products and Matrix Calculus with Applications.* J. Wiley and Sons, 1981.

[67] B.R. Haverkort, R. Marie, G. Rubino, and K. Trivedi. *Performability Modelling - Techniques and Tools.* J. Wiley and Sons, 2001.

[68] B. Hendrickson and R. Leland. *The Chaco user's Guide, Version 2.0.* Technical Report SAND94–2692, Sandia Nat. Lab., 1995.

[69] M. Jeckle, C. Rupp, J. Hahn, B. Zengler, and S. Queins. *UML 2 glasklar.* Hanser, 2004.

[70] P. Kemper. Numerical analysis based on superposed GSPNs. *IEEE Transactions of Software Engineering*, 22(9):615–628, 1996.

[71] W.J. Knottenbelt and P.G. Harrison. Distributed disk-based solution techniques for large Markov models. *3rd Int. Workshop on the Numerical Solution of Markov Chains (NSMC'99), in [86]*, pages 58–75, 1999.

[72] A. Kuhn. Prozessketten in der Logistik: Entwicklungstrends und Umsetzungsstrategien. Verlag Praxiswissen, 1995.

[73] P. J. Lanzkron, D.J. Rose, and D.B. Szyld. Convergence of nested classical iterative methods for linear systems. *Numerische Mathematik*, 58:685–702, 1991.

[74] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, 2000.

[75] C. Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets.* Wiley, 1998.

[76] B. Lubachevsky and D. Mitra. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *J. of the ACM*, 33(1):130–150, 1986.

[77] I. Marek and P. Mayer. Convergence analysis of an iterative aggregation/disaggregation method for computing the probability vector of stochastic matrices. *Numerical Linear Algebra with Applications*, 5(4):253–274, 1998.

[78] I. Marek and D.B. Szyld. Local convergence of the (exact and inexact) iterative aggregation method for linear systems and Markov operators. *Numerische Mathematik*, 69(1):61–82, 1994.

[79] I. Marek and D.B. Szyld. Comparison theorems for the convergence factor of iterative methods for singular matrices. *Linear Algebra and its Applications*, 316:67–87, 2000.

[80] I. Marek and D.B. Szyld. Comparison of convergence of general stationary iterative methods for singular matrices. *SIAM Journal on Matrix Analysis and Applications (to appear)*, 2002.

[81] J. Mäter and Studenten. Projektgruppe: Entwicklung eines Java-basierten Frameworks für verteilte Netzwerk-Messungen
. `http://ls4-www.informatik.uni-dortmund.de/QM/MA/jm/Toplehre/PG422/PG422.html`, 2003.

[82] V. Migallón, J. Penadés, and D.B. Szyld. Block two stage methods for singular systems and Markov chains. *Numerical Linear Algebra with Applications*, 3:413–426, 1996.

[83] V. Migallón, J. Penadés, and D.B. Szyld. Experimental study of parallel iterative solutions of Markov chains with block partitions. *3rd Int. Workshop on the Numerical Solution of Markov Chains (NSMC'99), in [86]*, pages 96–110, 1999.

[84] A.C. Moga and M. Dubois. Performance of asynchronous linear iterations with random delays. *Proc. of 10th Int. Parallel Processing Symposium (IPPS)*, pages 625–630, 1996.

[85] M. Neumann and R.J. Plemmons. Convergent nonegative matrices and iterative methods for consistent linear systems. *Numerische Mathematik*, 31:265–279, 1978.

[86] B. Plateau, W.J. Stewart, and M. Silva (Eds.). *Numerical Solution of Markov Chains*. Prensas University Press, 1999.

[87] D. Reed and Pablo Research Group. Pablo tool. `http://www-pablo.cs.uiuc.edu/`, 2003.

[88] Y. Saad. *Projection Methods for the Numerical Solution of Markov Chain Models*. in [49], 1990.

[89] A.W. Scheer. *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, 2001.

[90] P.J. Schweitzer. *A Survey of Aggregation-Disaggregation in Large Markov Chains*. in [49], 1990.

[91] W. J. Stewart. *Introduction to the numerical solution of Markov-chains*. Princeton, 1994.

[92] J. C. Strikwerda. A probabilistic analysis of asynchronous iteration. *Linear Algebra and its Application (to appear)*, 349:125–154, 2002.

[93] Y. Su and A. Bhaya. Convergence of pseudocontractions and applications to Two-stage and asynchronous multisplitting for singular M-matrices. *SIAM J. of Matrix Analysis and Applications*, 22(3):948–964, 2001.

[94] D.B. Szyld. The mystery of asynchronous iterations convergence when the spectral radius is one. *Research Report 98-102, Dept. of Mathematics, Temple Uni, Philadelphia*, 1998.

[95] D.B. Szyld and M. T. Jones. Two stage and multisplitting methods for the parallel solution of linear systems. *SIAM Journal on Matrix Analysis and Applications*, 13:671–679, 1992.

[96] A. Üresin and M. Dubois. Effects of asynchronism on the convergence rate of iterative algorithms. *Journal of parallel and distributed computing*, 34:66–81, 1996.

[97] J.S. Vetter and D.A. Reed. Real-time performance monitoring, adaptive control, and interactive steering of computational grids. *Int. Journal of High Performance Computing Applications*, pages 357–366, 2000.

[98] D. M. Young. *Iterative Solution of Large Linear Systems*. Dover (unabridged republication of Academic Press publication in 1971), 2003.