Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung

Von der Fakultät Informatik der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

Vorgelegt von
Rainer Pollak
aus Schwäbisch Gmünd

Hauptberichter:	Prof. Dr. Andreas Reuter
Mitberichter:	Priv. Doz. Dr.-Ing. habil. Fritz Schmidt
Tag der mündlichen Prüfung:	12.01.1999

Institut für Parallele und Verteilte Höchstleistungsrechner
der Universität Stuttgart
1999

Inhaltsverzeichnis

Zusammenfassung

Parallele und verteilte Systeme gewannen im Laufe der letzten Jahre, aufgrund der Entwicklung immer leistungsfähigerer Hardware und Kommunikationsmedien, zunehmend an Bedeutung. Um die enorme Leistungsfähigkeit dieser Systeme nutzen zu können, ist es erforderlich, die zur Verfügung stehenden Ressoucen des parallelen bzw. verteilten Systems möglichst gleichmäßig auszulasten.

Aufgabe eines dynamischen Lastbalancierers ist es deshalb, die zur Verarbeitung anstehenden Anwendungen möglichst gleichmäßig auf die zur Verfügung stehenden Ressourcen zu verteilen. Insbesondere sollte ein dynamischer Lastbalancierer seine Aufgabe weitgehend transparent für den Anwender erledigen.

Einen entscheidenden Einfluß auf die Leistungsfähigkeit eines dynamischen Lastbalancierers hat die zur Verfügung stehende entscheidungsrelevante Information. Dabei stehen einem dynamischen Lastbalancierer im wesentlichen zwei Informationsquellen zur Verfügung, das Betriebssystem und die Anwendungen, wobei existierende Lastbalancierungsansätze in der Regel nur auf die Informationsbasis des Betriebssystems zurückgreifen.

Die vorliegende Arbeit beschäftigt sich aus diesem Grund mit der Frage, inwieweit ein dynamischer Lastbalancierer in der Lage ist, Informationen aus unterschiedlichen Quellen mit unterschiedlichen Eigenschaften, effizient zu verarbeiten.

Dazu wird in einem ersten Schritt die für einen dynamischen Lastbalancierer verfügbare Information klassifiziert. Das Klassifikationsergebnis sind die vier sogenannten Informationsebenen.

Da die Untersuchung mit bereits existierenden parallelen Anwendungen durchgeführt werden soll, wird anschließend die hierarchische Lastbalancierungsumgebung PaLaBer (**Pa**ralleler **La**st**B**alancier**er**) vorgestellt, die im Hinblick auf diese Untersuchung entwickelt wurde. Die PaLaBer-Umgebung ermöglicht es, parallele Anwendungen ohne nennenswerte Modifikationen des Quelltextes mit Hilfe eines dynamischen Lastbalancierers zu unterstützen. Im Vergleich zu anderen Systemen weist das PaLaBer-System jedoch eine neue Schnittstelle zu den Anwendungen auf. Basierend auf dieser Schnittstelle können die Anwendungen den Lastblancierer mit Informationen über ihren gegenwärtigen und voraussichtlich zukünftigen Lastzustand versorgen.

Anhand von umfangreichen Meßreihen wird anschließend aufgezeigt, daß sich die Verwendung von Informationen der verschiedenen Ebenen im Zusammenhang mit der anwendungsspezifischen Lastbalancierung positiv auf die Effizienz der dynamischen Lastbalancierung auswirkt. Insbesondere wird gezeigt, wie ein allgemeiner, betriebssystemintegrierter Lastbalancierer mit Hilfe der neuen Informationsschnittstelle zu den Anwendungen in einfacher Weise auf ein anwendungsspezifisches Lastverhalten hin optimiert werden kann, ohne die dynamische Lastbalancierungsstrategie dafür modifizieren zu müssen.

Hinweise auf die automatische Informationsselektion durch einen dynamischen Lastbalancierer und somit auf die praktische Umsetzung der erzielten Ergebnisse schließen die Betrachtungen ab.

Die Suche nach Wahrheit ist einerseits schwierig, andererseits einfach - denn es ist klar, daß kein Mensch ganz ans Ziel gelangt oder es ganz verfehlt. Jeder von uns trägt zu unserem Wissen von der Natur ein wenig bei, und aus der Gesamtheit der Fakten entsteht eine gewisse Größe.

-- Aristoteles, 350 v.Chr.

Kapitel 1

Einführung in die Aufgabenstellung der Lastverwaltung

Im Laufe der letzten Jahrzehnte wurden eine Vielzahl von parallelen Rechnerarchitekturen, nicht zuletzt im Rahmen von Dissertationen, konzipiert und in jüngster Vergangenheit in Form von kommerziellen Systemen verfügbar gemacht. So vielfältig wie die konzipierten Rechnerarchitekturen sind die von diesen Architekturen unterstützten parallelen Verarbeitungsmodelle. Da jede dieser Rechnerarchitekturen zusammen mit ihrem unterstützten, parallelen Verarbeitungsmodell unterschiedliche Anforderungen an einen Lastverwalter stellen, ist das Ziel dieses Kapitels eine Einführung in die verschiedenen Ansätze zur Lastverwaltung zu geben. Dazu wird zuerst eine Klassifikation von parallelen Rechnerarchitekturen sowie verschiedene Ausprägungen der Parallelverarbeitung durchgeführt. Darauf aufbauend werden die Problemstellung der Lastverwaltung und die möglichen Lösungsansätze zur Lastverwaltung diskutiert.

1.1 Klassifikation paralleler Rechnerarchitekturen

Im Vergleich zur konventionellen von-Neumann-Architektur [Hwan93], bei der es aufgrund der Limitation auf eine Verarbeitungseinheit nur einen Befehls- und Datenstrom gibt, existieren bei parallelen Rechnerarchitekturen mehrere, mitunter sogar tausende von Verarbeitungseinheiten und somit mehrere/tausende Befehls- und/oder Datenströme. Aufgrund ihrer Skalierbarkeit sind parallele Rechnerarchitekturen [Hert93] somit zumindest theoretisch in der Lage, eine beliebig skalierbare Rechenkapazität zur Verfügung zu stellen Sie bieten dadurch eine geeignete Plattform zur Lösung der sogenannten *Grand Challenges* [Meue95] an.
Zur Klassifikation von parallelen Rechnerarchitekturen wurden in der Vergangenheit eine Vielzahl von Taxonomien vorgeschlagen [Unge89]. Zu den bekanntesten Klassifikationsschemata gehören das Erlanger Klassifikationsschema (ECS) und die Taxonomie von Flynn. Während es mit Hilfe des Tripels (K*K‘, D*D‘, W*W‘) der Erlanger Klassifikation (siehe Tabelle 1) nicht nur möglich ist, einen einzelnen Parallelrechner, sondern mit Hilfe mehrerer Zusatzoperatoren ganze Rechnernetze detailliert zu beschreiben, zeichnet sich die Klassifikation von Flynn (siehe Tabelle 2) durch ihre Anschaulichkeit und Einfachheit aus. Aus diesem Grund wird in der weiteren Arbeit die Klassifikation von Flynn verwendet.
Flynn teilt die Rechnerarchitekturen in Abhängigkeit von der Einfachheit bzw. Vielfachheit der

Ebene des Parallelismus	Nebenläufigkeit	Pipelining
Programm Prozeß	K Multiprozessor Multirechner(netz)	K‘ Makropipelineprozessor
Maschinenbefehle Gruppe von Maschinenbefehlen Datenstrukturen	D Feldrechner Assoziativer Feldrechner	D‘ Instruktionspipelining (Superskalar, VLIW)
Teilelemente von Maschinenbefehlen Datenwort	W Parallelwortrechner	W‘ Phasenpipelining (Pipelining des Maschinenbefehlszykluses, Arithmetisches Pipelining)

Tabelle 1: Erlanger Klassifikationssystem (ECS) [Wald95].

Befehls- und Datenströme in vier Klassen ein. Die SISD-Klasse entspricht der konventionellen von-Neumann-Architektur mit einem Befehls- und einem Datenstrom und beinhaltet somit sämtliche sequentiellen Rechnerarchitekturen.
Über die MISD-Klasse wird in der Literatur kontrovers diskutiert. Während einige Autoren [Heis94][Unge89] diese Klasse als leer betrachten, ordnen andere Autoren [Hwan93] dieser Klasse den Architekturtyp der *Systolischen Arrays* zu. Auf die MISD-Klasse wird im nachfolgenden nicht weiter eingegangen, da für die Parallelverarbeitung die SIMD- und MIMD-Klasse von entscheidender Bedeutung sind.
In die Klasse der SIMD-Architekturen, bei denen ein Befehlsstrom auf mehrere Datenströme angewandt wird, fallen die Vektorrechner sowie die Feldrechner. Diese Architekturen haben sich in einigen Anwendungsbereichen (Bildverarbeitung, Neuronale Netze, Strömungssimulationen, etc.) als besonders geeignet erwiesen. Aufgrund ihres Operationsprinzips, einen Befehlsstrom auf eine Vielzahl von Datenströmen anzuwenden, sind sie jedoch nicht als allgemeine Arbeitsplattform geeignet.
Die allgemeinste der vier Klassen ist die MIMD-Klasse, bei der mehrere Befehlsströme auf mehrere Datenströme angewandt werden. In diese Klasse fällt der gesamte Bereich der Rechnernetze, der Multiprozessor- und Multicomputersysteme. Auf diese Klasse wird im weiteren detailliert eingegangen.
Das Klassifikationsschema von Flynn zeichnet sich, wie gesagt, durch seine Anschaulichkeit und Einfachheit aus. Diese Einfachheit bedingt jedoch, daß die Unterteilung der Rechnerarchitekturen in nur vier Klassen eine sehr grobe Aufteilung darstellt. Aus diesem Grund wird in Abbildung 1 ein Klassifikationssubschema für die MIMD-Klasse präsentiert. Die vier Klassifikationsmerkmale stehen dabei orthogonal zueinander. Das erste Klassifikationsmerkmal unterteilt die MIMD-Klasse in

- speichergekoppelte MIMD-Systeme und in
- MIMD-Systeme mit verteiltem Speicher.

In diesem Zusammenhang muß darauf hingewiesen werden, daß die Anordnung des Hauptspeichers weitgehend unabhängig von der logischen Sicht des Adreßraumes ist. So kann einer parallelen Anwendung in einem MIMD-System mit verteiltem Speicher ein logisch globaler, gemeinsamer Adreßraum zur Verfügung gestellt werden.

	SD (Single Data)	MD (Multiple Data)
SI (Single Instruction)	SISD Konventionelle von-Neumann-Architektur	SIMD Vektorrechner, Feldrechner
MI (Multiple Instruction)	MISD Systolische Arrays	MIMD Verteilte Systeme, Multicomputer, Multiprozessorsysteme

Tabelle 2: Klassifikationsschema nach Flynn.

Die MIMD-Systeme mit gemeinsamem Speicher können noch weiter in die Architekturklassen [Wald95] mit

- gleichförmigem Speicherzugriff (UMA - uniform memory access),
- ungleichförmigem Speicherzugriff (NUMA - non-uniform memory access) und
- nur Cachespeicherzugriff (COMA - cache-only memory architecture)

unterteilt werden. Bei der UMA-Architektur haben alle Prozessoren des Rechners identischen Zugriff auf die Speichermodule (vgl. Sequent Symmetry). Da die Realisierung eines gleichförmigen Speicherzugriffs für große MIMD-Systeme sehr aufwendig und damit teuer ist, wird bei diesen Systemen häufig das NUMA-Architekturprinzip angewandt (vgl. FX2800 von Alliant). Dabei wird entweder eine Speicherhierarchie verwendet, oder es wird jedem Prozessor ein lokaler Speicher zugeordnet und der globale Speicherzugriff über ein leistungsfähiges Verbindungsnetzwerk realisiert. Beim COMA-Architekturprinzip wird der dem jeweiligen Prozessor zugeordnete Speicher nur noch als Cachespeicher verwendet (vgl. SNI-KSR1, DASH). Dabei liegen alle Cachespeicher innerhalb des globalen Adreßraumes, und der Zugriff auf einen nicht lokalen Cache wird mit Hilfe eines sogenannten *cache-directory* realisiert, indem die Daten durch den Cache-Mechanismus zu dem Prozessor transferiert werden, der die Daten angefordert hat.
Basierend auf dem dritten Klassifikationsschema, der Kommunikationsstruktur, kommt man zu den beiden Architekturtypen der

- speichergekoppelten Systeme und der
- nachrichtengekoppelten Systeme.

Im Vergleich zu den speichergekoppelten Systemen, bei denen die Interprozeßkommunikation über gemeinsame Variablen erfolgen kann, muß bei den nachrichtengekoppelten Systemen die Interprozeßkommunikation über den expliziten Nachrichtenaustausch erfolgen.
Das letzte Klassifikationsmerkmal ist das Verbindungsnetzwerk des MIMD-Systems. Dieses Klassifikationsschema ist für die Lastverwaltung von besonderem Interesse, da bei nachrichtengekoppelten Systemen mit physikalisch verteiltem Speicher (Shared-Nothing Architekturen) die Lastverwaltung diese Information bei ihren Lastverteilungsentscheidungen miteinbeziehen kann. Im folgenden werden nur noch Shared-Nothing Architekturen betrachtet.

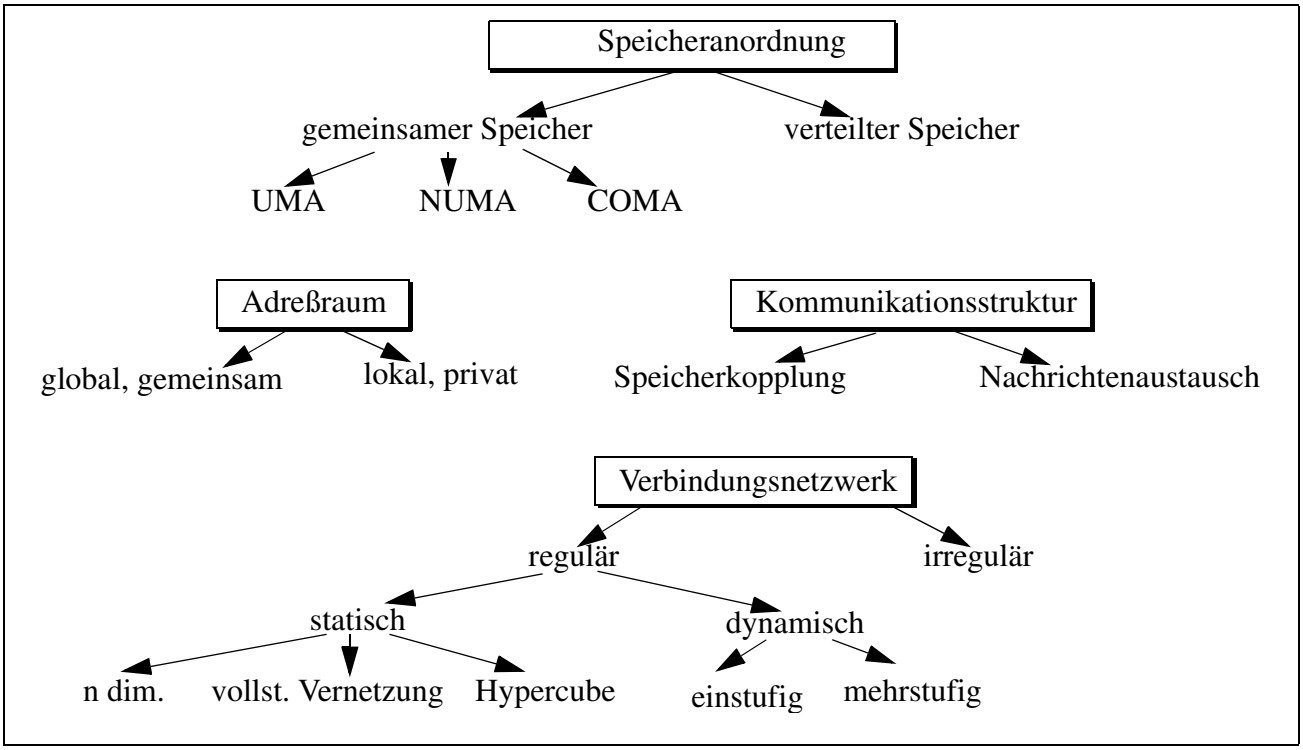


Abbildung 1: Klassifikationsmerkmale von MIMD-Architekturen.

Die Verbindungsnetzwerke von MIMD-Systemen lassen sich nach mehreren Kriterien ordnen:

- Topologie
- Verbindungsarten
  - Leitungsvermittlung
  - Paketvermittlung
- Arbeitsweise
  - synchron
  - asynchron
- Verbindungsaufbau
  - verteilt
  - zentral

Da für die Lastverwaltung die topologischen Aspekte des Verbindungsnetzwerkes von entscheidender Bedeutung sind (siehe Kapitel 2 Abschnitt 2.1), wurde in Abbildung 1 für das Klassifikationsmerkmal Verbindungsnetz nur die topologische Aufgliederung verwendet. Sie ist von besonderem Interesse für die Lastverwaltung, da durch entsprechende Plazierung der Prozesse auf den Knoten des Verbindungsnetzwerkes das Kommunikationsaufkommen im Netz beeinflußt werden kann. Für große, sogenannte massiv-parallele Systeme (MP-Systeme) kommen aus Komplexitätsgründen nur reguläre Verbindungsnetzwerke in Betracht. Die irregulären Verbindungsnetzwerke werden deshalb nicht weiter behandelt.
Bei den regulären Verbindungsnetzwerken wird prinzipiell zwischen statischen und dynamischen Netzwerken unterschieden. Während bei statischen Verbindungsnetzwerken (siehe Abbildung 2) eine

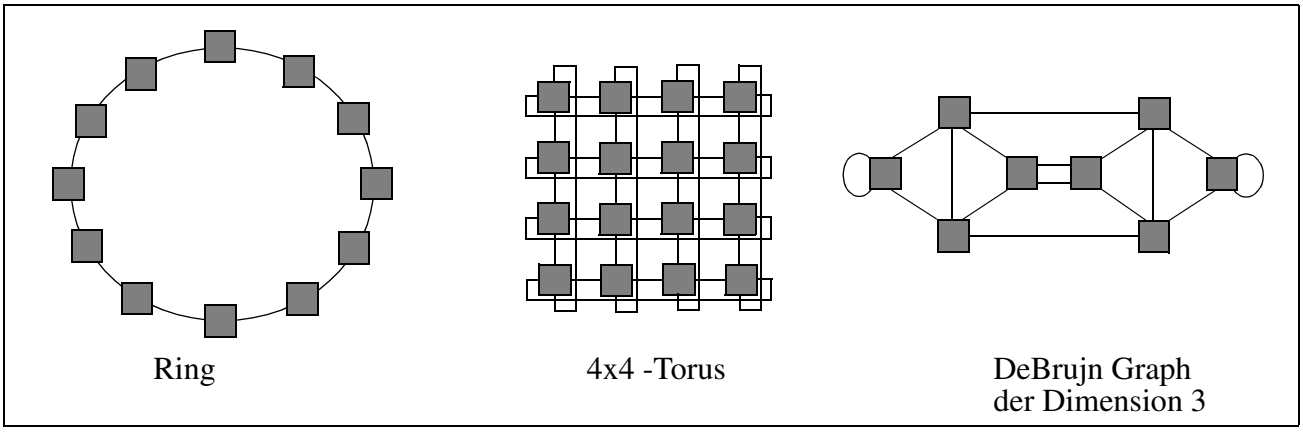


Abbildung 2: Verschiedene statische Netzwerktopologien.

Punkt-zu-Punkt Verbindung zwischen den Knoten (Prozessoren) besteht, wird bei den dynamischen Netzwerken nur bei Bedarf eine Verbindung zwischen den Knoten geschaltet.

Als Beispiele für einstufige, dynamische Verbindungsnetzwerke sind zu nennen:

- Busartige Verbindungsnetzwerke
- Schaltmatrizen (*crossbar switch*)

Als Vertreter für mehrstufige, dynamische Verbindungsnetze [Bräu93] sind zu nennen:

- Omega-Netz
- Banyan-Netz

Ausgehend von dem kurzen Überblick, der in diesem Abschnitt über die verschiedenen Aspekte paralleler Rechnerarchitekturen gegeben wurde, kann nun die Zielarchitektur für die Lastbalancierungsumgebung PaLaBer (**Pa**ralleler **La**st**B**alancie**r**), mit der die in den nachfolgenden Kapiteln vorgestellten Untersuchungen durchgeführt wurden, definiert werden. Als Zielarchitektur werden nachrichtengekoppelte MIMD-Systeme mit verteiltem Speicher und regulären Verbindungsnetzwerken verwendet. In diese Architekturklasse fallen nicht nur die Rechnernetze, sondern auch die meisten MP-Systeme (Intel Paragon, IBM SP2, ...). Die Konzepte des PaLaBer-Systems unterstützen somit eine große Architekturklasse und die erzielten Ergebnisse lassen sich auf eine Vielzahl von Plattformen verallgemeinern.

1.2 Ebenen der Parallelität

Nachdem im vorigen Abschnitt eine Grobklassifikation paralleler Rechnerarchitekturen vorgenommen wurde, wird jetzt auf die verschiedenen Ausprägungen der Parallelität eingegangen. Im Anschluß daran wird das für die weitere Arbeit relevante Lastmodell definiert. Eine moderne parallele Rechnerarchitektur unterstützt aus Leistungsgründen in der Regel mehrere Parallelitätsebenen. In [Wald95] werden fünf Parallelitätsebenen unterschieden:

- Maschinenbefehle
- Anweisungen und Schleifen
- Datenstrukturen
- kooperierende Prozesse
- Benutzerprogramme

Die Parallelitätsebene der *Maschinenbefehle* nutzt die parallelen Verarbeitungseinheiten eines Prozessors aus, um daten-/kontrollflußunabhängige Elementaroperationen parallel auszuführen. Dazu wird vom Übersetzer eine Datenabhängigkeitsanalyse durchgeführt und eine Transformation des Abhängigkeitsgraphen der Elementaroperationen vorgenommen. Die Ausführungsreihenfolge der Elementaroperationen wird dann vom Übersetzer bereits zum Übersetzungszeitpunkt und/oder zur Laufzeit durch eine effiziente Verarbeitungseinheit (*spekulative Programmausführung*) festgelegt. Als Prozessorarchitekturen, die diese Parallelitätsebene unterstützen, sind VLIW-Prozessoren (Very Long Instruction Word), superskalare Prozessoren und Pipeline-Prozessoren zu nennen. Der Parallelitätsgrad, d.h. die Anzahl der Elementaroperationen, die parallel ausgeführt werden können, ist in der Regel gering. Der Vorteil dieser Parallelitätsebene ist jedoch, daß sie weitgehend transparent gegenüber der Programmierschnittstelle ist.
Die Parallelitätsebene der *Anweisungen und Schleifen* betrachtet nun Gruppen von Anweisungen, sogenannte Basisblöcke. Basisblöcke sind Anweisungsfolgen zwischen zwei Kontrollflußanweisungen, die vom Übersetzer daraufhin überprüft werden, ob sie konkurrierend ausgeführt werden können. Von besonderem Interesse sind für den parallelisierenden Übersetzer dabei Schleifenkonstrukte, die, vergleichbar mit der Vektorisierung von Schleifen, daraufhin überprüft werden, ob ihre Schleifenkörper in daten- und kontrollflußunabhängige Basisblöcke zerlegt und somit parallelisiert werden können. Der auf dieser Ebene erzielbare Parallelitätsgrad hängt entscheidend von der Kontrollflußanalyse ab.
Die Parallelitätsebene der *Datenstrukturen* verwendet nur Wissen über den Aufbau der Datenstrukturen zur Parallelisierung. Ein sehr gutes Beispiel für diese Ebene ist die Parallelisierung von Vektor- und Matrixoperationen. In der Praxis finden sich zwei verschiedene Ansätze, wie diese Ebene unterstützt wird. Es gibt einerseits die Möglichkeit, auf diesen Datenstrukturen spezielle Operatoren anzubieten (vgl. HPF - High Performance Fortran), oder andererseits diese Parallelitätsebene weitgehend benutzertransparent durch den Übersetzer unterstützen zu lassen (vgl. konventionelle vektorisierende Übersetzer). Bei diesem Ansatz, der die Datenparallelität der Anwendung ausnutzt, kann der Parallelitätsgrad in Abhängigkeit der Datenstrukturen sehr hoch sein.
Während die ersten drei angeführten Parallelitätsebenen hauptsächlich den Übersetzer bzw. den Prozessor betreffen, sind die nächsten beiden Ebenen für die Lastverwaltung von besonderer Bedeutung, speziell für die im vorigen Abschnitt beschriebene *Shared-Nothing*-Architektur.
Die Parallelitätsebene der *kooperierenden Prozesse* ist für Shared-Nothing-Architekturen die wohl wichtigste Parallelitätsebene. Dabei wird die Anwendung in mehrere möglichst unabhängige Prozesse zergliedert. Der Informationsaustausch und die Synchronisation zwischen den Prozessen der parallelen Anwendung erfolgt dabei über das explizite Senden von Nachrichten. Da der Prozeßbegriff für diese Ebene von großer Bedeutung ist, wird nachfolgend eine Definition angegeben.

Definition 1: Ein Prozeß ist eine funktionelle Einheit, bestehend aus einem zeitlich invarianten Programm, einem Satz von Daten, mit dem der Prozeß initialisiert wird, und einem zeitlich varianten Zustand [Gilo93].

Da in der Prozeßdefinition von Giloi keine Angaben über das Prozeßgranulat gemacht werden, kann diese Definition sowohl für feingranulare (z.B. leichtgewichtige Prozesse, Threads) als auch für grobgranulare Prozesse (z.B. Unix-Prozesse) verwendet werden. In dieser Arbeit wird unter einem Prozeß eine grobgranulare Einheit entsprechend eines Unix-Prozesses verstanden.
Ein paralleles Programm besteht somit aus einer Menge von sequentiellen Prozessen, die über Nachrichten miteinander in Beziehung stehen. Zur Modellierung des parallelen Programms bietet sich ein Graph an. Mit Hilfe eines Programmgraphen kann dabei sowohl der Kontrollfluß als auch der Datenfluß innerhalb der parallelen Anwendung modelliert werden. Der Kontrollfluß modelliert die zeitlichen Beziehungen zwischen den Prozessen, der Datenfluß den Informationsaustausch zwischen

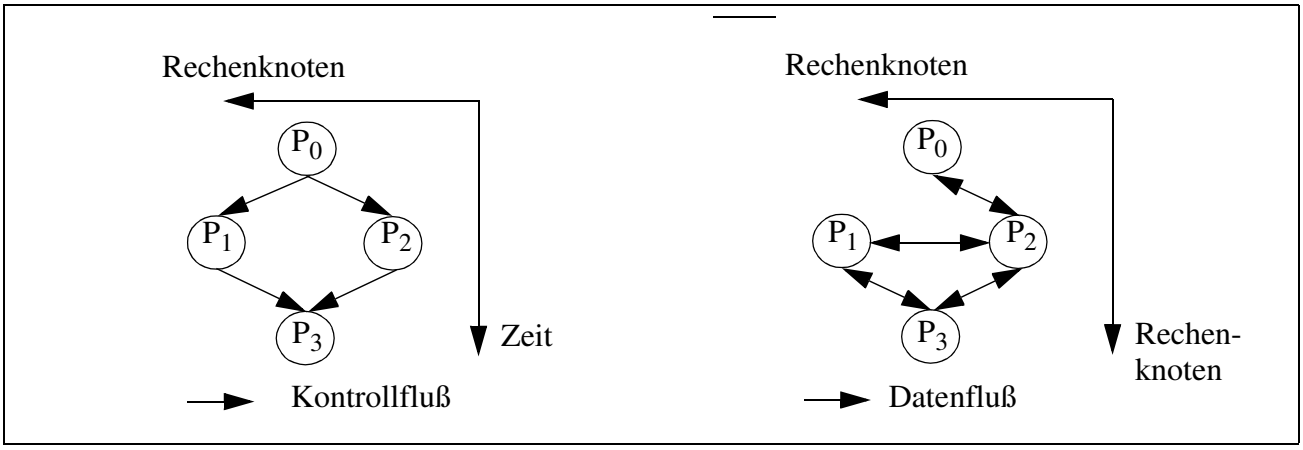


Abbildung 3: Beispiel für einen TPG und einen TIG.

ihnen. Im Falle der Kontrollflußmodellierung spricht man vom *Task-Präzedenzgraphen* (kurz TPG) [BeWa94], im Falle der Datenflußmodellierung vom *Task-Interaktionsgraphen* (kurz TIG) (siehe Abbildung 3).

Zur Lastmodellierung im PaLaBer-System wird eine Kombination des Interaktions- und Präzedenzgraphen verwendet. Ausgangspunkt für die Lastmodellierung ist ein TIG, bei dem die Prozeßmenge jedoch nicht notwendigerweise statisch sein muß, d.h. es können vergleichbar zum TPG zur Laufzeit Prozesse generiert bzw. terminiert werden. Die einzige Annahme über den Programmgraphen ist, daß der überwiegende Teil der Prozesse zum Startzeitpunkt t_0 generiert wird (siehe Abbildung 4). Im weiteren wird ein Interaktionsgraph, der diese Eigenschaft erfüllt, als *nicht statischer TIG* bezeichnet. In Abbildung 4 ist ein nicht statischer TIG bestehend aus 5 Prozessen dargestellt. Während die Prozesse P_0 bis P_3 bereits zum Startzeitpunkt der Anwendung generiert werden, wird der Prozeß P_4 erst im Lauf der Berechnung erzeugt.

Die letzte Parallelitätsebene ist die Ebene der *Benutzerprogramme*, bei der der Rechner simultan mehrere Benutzerprogramme abarbeitet. Diese Ebene wird durch den Mehrprozeßbetrieb realisiert. Dabei spielt es keine wesentliche Rolle, ob die simultan laufenden Anwendungen einem oder mehreren Anwendern zugeordnet werden können. Der Mehrprozeßbetrieb kann in parallelen Rechnerarchitekturen prinzipiell auf zwei Arten realisiert werden. Während beim Multitasking-Betrieb ein Prozessor durch ein Zeitscheibenverfahren von Prozessen verschiedener Anwendungen quasi-parallel genutzt wird, wird beim Space-Sharing eine Gruppe von Prozessoren jeweils einer parallelen Anwendung exklusiv zur Verfügung gestellt. Diese beiden Realisierungsmöglichkeiten des Mehrprozeßbetriebs unterscheiden sich nicht nur in ihrem Ansatz, sondern auch in ihrer Zielsetzung. Während mit Hilfe des Multitasking-Betriebs der Durchsatz der Rechenanlage gesteigert werden kann, wird beim Space-Sharing-Verfahren die Laufzeit einer einzelnen Anwendung gesteigert. In den meisten modernen, parallelen Rechnerarchitekturen wird derzeit nur das Space-Sharing-Verfahren angewandt.

Im Hinblick auf eine effiziente und somit kostengünstige Auslastung der Rechenanlage unterstützt das PaLaBer-System den Multitasking-Betrieb, d.h. die Prozessoren des parallelen Systems werden gleichzeitig von mehreren parallelen Anwendungen konkurrierend genutzt. Das vom PaLaBer-System unterstützte Anwendungsmodell besteht somit aus einer Gruppe von nicht statischen TIG‘s, die konkurrent abgearbeitet werden. Dieses Lastmodell wird nachfolgend mit *multiple non-static TIG* (MNTIG) bezeichnet.

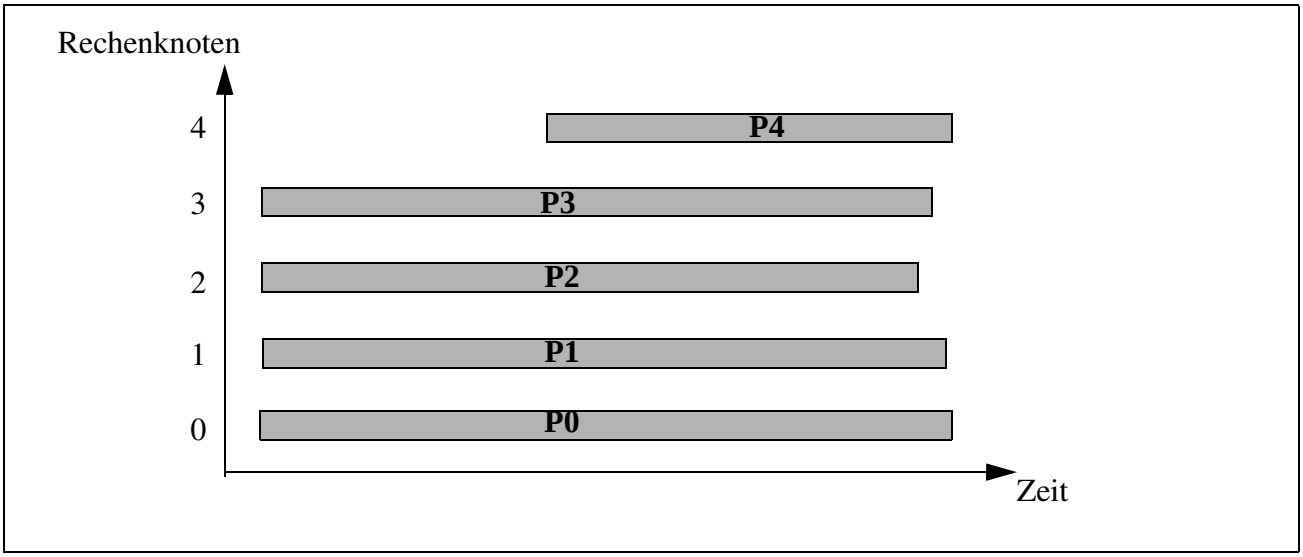


Abbildung 4: Beispiel eines nicht statischen TIG.

1.3 Aufgabenstellung der Lastverwaltung und Lösungsansätze

In den vorigen zwei Abschnitten wurden sowohl die parallele Zielarchitektur als auch die für die weitere Arbeit relevanten Parallelitätsebenen vorgestellt. Ziel dieses Abschnittes ist es nun, anhand der Zielarchitektur und des Anwendungsmodells (MNTIG) die Aufgaben der Lastverwaltung aufzuzeigen. Für eine ausführliche Literaturbesprechung dieses Themengebietes wird auf Kapitel 2 verwiesen.
Aufgabe der Lastverwaltung ist es, bei einer gegebenen Arbeitslast, in unserem Fall einer Menge von nicht statischen TIG‘s, die zur Verfügung stehenden Ressourcen der Rechnerarchitektur bestmöglichst auszunutzen. Bei der Abbildung der Arbeitslast auf die verfügbaren Ressourcen muß sowohl der zeitlich variante Ressourcenbedarf der Anwendungen als auch die zeitlich variante Verfügbarkeit der Systemressourcen berücksichtigt werden. Unabhängig davon, welches Optimierungskriterium der Lastverwalter verfolgt (Durchsatzsteigerung, Laufzeitoptimierung einer einzelnen Anwendung, etc.), stehen ihm drei sich zum Teil ergänzende Lastverwaltungsansätze zur Verfügung:

1. statische Lastbalancierung
2. dynamisches Remapping
3. dynamische Lastbalancierung

Diese drei Lastverwaltungsansätze, auf die in Kapitel 2 noch ausführlich eingegangen wird, lassen sich anhand der folgenden Merkmale klassifizieren:

- zeitliches Verhalten
- unterstützte Anwendungscharakteristik
- Arbeitsprinzip (*Steuerungs- oder Regelungsprinzip*)
- algorithmische Ansätze
- Informationsverwendung

Das wohl auffälligste Unterscheidungsmerkmal der drei Lastverwaltungsansätze ist deren *zeitliches Verhalten* (siehe Abbildung 5). Beim statischen Lastverwaltungsansatz wird vor dem eigentlichen

Programmlauf (evtl. schon zum Übersetzungszeitpunkt) eine möglichst gute Lastverteilung der Anwendungsprozesse auf die Knoten der parallelen Rechnerarchitektur ermittelt. Im Vergleich dazu wird beim dynamischen Remapping und bei der dynamischen Lastbalancierung auch während des Programmlaufes auf die Lastverteilung eingewirkt. Beim dynamischen Remapping findet bei Bedarf

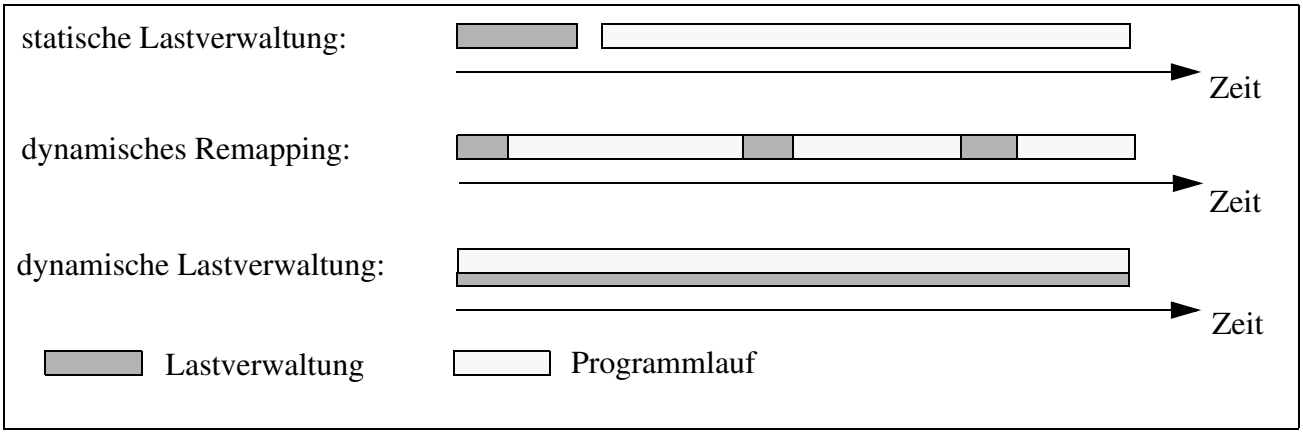


Abbildung 5: Zeitliches Verhalten verschiedener Lastverwaltungsansätze.

zu diskreten Zeitschritten eine Neuverteilung der Arbeitslast auf die Systemressourcen statt, hingegen ist bei der dynamischen Lastbalancierung die Lastverteilung ein kontinuierlicher Prozeß, der parallel zum Programmlauf stattfindet.
Das zeitliche Verhalten der Lastverwaltungsansätze stellt natürlich gewisse Anforderungen an die *Anwendungscharakteristik*, d.h. an das Lastverhalten und somit an den Ressourcenbedarf der parallelen Anwendungen. Um mit Hilfe der statischen Lastbalancierung eine gute Lastverteilung erreichen zu können, müssen mehrere Kriterien erfüllt sein:

1. Die Prozesse der parallelen Anwendung müssen ein homogenes oder zumindest sinnvoll modellierbares Lastverhalten aufweisen, da alle statischen Lastverwaltungsverfahren auf ein abstraktes Lastmodell der parallelen Anwendung angewiesen sind.
2. Das Lastverhalten der Prozesse muß bereits vor dem Programmlauf ermittelbar, zumindest jedoch abschätzbar sein, da dies ein wichtiger Startparameter eines jeden statischen Lastbalancierers darstellt.
3. Die Knoten bzw. eine Knotengruppe der parallelen Rechnerarchitektur müssen der parallelen Anwendung exklusiv zur Verfügung stehen, da die statische Lastbalancierung keine Informationen über den aktuellen Systemzustand in ihre Berechnung einbezieht.

Aufgrund dieser sehr restriktiven Anforderungen an die Anwendungscharakteristik ist die statische Lastbalancierung natürlich nicht für alle Anwendungsklassen anwendbar. Diese Anforderungen werden beim dynamischen Remapping und bei der dynamischen Lastbalancierung sukzessive abgeschwächt. Während das dynamische Remapping davon ausgeht, daß eine parallele Anwendung ausreichend lange homogene Verarbeitungsphasen aufweist, für die es sich lohnt, jeweils eine komplette Neuverteilung zu berechnen, stellt die dynamische Lastbalancierung keine Anforderungen an die Anwendungscharakteristik.
Das dritte Klassifikationsmerkmal, das *Arbeitsprinzip,* unterscheidet die Lastverwaltungsansätze danach, ob sie nach dem Steuerungs- oder Regelungsprinzip arbeiten (siehe Abbildung 6). Der wesentliche Unterschied zwischen der Steuerung und Regelung besteht darin, daß es sich bei der Regelung um einen rückgekoppelten Prozeß handelt. Dementsprechend arbeitet die statische Lastbalancierung nach dem Steuerungsprinzip während dynamisches Remapping und dynamische

Lastbalancierung nach dem Regelungsprinzip arbeiten, da sie die aktuelle Lastsituation, die von ihnen mitbeeinflußt wird, in ihren Entscheidungsprozeß miteinbeziehen.

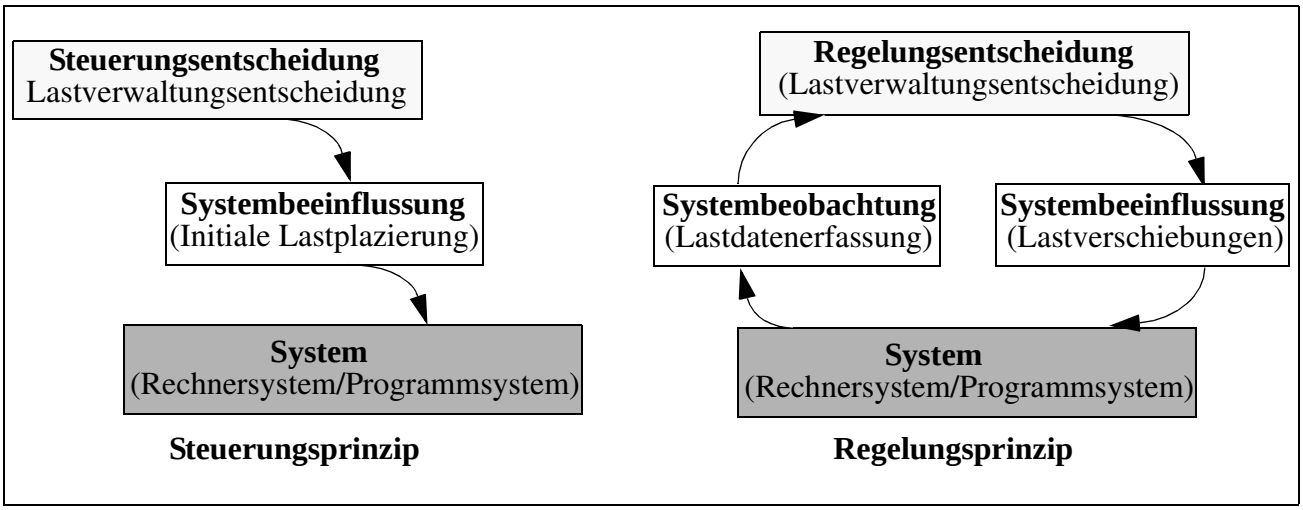


Abbildung 6: Steuerungs- und Regelungsprinzip der Lastverwaltung.

In einem engen Zusammenhang mit den zeitlichen Rahmenbedingungen der Lastverwaltungsansätze sind die *algorithmischen Ansätze* zu sehen. Generell gehört die Problemstellung der Lastverwaltung in die Klasse der NP-vollständigen Probleme [GaGJ78][GaJo79], d.h. eine im Hinblick auf das jeweilige Optimierungskriterium optimale Lösung ist deterministisch nur mit einem Zeitaufwand erzielbar, der exponentiell mit der Problemgröße wächst. Deshalb werden für die meisten praxisrelevanten Lastverwaltungsansätze suboptimale bzw. heuristische Lösungsansätze verwendet. Dies bedeutet jedoch nicht, daß es für spezielle Anwendungsfälle, d.h. für Programmgraphen mit einer speziellen Struktur, nicht auch optimale Lösungen von geringerer Zeitkomplexität gibt [Bokh88][KaNa84]. Prinzipiell kann jedoch festgehalten werden, daß für die Lösung der statischen Lastbalancierung sehr komplexe Algorithmen verwendet werden. Für das Remapping und die dynamische Lastbalancierung sind Lösungen mit geringerer Zeitkomplexität notwendig, da die Berechnungen in diesen Fällen zur Ausführungszeit der Anwendung durchgeführt werden.
Das letzte Klassifikationsmerkmal, die *Informationsverarbeitung,* unterscheidet die Lastverwaltungsansätze in zweierlei Hinsicht. Erstens im Hinblick auf die Komplexität der verwendeten Information und zweitens im Hinblick auf die Aktualität der Information, die im Entscheidungsprozeß berücksichtigt werden kann. Die statische Lastbalancierung kann aufgrund der Entkopplung der Lastverteilungsberechnung und des Anwendungslaufes sehr komplexe Informationen (z.B. Kommunikationsaufkommen zwischen den Anwendungsprozessen, etc.) verwenden. Diese Entkopplung bedingt jedoch auch, daß die statische Lastbalancierung auf statische bzw. statistische Informationen über das Laufzeitverhalten der parallelen Anwendungen bzw. der Betriebssystemcharakteristik angewiesen ist. Im Vergleich dazu können die beiden anderen Ansätze den aktuellen Lastzustand des Systems in ihre Lasttransferentscheidungen miteinbeziehen. Allerdings sind sie aufgrund der zeitlichen Restriktionen nicht bzw. nur sehr bedingt in der Lage, komplexe Informationen und Informationsbeziehungen in ihren Entscheidungsprozeß miteinzubeziehen.

1.4 Zielsetzung der Arbeit

Wie in diesem Kapitel kurz aufgezeigt, stellen parallele Rechnersysteme eine geeignete Plattform zur Lösung der sogenannten *Grand Challenges* dar. Voraussetzung dafür ist jedoch, daß die Ressourcen

des parallelen Systems effizient ausgelastet werden. Wie im Abschnitt 1.3 dargestellt wurde, gibt es drei prinzipielle Realisierungsmöglichkeiten für eine Lastverwaltung. Da von diesen drei Varianten die dynamische Lastbalancierung die geringsten Anforderungen an das Anwendungsprofil stellen, beschäftigt sich diese Arbeit in den nachfolgenden Untersuchungen in erster Linie mit der dynamischen Lastbalancierung.
Insbesondere setzt sich die vorliegende Arbeit in den folgenden Kapiteln mit der Fragestellung auseinander, inwieweit die Effizienz eines dynamischen Lastverwalters durch eine system- und anwendungsspezifische Informationsschnittstelle positiv beeinflußt werden kann. Dabei sind zwei Untersuchungsschwerpunkte von besonderer Bedeutung:

1. Können durch eine entsprechende Informationsversorgung die Laufzeitgewinne bzw. Durchsatzsteigerungen, die in einer Vielzahl von Veröffentlichungen [Beck95][Ludw93] im Berecih von 10% bis 20% liegen, noch weiter gesteigert werden?
2. Können die erzielbaren Gewinne in dem Sinne stabilisiert werden, daß sie ohne Veränderung des dynamischen Lastverwaltungsverfahrens für eine große Palette von Anwendungen mit unterschiedlichster Anwendungscharakteristik erreichbar sind, indem die dynamische Lastverwaltung mit entsprechenden System- und Anwendungsinformation versorgt wird?

Speziell die zweite Fragestellung ist von großer Bedeutung, da die in der Literatur veröffentlichten Ergebnisse in der Regel mit anwendungspezifischen Lastverwaltungsverfahren erzielt wurden. D.h. mit Verfahren, die auf das jeweilige Lastverhalten einer Anwendung bzw. einer Anwendungsgruppe hin optimiert wurden und somit nicht als allgemeine Lastverwaltungskomponente in einem parallelen System verwendet werden können.

Kapitel 2

Lastverwaltungsverfahren

In diesem Kapitel sollen aus den verschiedenen Ansätzen zur Lastverwaltung einige in der Literatur beschriebene Verfahren exemplarisch vorgestellt werden. Dabei liegt der Beschreibungsschwerpunkt auf den verschiedenen Lösungsansätzen. Auf den Aspekt der Informationsverwendung wird dann detailliert in Kapitel 3 eingegangen.
Wie bereits im ersten Kapitel beschrieben, gibt es zur Lastverwaltung in parallelen und verteilten Systemen prinzipiell drei Ansätze:

- statische Lastbalancierung
- dynamisches Remapping
- dynamische Lastbalancierung

Alle drei Ansätze werden in diesem Kapitel beschrieben, wobei der Schwerpunkt aufgrund der Zielsetzung dieser Arbeit auf der dynamischen Lastbalancierung liegen wird.

2.1 Statische Lastbalancierung

Bei der statischen Lastbalancierung [Bokh81] wird vor der eigentlichen Programmausführung, z.B. bereits zum Übersetzungszeitpunkt, eine möglichst gute Abbildung der Arbeitslast auf die zur Verfügung stehenden Ressourcen ermittelt. Monetäre Gesichtspunkte ausgegrenzt, besteht bei der Berechnung der Lastverteilung nur die Restriktion, daß die erzielbaren Laufzeitgewinne der parallelen Anwendung größer sein müssen als die Berechnungsdauer für die Ermittlung der statischen Verteilung. Viele der Verfahren zur statischen Lastbalancierung lassen sich auf die Lösung eines diskreten Optimierungsverfahrens der Form

$$\varphi(x_1, \ldots, x_n) \rightarrow \min$$

unter Berücksichtigung gewisser Randbedingungen der Art

$$\Sigma_{1 \leq j \leq n} x_j = C \text{ mit } x_j \in N, j = 1, \ldots, n$$

zurückführen. Bei diesen Optimierungsverfahren handelt es sich, wie [Heis94] schreibt, um „notorisch NP-harte Probleme“ [GaJo79]. Auch Verfahren, die den Suchraum drastisch eingrenzen und dennoch in der Lage sind, eine optimale Lösung zu ermitteln, wie z.B. das Branch-and-Bound Verfahren, haben häufig eine für praktische Zwecke zu hohe Zeitkomplexität. Aus diesem Grund ist man auch bei der statischen Lastbalancierung, speziell im Hinblick auf große Probleme, häufig auf heuristische Suchverfahren angewiesen. In der Literatur wird eine Vielzahl von heuristischen Ansätzen zur Lösung des diskreten Optimierungsproblems vorgestellt:

- Gradientenabstiegsverfahren [Heis94]
- Sondierungspfade [Heis94]
- dynamische Programmierung [Bogl92][ChNa91]
- Akzeptanz von Verschlechterung
  - Simulated Annealing [BoMi88][HwXu90]
- Parallele Suchverfahren
  - Iteration [Heis94]
  - Genetische und Evolutionäre Algorithmen [SrPa94]
  - Hopfieldnetze [KaKa90][KGDK90]
  - Mean-Field Annealing [BuMa93]

Exemplarisch für diese Vielzahl von heuristischen Lösungsverfahren für das diskrete Optimierungsproblem wird das Simulated Annealing Verfahren ausführlicher beschrieben. Dieser Lösungsansatz wurde in den letzten Jahren in verschiedenen Arbeiten zur Ermittlung einer statischen Lastverteilung erfolgreich verwendet.

2.1.1 Simulated Annealing

Beim Simulated Annealing [KiGV83] wird der Abkühlvorgang einer Materie vom flüssigen in den festen Aggregatzustand simuliert. Das Simulated Annealing geht dabei auf einen Algorithmus von Metropolis et al. [MRRTT53] von 1953 zurück. Metropolis simulierte mit Hilfe eines Monte-Carlo-Verfahrens den Abkühlvorgang sogenannter Spingläser. Damit sich beim Abkühlvorgang dieser Gläser eine möglichst regelmäßige Gitterstruktur der Moleküle und somit ein möglichst energiearmer Zustand ergibt, muß das Material sehr vorsichtig und langsam abgekühlt werden. Ausgehend von einer sehr hohen Temperatur, bei der die einzelnen Moleküle des Materials noch sehr energiereich, d.h. bewegungsfähig, sind und sich dadurch noch erhebliche Änderungen in der Molekülanordung ergeben können, wird mit abnehmender Temperatur die Bewegungsfähigkeit der Moleküle geringer und die Molekülanordnung statischer. Die Bewegungen der Moleküle entsprechen dabei kleinen stochastischen Modifikationen des Systemzustandes.
Der Algorithmus geht nun davon aus, daß Systemzustände, die durch eine Modifikation der Anordnung entstehen und eine höhere Energie als der unmittelbare Ausgangszustand aufweisen, eine Verschlechterung der zu optimierenden Zielfunktion ($\Delta\varphi \geq 0$) darstellen und nur mit einer Wahrscheinlichkeit von $\exp(-\Delta\varphi/T)$ angenommen werden. Der Temperaturparameter T wird dabei schrittweise herabgesetzt und somit die Wahrscheinlichkeit, eine Zustandsverschlechterung zu akzeptieren, verringert. Hingegen werden Systemzustandverbesserungen immer akzeptiert. Dies hat zur Folge, daß mit abnehmender Temperatur die Wahrscheinlichkeit zunimmt, daß das Suchverfahren in einem lokalen Minimum stecken bleibt und sich ein statischer Systemzustand einstellt.
Ein statischer Lastbalancierungsansatz, der basierend auf dem Simulated Annealing Ansatz versucht, eine möglichst gute initiale Plazierung zu ermitteln, ist der Ansatz von Hwang und Xu [HwXu90].

Bei diesem Ansatz wird sowohl das Rechnersystem, in diesem Fall ein Parallelrechner mit verteiltem Speicher, als auch die parallele Anwendung als Graph dargestellt [NoTh93](siehe Abbildung 7). Im Programmgraph werden die einzelnen Module der parallelen Anwendung durch Knoten, die Kommunikationsverbindungen zwischen den Modulen durch ungerichtete Kanten des Programmgraphs (TIG) modelliert. Im Maschinengraph stellen die Knoten die Prozessoren dar, die Kanten die Kommunikationskanäle zwischen den Prozessoren. Die Distanz d_{ij} zwischen zwei Prozessoren entspricht dabei der Anzahl der zu überquerenden Kommunikationsverbindungen (*hops*) zwischen den beiden Prozessoren. Wie bei allen statischen Lastverwaltungsverfahren muß auch beim Simulated Annealing zumindest eine grobe Abschätzung des Rechenzeitbedarfes (m_i) und für jede Kommunikationsverbindung zwischen zwei Modulen eine Abschätzung des Nachrichtenaufkommens (e_{ij}) vorliegen.
Um das Simulated Annealing Verfahren anwenden zu können, ist eine formale Beschreibung des Lastzustandes und der zu minimierenden Kostenfunktion erforderlich. Hwang und Xu stellen den Lastzustand des Parallelrechners bestehend aus n Knoten als Lastvektor $L = \{l_i \mid i = 0,1, .., n\}$ dar. Die Last l_i eines einzelnen Knotens ergibt sich dabei aus der Summe der zugewiesenen Module ($l_i = \Sigma_j m_j$). In der Kostenfunktion E wird dabei sowohl die Prozessorlast als auch die Kommunikationslast berücksichtigt:

$$E = E_{imbalance} + E_{communication} = \sum_{0\le i\le n} \mid l_i - (1/n \times \sum_{0\le j\le n} l_j) \mid + 1/2 \times \sum_{0\le i\le n} \sum_{0\le j\le n} (d_{ij} \times \sum_x \sum_y e_{xy})$$

Die Größe e_{xy} beschreibt dabei das Kommunikationsaufkommen zwischen allen Programmodulen M_x auf Knoten i und allen Programmodulen M_y auf Knoten j. Das Optimierungsverfahren beginnt mit einer initialen Lastverteilung L_0, die durch sukzessive Manipulationen mit Hilfe des Simulated Annealing Verfahrens zu verbessern versucht wird.

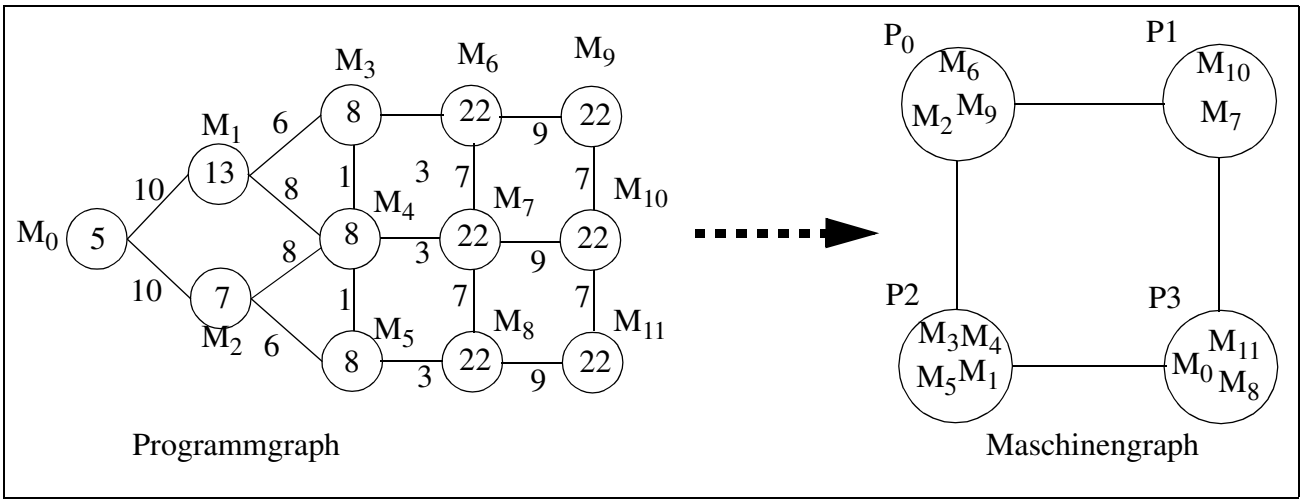


Abbildung 7: Abbildung eines Programmgraphen auf einen Prozessorgraph.

2.2 Dynamisches Remapping

Für das dynamische Remapping können prinzipiell die gleichen Lösungsverfahren verwendet werden wie für die statische Lastbalancierung. Da die Berechnungszeit einer Lastneuverteilung beim dynamischen Remapping bereits zur Laufzeit der Anwendung mitgerechnet wird, ist die Zeitkomplexität des Lösungsverfahrens von entscheidender Bedeutung. Deshalb finden bei der Problemstellung des dynamischen Remappings verstärkt heuristische Verfahren Anwendung.
Da sich die algorithmischen Ansätze des dynamischen Remapping nicht grundsätzlich von denen der statischen Lastbalancierung unterscheiden, wird in diesem Abschnitt der Schwerpunkt auf die

Fragestellung gelegt, wann eine Lastneuverteilung durchgeführt werden soll, bzw. anhand welcher Kriterien ein Lastverwalter in die Lage versetzt werden kann, einen möglichst günstigen Zeitpunkt zu ermitteln. Zur Lösung dieser Fragestellung gibt es generell zwei Lösungsansätze. Entweder ergibt sich der Zeitpunkt für eine Lastneuverteilung in natürlicher Weise durch die Anwendung bzw. durch deren Lastverhalten oder der Lastverwalter muß durch die Systembeobachtung die Kosten für eine Lastneuverteilung gegen den potentiellen Laufzeitgewinn abwägen. Für beide Lösungsansätze werden in den beiden nachfolgenden Unterabschnitten ausgewählte Verfahren vorgestellt.

2.2.1 Der Ansatz von Choudhary et al.

In der Arbeit von Choudhary et al. [CoNK93] wird ein anwendungsspezifischer, dynamischer Remapping-Ansatz für die Stereobildverarbeitung vorgestellt. Die zugrundegelegte Anwendungsstruktur ist dabei eine Pipeline (siehe Abbildung 8), wobei sich die einzelnen Verarbeitungsstufen der Pipeline als eindimensionaler TIG darstellen lassen.

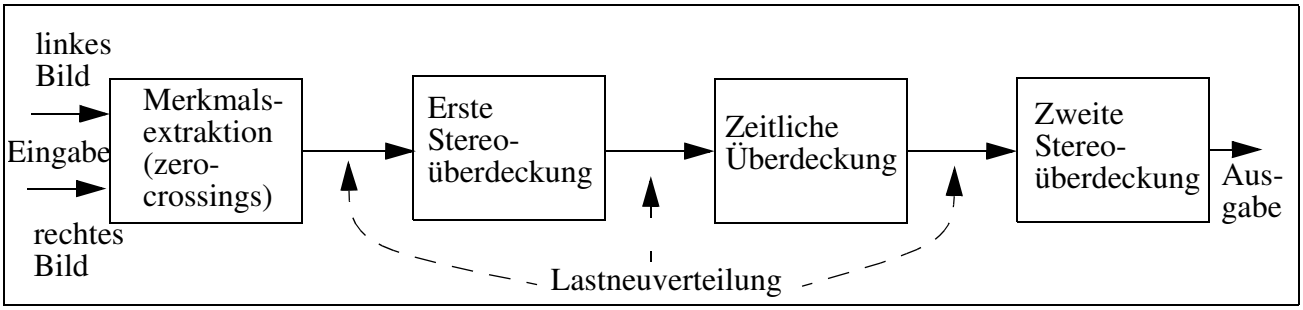


Abbildung 8: Anwendungsstruktur.

Als Eingabe erhält die vierstufige Pipeline zwei Bilder, die jeweils entlang einer Dimension partitioniert werden, wodurch sich der eindimensionale TIG erklärt. Da der Berechnungsaufwand der vier Stufen erheblich variiert und zudem noch stark datenabhängig ist, kann dieses Lastverwaltungsproblem nicht durch eine einmalige, statische Bildpartitionierung zufriedenstellend gelöst werden. Stattdessen muß nach jeder Pipelinestufe eine neue Bildpartitionierung durchgeführt werden.
In [ChNa91] wurde bereits nachgewiesen, daß die Berechnung einer optimalen Lastverteilung bei parallelen Anwendungen mit einer pipelineartigen Struktur eine Zeitkomplexität von O(MP) hat, wobei M die Anzahl der Anwendungsmodule und P die Anzahl der Prozessoren ist. Der Algorithmus basiert auf dem Arbeitsprinzip der dynamischen Programmierung und einer Vorverarbeitung mit einer Zeitkomplexität von O(M). Somit gehört dieser Spezialfall der Lastverwaltung nicht in die Klasse der NP-harten Probleme. Trotzdem ist die Berechnung einer optimalen Lösung für große M bzw. P im Zusammenhang mit dem dynamischen Remapping nicht durchführbar, da die Berechnung der Lastneuverteilung bereits zur Ausführungszeit der Anwendung mitgerechnet wird. Aus diesem Grund wird in [CoNK93] ein heuristischer Ansatz mit der Zeitkomplexität O(M) präsentiert, inklusive einer Abschätzung der Lastverteilungsqualität. Zur Beschreibung des Remapping-Verfahrens ist die Einführung einiger Begriffe und Definitionen erforderlich.
Sei π: {0, ..., M-1} $\rightarrow$ {0, ..., P-1} die Abbildungsfunktion der Anwendungsmodule der jeweiligen Pipelinestufe auf die zur Verfügung stehenden Prozessoren. Da aus Leistungsgründen unmittelbar aufeinanderfolgende Anwendungsmodule des TIG auf den selben bzw. zumindest auf einen benachbarten Prozessor plaziert werden sollten, gilt für die Abbildungsfunktion π weiter $\pi(i) = k$ oder $\pi(i) = k+1$. Demzufolge bekommt jeder Prozessor j, $0 \leq j \leq P-1$, eine zusammenhängende Folge von Berechnungsmodulen (start[j], end[j]). Die Berechnungslast, die von einem Anwendungsmodul i

verursacht wird, wird mit Load[i] bezeichnet. Die Last eines Prozessors k (bezeichnt mit λ_k) ergibt sich somit aus der Summe aller Anwendungsmodule auf dem Prozessor. Mit $C(\pi) = \max_{0 \le k \le P-1}\{\lambda_k\}$ bezeichnen die Autoren die Kosten einer Lastneuverteilung und mit C* die Kosten der optimalen Lastverteilung. Die letzten beiden Größen, die noch benötigt werden um den Algorithmus zu beschreiben, sind $\varepsilon = \max_{0 \le i \le M-1}\{\text{Load}[i]\}$, das Anwendungsmodul mit der größten Berechnungslast, und μ die durchschnittliche Prozessorlast. Der Lastverteilungsalgorithmus beschrieben in Pseudo-Code lautet wie folgt:

```
begin
        sum := Σ_{0≤i≤M-1} Load[i]          /* Berechnungsaufwand aller Module */
        μ := sum / P                        /* durchschnittlicher Berechnungsaufwand */
        avg := μ
        j := 0
        start[0] := 0                       /* Erstes Modul wird von Prozessor 0 bearbeitet */
        end[p-1] := M-1                     /* Letztes Modul wird von Prozessor P-1 ... */
        low := false
        for i:=0 to P-2 do begin
                tempload := 0
                while(tempload < avg) do begin
                        tempload := tempload + Load[j]      /* Abbildung von Modul i auf Proz. j */
                        j := j+1
                endwhile
                if(low = true) then begin                   /* Korrekturschritt */
                        j := j - 1
                        tempload := tempload - Load[j]
                endif
                end[i] := j - 1
                start[i+1] := j
                sum := sum - tempload
                avg := sum / (P-i-1)        /* durchschnittlicher Berechnungsaufwand der restl. Module */
                if(avg < μ) then low := true
                            else low := false
        endfor
end.
```

Den Prozessoren werden in aufsteigender Reihenfolge ihre Anwendungsmodule zugeteilt. Da es in der Regel aufgrund des Partitionsgranulats nicht möglich ist, einem Prozessor exakt die durchschnittliche Berechnungslast zuzuweisen, wird dem ersten Prozessor eine Berechnungslast kleiner gleich der durchschnittlichen Berechnungslast zugeweisen, dem zweiten dann eine Berechnungslast, die größer ist als die Durchschnittliche usw.. Dabei wird in der Hilfsgröße *avg* die verbleibende, durchschnittliche Berechnungslast gespeichert. Da bei diesem Algorithmus jedes Berechnungsmodul exakt einmal *angefaßt* wird, ist die Zeitkomplexität folgerichtig O(M). In [CoNK93] haben die Autoren auch eine Qualitätsabschätzung ihres Verfahrens gegenüber der optimalen Lastverteilung angegeben und kommen zu der Beziehung $C(\pi) \le 5/2\ C^*$.

2.2.2 Der Ansatz von Nicol/Saltz

Die Arbeit von Nicol und Saltz [NiSa88] beschäftigt sich mit der wichtigen Frage, wann eine globale Neuverteilung der Arbeitslast vorgenommen werden soll, wenn sich dieser Zeitpunkt nicht in natürlicher Weise durch die Anwendung bzw. deren Lastverhalten ergibt. Da die Antwort auf diese Frage stark von der jeweiligen Arbeitslast bzw. von deren Lastverhalten abhängt, schränken sie ihre Lösung auf folgende Anwendungsklasse ein. Betrachtet werden parallele Anwendungen, die zyklisch

Rechen- und Kommunikationsphasen durchlaufen. Dabei handelt es sich bei den Kommunikationsphasen um globale Synchronisationsphasen und die Dauer der Rechenphasen verändert sich im Laufe der Berechnung. Dieser Anwendungsklasse gehören zum Beispiel gitterbasierte Anwendungen an, bei denen sich entweder zur Laufzeit der Rechenbedarf pro Gitterpunkt ändert oder zur Laufzeit die Gitterauflösung partiell verändert wird. Zusätzlich zu dieser Annahme über das Anwendungsverhalten basiert der Algorithmus von Nicol und Saltz noch auf zwei weiteren realitätsnahen Annahmen:

1. Die Verzögerungen, die durch den Vorgang der globalen Lastneuverteilung entstehen, sind bekannt.
2. Es ist möglich, für jeden Prozessor die Leerlaufzeit seit der letzten Neuverteilung zu erfassen.

Der SAR-Algorithmus (*stop at rise*) zur Bestimmung, wann die nächste Lastneuverteilung durchgeführt werden soll, lautet wie folgt:

Sei $T_j(n)$ die Dauer der Berechnungsphase von Prozessor j in der n-ten Iteration. Da nach jeder Rechenphase eine globale Synchronisation durchgeführt wird, ergibt sich die Dauer der n-ten Iteration bei N Prozessoren aus $T_{max}(n) = \max_{1 \le j \le N} \{T_j(n)\}$. Folglich berechnet sich die durchschnittliche Bearbeitungsdauer der n-ten Iteration aus $\tilde{T}(n) = \frac{1}{N}\left(\sum_{1=j}^{N} T_j(n)\right)$ und die durchschnittliche Prozessorleerlaufzeit aus $T_{max}(n) - T(n)$.

Zur eigentlichen Entscheidungsfindung wird nun die Kostenfunktion

$$W(n) = \frac{\sum_{j=1} (T_{max}(j) - T(j)) + C}{n}$$

verwendet. Die Konstante C steht dabei für die Kosten einer Lastneuverteilung, die Summe ist die durchschnittliche Prozessorleerlaufzeit seit der letzten Neuverteilung. Das Ziel des SAR-Algorithmuses ist nun, sobald die Funktion W(n) ihr erstes lokales Minimum erreicht, eine Neuverteilung der Arbeitslast zu initiieren. Aufgrund der oben angeführten Annahmen und umfangreichen Simulationsstudien kamen die Autoren zu dem Ergebnis, daß die Funktion W(n) i.d.R. im Laufe der ersten Iterationen streng monoton fällt, um dann später streng monoton zu steigen. Basierend auf dieser Feststellung bzw. Annahme empfehlen die Autoren genau dann eine Lastneuverteilung durchzuführen, sobald die Bedingung

$$W(n) > W(n-1)$$

erfüllt ist. Die Leistungsfähigkeit ihres SAR-Algorithmus haben die Autoren mit zwei Referenzverfahren (keine Lastneuverteilung und Lastneuverteilung alle m Iterationen) verglichen und konnten nachweisen, daß es mit Hilfe des SAR-Verfahrens möglich ist, deutliche Laufzeitverbesserungen zu erzielen.

2.3 Dynamische Lastbalancierung

In diesem Abschnitt wird ein Überblick über die verschiedenen Verfahren zur dynamischen Lastbalancierung zu geben. Da im Bereich der Lastverwaltung seit über 20 Jahren intensive Forschung betrieben wird und zwar von Wissenschaftlern aus den unterschiedlichsten Fachrichtungen, herrscht in diesem Bereich und insbesondere im Bereich der dynamischen Lastbalancierung eine geradezu babylonische Sprachverwirrung. Aus diesem Grund wird im

nächsten Unterabschnitt eine Begriffserklärung, basierend auf einem völlig neuen Klassifikationsschema für dynamische Lastbalancierungsstrategien, durchgeführt.

2.3.1 Klassifikationsschema für dynamische Lastbalancierungsstrategien

Im Laufe der letzten 20 Jahre gab es eine Vielzahl von Veröffentlichungen, die versuchten eine einheitliche Begriffswelt einzuführen [CaKu88][ErMS94][Scha92]. Bevor jetzt einige der bekanntesten Klassifikationen vorgestellt werden, sollen die Anforderungen aufgezeigt werden, die eine Klassifikation im allgemeinen und ein Klassifikationsschema für dynamische Lastverwaltungsansätze im besonderen erfüllen muß.
Eine Klassifikation muß unabhängig vom Klassifikationsgegenstand *einfach* sein, damit sie angewendet wird (siehe die Klassifikation von Flynn in Kapitel 1), und sie muß einen schnellen *Überblick* über den Sachverhalt geben, damit ihre Anwendung Sinn macht. Ein Klassifikationsschema für dynamische Lastbalancierungsverfahren muß zusätzlich noch Anworten auf die folgenden fünf Fragen geben:

1. Wer soll welche Informationen bekommen?
2. Wer entscheidet über die Notwendigkeit eines Lastausgleichs?
3. Zwischen welchen Knoten des Systems soll ein Lastausgleich stattfinden?
4. Welche Lasteinheit soll migriert werden?
5. Wie groß ist der Migrationsradius einer einzelnen Lastbalancierungsinstanz im Falle eines verteilten Ansatzes?

Im nachfolgenden werden drei bekannte und in der Literatur häufig referenzierte Klassifikationsschemata vorgestellt und daraufhin überprüft, ob sie die oben genannten Anforderungen an ein Klassifikationsschema erfüllen.
Das wohl bekannteste und am weitesten verbreitete Klassifikationsschema stammt von Casavant und Kuhl [CaKu88]. Dieses Klassifikationsschema (siehe Abbildung 9) verwendet für die Klassifikation von statischen und dynamischen Lastverwaltungsansätzen einen hierarchischen Ansatz. Allerdings ist dieser Ansatz nicht konsistenz, da die Autoren einige Klassifikationsmerkmale, die sie nicht eindeutig einer Hierarchieebene zuordnen konnten, in einer flachen Zusatzklassifikation zusammengefaßt haben. Da man sich in der Literatur im allgemeinen nur auf den hierarchischen Teil der Klassifikation bezieht [Gosc91], wird im weiteren nur dieser betrachtet.
In der obersten Ebene unterscheiden Casavant und Kuhl zwischen den *lokalen Verfahren* [Tane87], die die Ausführungszeiten und Prioritäten von Prozessen planen, die bereits einem Prozessor zugewiesen wurden, und den *globalen Verfahren*, die den Kontext des gesamten Systems verwenden, um Prozesse Prozessoren zuzuweisen. Die globalen Verfahren werden noch weiter unterteilt in *statische* [BeBo85] [BeBo87] und *dynamische Lastbalancierungsverfahren* [BeSS93] [He89] [KuWa90] [Scha92]. Auf die statischen Verfahren wird jedoch im folgenden nicht weiter eingegangen. Die dynamischen Verfahren, bei denen die Entscheidungen in Abhängigkeit vom aktuellen Lastzustand getroffen werden, unterscheiden die Autoren weiter in *zentrale* [BoKu90] und *dezentrale* [LiKe87] [MuZa95] Verfahren. Während bei den zentralen Verfahren die Entscheidungen von einer Instanz getroffen werden, wird die Entscheidungsfindung bei den dezentralen Verfahren auf mehrere Instanzen verteilt. Im Laufe der letzten Jahre wurde die Unterteilung der dynamischen Verfahren in zentrale und dezentrale Verfahren nun weiter unterteilt in zentrale, dezentrale und *hierarchische* Verfahren [AhGh91][AhGF94][BoNG88][FeRu90][TiWi84]. Die hierarchischen Verfahren unterscheiden sich von den dezentralen Verfahren dadurch, daß es zwischen den verschiedenen Instanzen eine streng strukturierte und aufeinander aufbauende Aufgabenverteilung gibt. Die dezentralen Verfahren werden

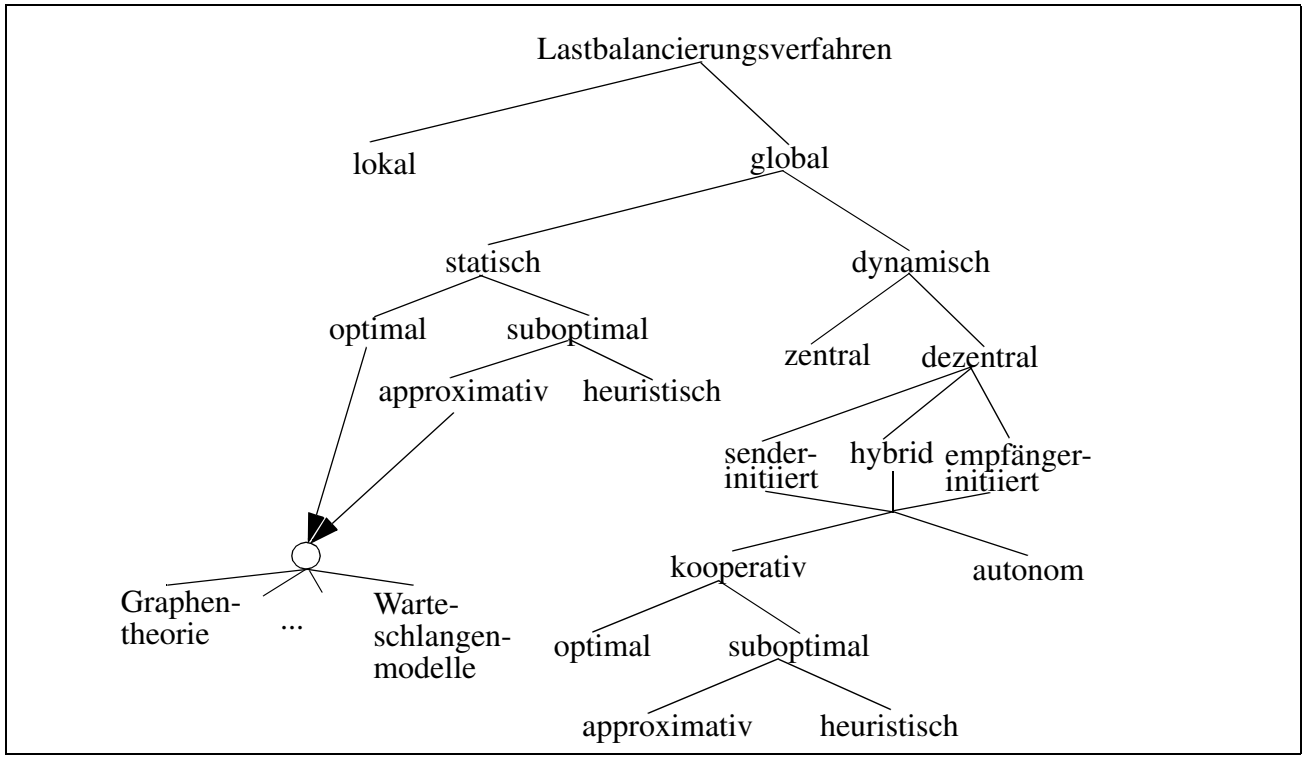


Abbildung 9: Klassifikationsschema nach Casavant und Kuhl.

noch weiter unterteilt im Hinblick darauf, wer die Entscheidung zum Lasttransfer trifft. Bei den *sender-initiierten* Verfahren [StSi84] wird der Lasttransfer vom überbelasteten Knoten initiiert, während bei den *empfänger-initiierten* Verfahren [NiXG85][NiXG85/2] ein unterbelasteter Knoten versucht, einen Lastausgleich herbeizuführen. Bei den *hybriden* Verfahren [BeDe94] wird in Abhängigkeit des Lastzustandes zwischen den beiden oben genannten Verfahren gewechselt, da Untersuchungen [EaLZ86] belegen, daß sender-initiierte Verfahren sich besonders für leicht belastete Systeme eignen, empfänger-initiierte Verfahren hingegen für stark belastete Systeme geeignet sind. Die letzte Unterscheidung, die in diesem Zusammenhang noch besprochen wird, ist die Unterscheidung in *kooperative* [FeYN88] und *autonome* Verfahren. Die Unterscheidungen zwischen optimalen und suboptimalen Verfahren usw. ist weitgehend selbsterklärend.

Während bei den autonomen Verfahren keine Lastinformationen zwischen den physischen Instanzen ausgetauscht wird und somit jede Instanz völlig unabhängig von den anderen Entscheidungsträgern agiert, beziehen die kooperativen Ansätze Lastinformationen der anderen Instanzen in ihren Entscheidungen mit ein.

Dieses Klassifikationsschema erfüllt ohne Zweifel die Kriterien bzgl. der Einfachheit und Klarheit, weshalb dieses Schema eine weite Verbreitung gefunden hat und auch in diesem Kapitel auszugsweise als Gliederungsgrundlage verwendet wurde. Es beantwortet hingegen nicht alle, speziell die für den Entwurf eines dynamischen Lastverwaltungssystems relevanten Fragen. So bleibt z.B. die Frage unbeantwortet, aufgrund welcher Information eine Instanz ihre Entscheidungen treffen muß. Auch die Frage des Migrationsraumes bleibt bei diesem Klassifikationsschema offen.

Baumgartner und Wah stellen in [BaWa88] ein Klassifikationsschema in tabellarischer Form vor. Es unterteilt die Verfahren anhand von 6 Kriterien. Insgesamt lassen sich mit diesem Schema 144 verschiedene Klassifikationsausprägungen erstellen, weshalb diese Klassifikation nicht unbedingt die Anforderungen bzgl. der Einfachheit und Klarheit erfüllt. Zudem beantwortet es ebenfalls nicht alle relevanten Fragen. So wird in dieser tabellarischen Klassifikation der Informationsaspekt nicht berücksichtigt, sowie die Frage des Migrationsradiuses unbeantwortet gelassen.

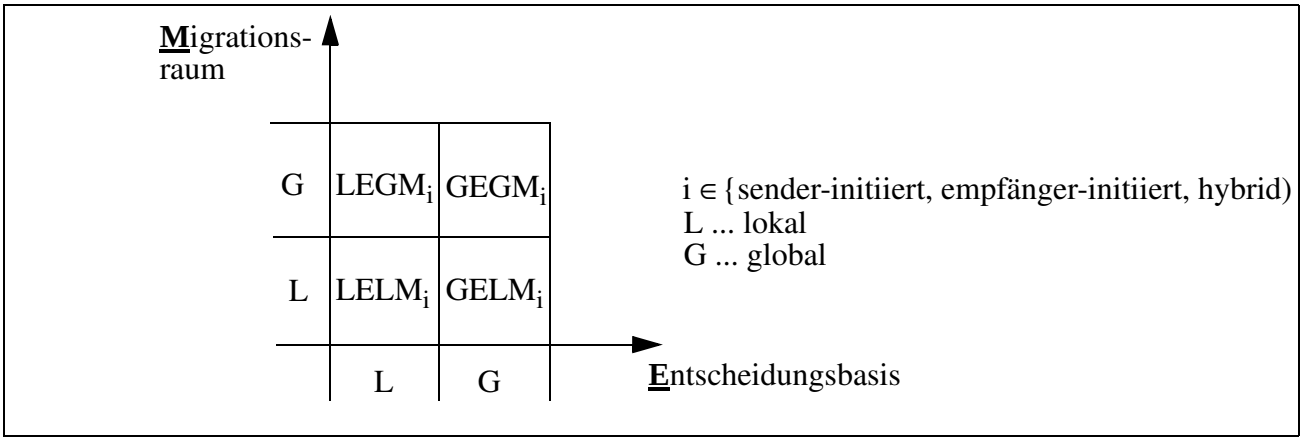


Abbildung 10: Klassifikationsschema von Lüling, Monien und Ramme

Ein sehr einfaches und übersichtliches Klassifikationsschema [LüMR91] stammt von Lüling et al.. Leider bezieht sich dieses Schema nur auf dezentrale Lastbalancierungsansätze. Im Vergleich zu anderen Klassifikationsversuchen verwendet dieser Ansatz drei Klassifikationskriterien (siehe Abbildung 10):

- Die Entscheidungsbasis mit den Ausprägungen lokal und global,
- den Migrationsraum mit den Ausprägungen lokal und global und
- die Initiierung des Lasttransfers mit den Ausprägungen sender-initiiert, empfänger-initiert und hybrid.

Alle drei angeführten Taxonomien haben ihre Stärken und Schwächen. Sie treffen jedoch implizit die Vereinfachung, daß es sich bei einer dynamischen Lastbalancierungsstrategie um eine atomare, in sich geschlossene Strategie handelt. Tatsächlich besteht eine Lastbalancierungsstrategie jedoch aus mehreren, genauer gesagt, fünf Teilstrategien.
Die erste Entscheidung, die beim Entwurf einer Lastbalancierungsstrategie getroffen werden muß, betrifft die Informationsbasis der Strategie:

- Die **Informationsbasis** [BaSh85][BeDe94] gibt an, aufgrund welcher Informationen eine dynamische Lastbalancierungsinstanz ihre Entscheidungen trifft. Im Falle einer lokalen Informationsbasis verwendet eine Lastbalancierungsinstanz nur Informationen ihres lokalen Knotens, im Falle einer bereichsbeschränkten Informationsbasis wird neben der lokalen Information noch die Information von einer begrenzten Anzahl von Nachbarknoten berücksichtigt. Im Falle einer globalen Informationsbasis werden Informationen des gesamten Systems verwendet.

In der Klassifikation von Casavant und Kuhl fallen die Lastbalancierungsstrategien mit einer lokalen Informationsbasis in die Klasse der autonomen Verfahren, die bereichsbeschränkten und die globalen Verfahren in die Klasse der kooperativen Lastbalancierungsansätze.
Das zweite Design-Merkmal beim Entwurf einer Lastbalancierungsstrategie betrifft die Transferentscheidung:

- Die **Transferentscheidung** [LiRa92] legt fest, wann ein Lasttransfer stattfinden soll und, vorallem, wer diese Entscheidung zu treffen hat. Die Transferentscheidung ist somit gleichbedeutend mit der Entscheidung, ob im System ein Lastungleichgewicht vorhanden ist, auf das das Lastbalancierungssystem reagieren muß.

Stellt ein Lastbalancierer aufgrund der Transferentscheidung fest, daß ein Lastungleichgewicht vorhanden ist, muß er festlegen, zwischen welchen Knoten des Systems ein Lasttransfer stattfinden soll. Diese Aufgabe wird als Lokationsentscheidung bezeichnet:

- Die **Lokationsentscheidung** [XuHw93] ermittelt, zwischen welchen Knoten ein Lastaustausch stattfinden soll.

Die beiden letzten Entscheidungen, die von Relevanz sind, betreffen die Auswahlsentscheidung und den Migrationsraum.

- Die **Auswahlentscheidung** legt fest, welche Lasteinheit zwischen den durch die Lokationsentscheidung ausgewählten Knoten ausgetauscht werden soll.
- Hingegen gibt der **Migrationsraum** an, ob eine Lastbalancierungsinstanz nur Last innerhalb eines bestimmten Bereiches verschieben darf oder innerhalb des gesamten Systems.

In Tabelle 3 ist das vollständige Klassifikationsschema, das im weiteren Verlauf des Abschnittes verwendet wird, dargestellt.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
lokal	zentral	zentral	zentral	bereichsbeschränkt
bereichsbeschränkt				
global	dezentral	dezentral	dezentral	global

Tabelle 3: Das für diese Arbeit verwendete Klassifikationsschema.

In den weiteren Unterabschnitten wird auf verschiedene Lastbalancierungsansätze, die in der Literatur beschrieben werden, eingegangen, wobei die Begriffsdefinitionen des Klassifikationsschemas von Casavant und Kuhl verwendet werden, da sich diese Begriffe inzwischen weitgehend etabliert haben. Das Kapitel schließt mit einer kurzen Beschreibung dreier kommerziell verfügbarer Lastverwaltungssysteme.

2.3.2 Zentrale Lastbalancierung

Bei einem strikt zentralen Lastbalancierer (siehe Tabelle 4) werden alle Entscheidungen von einer einzigen Lastbalancierungsinstanz getroffen. Eine strikt zentrale Entscheidungsfindung impliziert, daß alle entscheidungsrelevanten Informationen zum zentralen Lastbalancierer gesendet werden müssen und diese Instanz natürlich auch die Möglichkeit haben muß, auf alle Knoten des Systems Last zu transferieren. Trotz dem offensichtlichen Mangel an Skalierbarkeit hat ein strikt zentraler Ansatz zwei Vorteile, die ihn besonders im Hinblick auf kleinere Systemumgebungen attraktiv machen:

- Ein strikt zentraler Ansatz besitzt nur einen Kontrollfluß und ist somit relativ einfach in der Realisierung.
- Durch die Akkumulation aller entscheidungsrelevanter Information in einer Instanz verfügt ein zentraler Ansatz über den globalen Überblick. Dadurch ist ein strikt zentraler Lastbalancierer zumindest theoretisch in der Lage, optimale Entscheidungen zu treffen.

In der Praxis werden jedoch durch die mangelnde Skalierbarkeit häufig Variationen des strikt zentralen Lastbalancierungsansatzes erzwungen. Eine typische Variante stellt z.B. die Arbeit von Lin

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
global	zentral	zentral	zentral	global

Tabelle 4: Klassifikation eines strikt zentralen Lastbalancierungsansatzes.

und Raghavendra [LiRa92] dar. Bei diesem Ansatz werden die *Informationsverarbeitung* und die *Lokationsentscheidung* durch eine zentrale Komponente, den *Central Job Dispatcher,* realisiert. Dies gibt dem Lastbalancierer die Möglichkeit, bei seinen Entscheidungen Informationen über den globalen Systemzustand zu verwenden. Hingegen wird die *Transferentscheidung* und die *Auswahlentscheidung* aus Gründen der Leistungsfähigkeit verteilt realisiert.
Auf jedem Knoten des verteilten Systems läuft eine einfache Lastbalancierungskomponente. Jede Komponente verfügt über eine Liste von noch auszuführenden Aufträgen. Es wird dabei die Annahme getroffen, daß die Aufträge unabhängig voneinander sind und auf jedem Knoten des Systems generiert werden können. Dieser Ansatz geht somit von einer feingranularen Arbeitslast aus, wie sie z.B. in Client-/Serverarchitekturen verwendet wird [Beck95]. Die lokalen Komponenten informieren den Central Job Dispatcher über jede Änderung ihres Lastzustandes, d.h. immer wenn sich die Anzahl der noch zu bearbeitenden Aufträge ändert, wird eine Lastnachricht an die zentrale Komponente geschickt. Der Central Job Dispatcher aktualisiert mit Hilfe der Lastnachrichten seine globale Lasttabelle. In dieser Tabelle speichert er den Lastzustand (Anzahl der Aufträge) jedes Knotens des verteilten Systems. Wann immer ein Knoten des Systems leerläuft, schickt er einen *job request* an den Central Job Dispatcher. Dieser sucht daraufhin mit Hilfe der Lasttabelle den Knoten mit der größten Anzahl von noch nicht bearbeiteten Aufträgen und fordert diesen zum Lastausgleich auf.
Die Leistungsfähigkeit dieses zentralen Lastbalancierungsansatzes wurde mittels einer umfangreichen Simulationsstudie überprüft. In verschiedenen Simulationsläufen wurden die Auswirkungen unterschiedlicher Systemauslastungen und Auftragstransferzeiten auf das Lastbalancierungsergebnis ermittelt. Die Autoren kommen zu den Ergebnissen, daß

1. der zentrale Ansatz, vorausgesetzt daß die Auftragstransferzeit maximal 5% der durchschnittlichen Auftragsbearbeitungszeit beträgt, nahezu das optimale Ergebnis erreicht,
2. die Anzahl der benötigten Nachrichten zum Informationsaustausch vergleichbar zu bekannten, völlig verteilten Lastbalancierungsansätzen ist und
3. mit Hilfe dieses Verfahrens mehr als 1000 Knoten verwaltet werden können.

Allerdings wurde bei den Simulationsläufen die Annahme getroffen, daß die Kosten für den Informationsaustausch (bzgl. des Lastzustandes eines Knotens) vernachlässigt werden können, weshalb die Angaben über die Skalierbarkeit des Verfahrens nur eine theoretische Obergrenze darstellen können, da die Informationsverarbeitung in einer einzigen Instanz erfolgt. In Tabelle 5 ist der Lastbalancierungsansatz in das oben definierte Klassifikationsschema eingeordnet.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
global	dezentral	zentral	dezentral	global

Tabelle 5: Einordnung des Ansatzes von Lin und Raghavendra.

2.3.3 Dezentrale Lastbalancierung

Da die Skalierbarkeit bei einem zentralen Lastbalancierer der wesentliche Schwachpunkt ist, liegt das Vorgehen nahe, die Aufgaben eines Lastbalancierers auf mehrere Instanzen zu verteilen. In Tabelle 6 ist die Klassifikation eines vollständig dezentralen Lastbalancierungsansatzes angegeben. Die Leis-

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
lokal	dezentral	dezentral	dezentral	lokal

Tabelle 6: Einordnung eines völlig dezentralen Lastbalancierungsansatzes.

tungsfähigkeit eines solchen völlig dezentralen Ansatzes mit einer lokalen Informationsbasis ist jedoch nur gering, da es durch die mangelnde Kooperation zwischen den Lastbalancierungsinstanzen u.U. zu kontra-produktiven Lasttransferentscheidungen kommen kann. Aus diesem Grund weichen die in der Literatur vorgeschlagenen Ansätze in der Regel in bezug auf die Informationsbasis und teilweise noch in bezug auf den Migrationsraum vom völlig verteilten Ansatz ab.
Da eine globale Informationsbasis der Skalierbarkeit des Ansatzes widersprechen würde, verwenden die meisten dezentralen Ansätze eine bereichsbeschränkte Informationsbasis [LiKe87][Ludw93]. Hierbei stellt sich allerdings die Frage, wer welche Information bekommen soll. Eine sehr interessante Umsetzung der bereichsbeschränkten Informationsbasis stammt von Barak und Shiloh [BaSh85]. Dieses Verfahren wird z.B. auch im Lastverwalter für die Service-Partition des Parallelrechners Intel Paragon XP/S [TrZB94] (vgl. Kapitel 4) verwendet. Der Algorithmus geht davon aus, daß auf jedem der N Knoten eine Lastverwaltungsinstanz vorhanden ist, die in einem Lastvektor L der Länge m sowohl ihren eigenen Lastzustand (L[0]) als auch die Lastzustände von m-1 anderen Instanzen speichert. Der Lastinformationsaustausch erfolgt nun in vier Schritten:

1. In regelmäßigen Abständen aktualisiert jede Instanz des Lastbalancierers ihren lokalen Lastvektoreintrag L[0].
2. Anschließend wählt jede Instanz mit Hilfe eines Zufallsgenerators einen Knoten j ($1 \le j \le N$) als Empfänger für eine Lastnachricht aus.
3. Die Instanz verschickt die erste Hälfte ihres Lastvektors (bezeichnet mit Lr) zum Knoten j.
4. Jede Instanz, die einen Lastvektor Lr zugeschickt bekommen hat, berechnet basierend auf den folgenden Regeln ihren neuen Lastvektor L:

$$L_{neu}(2i) := L_{alt}(i) \qquad 1 \le i \le m/2-1$$
$$L_{neu}(2i+1) := Lr(i) \qquad 0 \le i \le m/2-1$$

Der Aktualisierungsalgorithmus in Schritt 4 stellt sicher, daß jeweils in der ersten Hälte des Lastvektors die aktuellste Information gehalten wird. Die prinzipielle Arbeitsweise des Verfahrens ist in Abbildung 11 dargestellt. Wie zu erkennen ist, garantiert das Verfahren, daß im neuen Lastvektor L_{neu} jeweils die aktuellsten Einträge der beiden Vektoren L_{alt} und Lr übernommen werden.
Ein gänzlich anderer Ansatz zur Realisierung einer bereichsbeschränkten Informationsbasis wird in den Arbeiten von Lin/Keller [LiKe87] und Kalé [Kalé88] verwendet. Während im obigen Verfahren die Lastwerte mit Hilfe eines Zufallsgenerators auf die Instanzen im System verteilt werden, werden in den beiden nachfolgenden Verfahren die Lastwerte nur innerhalb einer Gruppe von benachbarten Knoten ausgetauscht.
Das Gradientenverfahren von Lin/Keller [LiKe87] gehört dabei wohl zu den bekanntesten Vertretern der dezentralen Lastbalancierungsstrategien. Die Lastbalancierungsstrategie arbeitet in zwei Schrit-

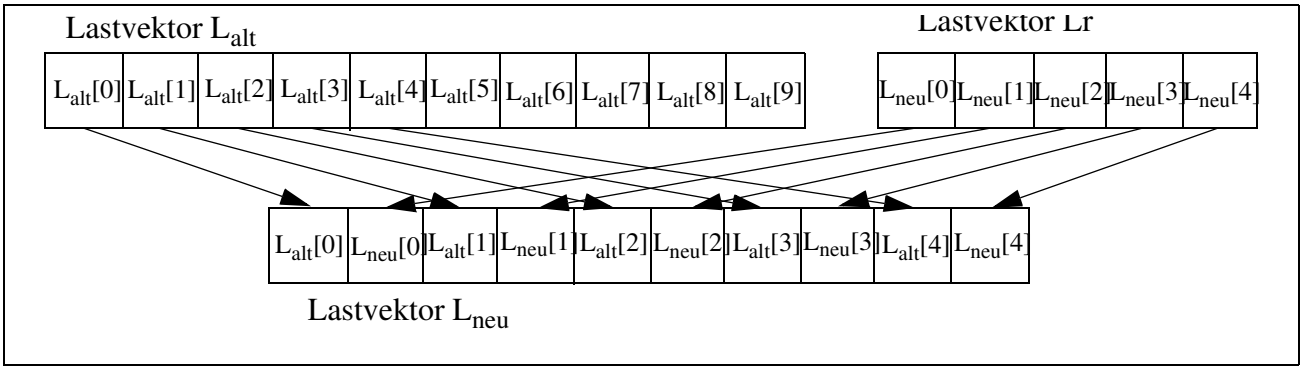


Abbildung 11: Arbeitsweise des Informationsaustauschverfahrens von Barak und Shiloh.

ten. In einem ersten Schritt berechnet jeder Knoten autonom seinen Lastzustand. Es werden dabei diedrei Lastzustände *unterbelastet*, *normal belastet* und *überbelastet* unterschieden. Ein Knoten im Lastzustand *unterbelastet* fordert zusätzliche Aufträge an. Ein Knoten im Lastzustand *überbelastet* hingegen versucht, Aufträge an unterbelastete Knoten abzugeben. In einem zweiten Schritt berechnet dann jeder Knoten seine Entfernung zum nächstgelegenen, unterbelasteten Knoten. Das Tupel aller Distanzwerte wird als Gradientenoberfläche bezeichnet. Mit Hilfe der Gradientenoberfläche kann jeder überbelastete Knoten autonom Aufträge zum nächstgelegenen unterbelasteten Knoten migrieren, indem er sie in Richtung des steilsten Gradienten transferiert. Dabei steht die Steigung des Gradienten in direkter Korrelation zum Lastunterschied zwischen den Knoten im System. Auftragsmigrationen erfolgen dabei immer nur zwischen benachbarten Knoten. Um für die Berechnung der Gradientenoberfläche keine zentrale Instanz bzw. globale Synchronisation zu benötigen, wird in [LiKe87] ein verteilter Algorithmus vorgestellt, mit dem es möglich ist, die Gradientenoberfläche sukzessive anzunähern. Dazu wird jedem Knoten ein Druckwert zugeordnet, den er allen seinen unmittelbaren Nachbarknoten mitteilt. Ein Knoten im Lastzustand *unterbelastet* hat dabei einen Druckwert von 0. Für alle anderen Knoten ergibt sich der Druckwert aus dem Minimum der um eins erhöhten Druckwerte der Nachbarknoten. Das Ergebnis der Berechnung wird als Druckoberfläche bezeichnet und ist in Abbildung 12 für ein 3x4 Rechteckgitter dargestellt. In dieser Beispielkonfiguration migriert Knoten 7, der als einziger Knoten überbelastet ist, Aufträge über Knoten 6 an Knoten 5, der sich im Lastzustand „unterbelastet“ befindet. In Tabelle 7 ist die Klassifikation des Gradientenverfahren dargestellt.

Das Gradientenverfahren weist im wesentlichen zwei Schwachstellen auf. Erstens wird nur dann von einem überbelasteten Knoten Last abgezogen, wenn es mindestens einen unterbelasteten Knoten im System gibt. Ein Lastausgleich zwischen überbelasteten und normal belasteten Knoten findet nicht statt, unabhängig davon, wie groß der Lastunterschied zwischen diesen Knoten ist. In [LüMR91] wird eine Variation des Gradientenverfahrens vorgestellt, die diesen Schwachpunkt des Gradientenverfahrens behebt.

Der zweite wesentliche Schwachpunkt des Gradientenverfahrens ist, daß das Gradientenmodell davon ausgeht, daß die einzelnen Aufträge unabhängig voneinander sind, d.h. daß es keine Kommunikations- bzw. Sychronisationsbeziehungen gibt. Das Arbeitslastmodell entspricht somit dem des Lastbalancierungsansatzes von Barak und Shilo. Wendet man dieses Verfahren nun auf Aufträge an, die Teile eines enggekoppelten, parallelen Programmes sind, können diese Aufträge beliebig auseinanderdriften, ein Sachverhalt, der sich negativ auf das Kommunikationsaufkommen auswirken kann. In [Heis94] wird eine Erweiterung des Gradientenverfahrens auf parallele Anwendungen vorgestellt, bei dem sowohl Kommunikationsbeziehungen als auch Migrationskosten berücksichtigt werden. Diese Größen werden als Kräfte realisiert, die den durch Lastunterschiede entstehenden Druckkräften entgegenwirken.

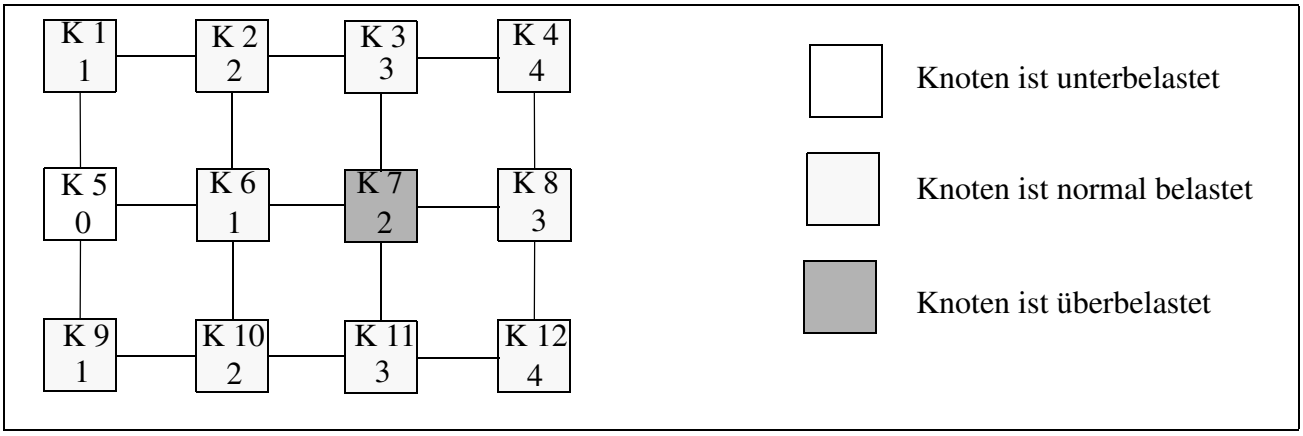


Abbildung 12: Druckoberfläche für ein 3 x 4 Rechteckgitter.

Der zweite dezentrale Lastbalancierungsansatz, der beim Austausch der Lastinformationen ebenfalls Informationen über die Nachbarschaftsbeziehungen der Knoten ausnutzt, stammt von Kalé [Kalé88]. In dieser Arbeit wird der dezentrale Lastbalancierungsansatz *Contracting Within Neighborhood* vorgestellt (vgl. Tabelle 8). Jeder Knoten des parallelen Systems verfügt bei diesem Ansatz über eine Last-

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
global	dezentral	dezentral	dezentral	bereichsbeschränkt

Tabelle 7: Einordnung der Gradientenmethode und ihrer Variationen.

tabelle mit den Lastwerten seiner Nachbarknoten. Zur Konsistenzhaltung der Lasttabellen teilt ein Knoten jede Laständerung seinen Nachbarknoten mit. Wird auf einem Knoten ein neuer Auftrag generiert, wird dieser mit einem Migrationszähler versehen und unabhängig vom lokalen Lastzustand dem Nachbarknoten mit der geringsten Last zugeteilt. Bei jeder Migration eines Auftrages wird dessen Migrationszähler entsprechend erhöht. Erst wenn ein Auftrag eine Mindestanzahl von Migrationen absolviert hat, darf er von einem unterbelasteten Knoten bearbeitet werden. Hat ein Auftrag hingegen die maximale Anzahl von Migrationen absolviert, muß er auf jeden Fall bearbeitet werden. Durch die minimale und maximale Anzahl der Migrationen pro Auftrag ist sichergestellt, daß sich die Last einerseits schnell im System verteilt und andererseits ein Auftrag in endlicher Zeit von einem Knoten bearbeitet wird. In [Kalé88] wird auf die offensichtliche Tatsache hingewiesen, daß es durch die minimale Anzahl von Migrationen zu unnötig vielen Migrationen kommen kann bzw. durch die maximale Anzahl von Migrationen der Fall auftreten kann, daß das Verfahren nicht in der Lage ist, ein Lastungleichgewicht auszugleichen.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
bereichsbeschränkt	dezentral	dezentral	dezentral	bereichsbeschränkt

Tabelle 8: Einordnung des Ansatzes Contracting Within Neighborhood.

2.3.4 Hierarchische Lastbalancierung

Wie in den beiden vorigen Unterabschnitten aufgezeigt wurde, besteht die Stärke eines zentralen Lastbalancierungsansatzes in der Verarbeitung von komplexer Information, während ein dezentraler Ansatz eine wesentlich bessere Skalierbarkeit aufweist. Hierarchische Lastbalancierungsansätze vereinigen nun die beiden genannten Stärken in nahezu perfekter Weise. Hierarchische Ansätze sind gut skalierbar aufgrund der hohen Parallelität in den tieferen Ebenen der Hierarchie, sie sind jedoch zumindest in den höheren Ebenen der Hierarchie auch in der Lage, komplexe Informationen (siehe dazu Kapitel 3) zu verarbeiten. Nachfolgend werden zwei bekannte, hierarchische Ansätze vorgestellt. Auf eine Klassifikation eines allgemeinen, hierarchischen Ansatzes wird verzichtet, da es bei den hierarchischen Ansätzen sehr vielfältige Variationsmöglichkeiten gibt.
Gupta und Gopinath stellen in [GuGo90] [GuGo91] ein zweistufiges Lastbalancierungsverfahren vor (vgl. Tabelle 9). Dabei wird das verteilte System unter Berücksichtigung der Kommunikationsverbindungen des Systems in Partitionen aufgeteilt. Beim Auftreten von Lastungleichgewichten wird in einem ersten Schritt versucht, die Last innerhalb einer Partition gleichmäßig zu verteilen. Ist dies nicht möglich, wird in einem zweiten Schritt ein Lasttransfer über Partitionsgrenzen hinweg initiiert. Als Lasttransfermechanismus findet bei diesem Ansatz die preemptive Prozeßmigration Anwendung [KrLi88]. Zur Lastverwaltung innerhalb einer Partition bzw. zur Lastverwaltung über Partitionsgrenzen hinweg werden zwei unterschiedliche Lastbalancierungsstrategien verwendet. Für die Lastbalancierung innerhalb einer Partition wird eine Variante des Drafting-Algorithmus [NiXG85] verwendet. Dazu verfügt jeder Knoten über eine Lasttabelle, die die Lastwerte aller Knoten seiner Partition enthält. Mit Hilfe der Lasttabelle kann ein unterbelasteter Knoten an alle überbelasteten Knoten seiner Partition einen *draft-request* schicken. Für den Lasttransfer über Partitionsgrenzen hinweg wird eine Variante des Bidding-Algorithmus [StSi84] verwendet. Stellt ein überbelasteter Knoten fest, daß es innerhalb seiner Partition keine unterbelasteten Knoten gibt, fordert er in jeder anderen Partition einen Knoten zum Lastausgleich auf. Da ein Knoten nicht über Lastinformation von anderen Partitionen verfügt, werden zur Auswahl der potentiellen Lasttransferpartner nur statische Informationen (z.B. Entfernung im Kommunikationsnetz, etc.) herangezogen. Befindet sich ein statischer Lasttransferpartner nicht im Lastzustand *unterbelastet*, leitet er die Nachricht, falls möglich, an einen unterbelasteten Knoten innerhalb seiner Partition weiter.
Der Vorteil dieses Verfahrens ist seine Skalierbarkeit. Insbesondere hängt der Verwaltungsaufwand in erster Linie von der Größe und nicht von der Anzahl der Partitionen ab. Allerdings ist der Verwaltungsaufwand des Drafting-Algorithmus zum Lasttransfer innerhalb einer Partition erheblich, da ein unterbelasteter Knoten an mehrere überbelastete Knoten Lasttransferanfragen schickt, aber es immer nur mit einem überbelasteten Knoten zu einem Lasttransfer kommt. Zusätzlich kann die Konsistenzhaltung der Lasttabellen eine nicht unerhebliche Last verursachen - ändert sich der Lastzustand eines Knotens, bedeutet dies, daß er an alle Knoten innerhalb seiner Partition eine Lastnachricht schicken muß.
Weiterhin macht dieses Verfahren keinen Gebrauch von den Vorteilen, die ein hierarchischer Ansatz in bezug auf die Aggregation und Verwertung von Information in höheren Hierarchieebenen bietet.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
bereichsbeschränkt	dezentral	dezentral	dezentral	global

Tabelle 9: Einordnung des Ansatzes von Gupta und Gopinath.

In [AhGF94] wird ein ebenfalls zweistufiger, hierarchischer Lastbalancierungsansatz für große Hypercube-Parallelrechner vorgestellt. Das Verfahren läßt sich jedoch auch auf Parallelrechner mit anderen Kommunikationstopologien verallgemeinern. Das von diesem Lastverwaltungsverfahren unterstützte Programmier- und Ablaufmodell besteht aus einer Menge von Aufträgen, die asynchron ausgeführt werden können. Jeder Auftrag generiert im Anschluß an seine Berechnung entweder neue Aufträge (Kinder) oder sendet sein Berechnungsergebnis an seinen Elternauftrag. Die Beziehungen unter den Aufträgen lassen sich somit als Task Präzedenz Graph (TPG) darstellen. Die Knoten des Graphes repräsentieren die Aufträge, die Kanten die Eltern-Kinder-Beziehungen zwischen den Aufträgen. Zu Beginn und Ende jeder Berechnung gibt es genau einen Auftrag, die Wurzel des TPG. Im Vergleich zu vielen anderen TPG-basierten Lastverwaltungsverfahren geht dieser hierarchische Lastbalancierungsansatz davon aus, daß der TPG a priori nicht bekannt ist und somit weder die Anzahl der Aufträge noch der maximale Parallelitätsgrad bekannt ist.
Im Hinblick auf eine hierarchische Lastbalancierung wird der Parallelrechner in sogenannte *Spheres* partitioniert. In jeder Sphere gibt es einen ausgezeichneten Knoten, den *Median*. Der Median verfügt über eine Liste mit den Lastwerten aller Knoten, die zu seiner Sphere gehören. Dem Median übergeordnet gibt es noch einen zentralen Lastbalancierer, der z.B. auf einem Vorrechner plaziert werden kann. Vergleichbar zu einem Median verfügt auch der zentrale Lastbalancierer über eine Tabelle mit den Lastwerten aller Median-Knoten. Wird eine parallele Anwendung dem System übergeben, wird dieser Auftrag vom zentralen Lastbalancierer an einen Median mit minimaler Auslastung weitergeleitet. Dieser ist dann für die gleichmäßige Verteilung der anstehenden Aufträge innerhalb seiner Sphere verantwortlich, wobei als Lasttransfermechanismus nur die initiale Auftragsplazierung zur Verfügung steht.
Im Rahmen einer Simulationsstudie wurde der hierarchische Lastbalancierungsansatz mit einer Variante des in [Kalé88] beschriebenen dezentralen Verfahrens verglichen. In allen untersuchten Konfigurationen war der hierarchische Lastbalancierer dem dezentralen Lastbalancierer überlegen. Allerdings weist der hierarchische Ansatz zwei entscheidende Nachteile auf:

1. Alle Aufträge einer parallelen Anwendung werden innerhalb einer Sphere abgearbeitet, d.h. es findet kein Lasttransfer zwischen den Spheres statt.
2. Das unterstützte Programmier- und Ablaufmodell entspricht nicht dem auf heutigen Parallelrechnern vorherrschenden Programmier- und Ablaufmodell von grobgranularen Prozessen und erfordert somit in vielen Fällen eine Restrukturierung existierender paralleler Anwendungen.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
bereichsbeschränkt[a]	dezentral	dezentral	dezentral	bereichsbeschränkt

Tabelle 10: Einordnung des Ansatzes von [AhGF94].

a. Der zentrale Lastbalancierer auf dem Vorrechner verfügt über einen globalen Systemüberblick. Da die eigentliche Lastbalancierung jedoch innerhalb der Spheres stattfindet, wird die Informationsbasis als bereichsbeschränkt bezeichnet.

2.4 Kommerzielle Lastverwaltungssysteme

In diesem Abschnitt werden drei kommerzielle Lastverwaltungssysteme, der Task Broker von HP, der LoadLeveler von IBM sowie das Codine System von Genias Software, vorgestellt. Da sowohl der LoadLeveler von IBM als auch das Codine System von Genias Software auf dem Condor-System basieren, werden diese beiden kommerziellen Systeme in einem Abschnitt beschrieben.

2.4.1 HP Task Broker

Der HP Task Broker [GALSTW93] ist ein Lastverwaltungswerkzeug für heterogene, Unix-basierte Systeme [HaAL95][MiTS89][SaJP94]. Die Lastverwaltung ist in einer anwendungstransparenten Art und Weise realisiert, so daß keine Modifikationen der Anwendungsprogramme erforderlich sind. Als einziger Lasttransfermechanismus wird die initiale Prozeßplazierung verwendet.
Beim Task Broker handelt es sich um einen dezentralen Lastbalancierungsansatz, da auf jedem Knoten des Systems ein *Daemon-Prozeß* läuft, der als Auftraggeber (Client) oder als Auftragbearbeiter (Server) fungiert. Der Task Broker basiert dabei im wesentlichen auf zwei Protokollen zum Ausführen eines Programms (Auftrag) auf einem Serverknoten und der Auswahl eines geeigneten Serverknotens. Beide Protokolle sind nachfolgend kurz beschrieben.
Das Protokoll für die Ausführung eines Programms auf einem nicht lokalen Rechnersystem arbeitet wie folgt:

1. Ermittlung der gewünschten Maschinenkonfiguration vom Endbenutzer (Graphikbeschleuniger, etc.)
2. Auswahl des besten Rechenservers für den gegeben Berechnungsauftrag
3. Verbindungsaufbau zwischen dem lokalen und dem entfernten Rechensystem via *telnet*, *remsh*, *crp*
4. Übertragung des Programmes- und der Daten via *ftp* oder *NFS*
5. Ausführung des Programms auf dem entfernten Rechensystem
6. Rücktransfer der Ergebnisdatei auf das lokale Rechensystems via *ftp* oder *NFS*

Das Protokoll zur Auswahl des besten Rechenservers arbeitet nach dem Bidding-Verfahren (vgl. Abschnitt 2.3.4) und besteht aus 7 Teilschritten.

1. Der Benutzer übergibt den Berechnungsauftrag an den lokalen Task Broker (Client -Daemon).
2. Der Client-Daemon sendet daraufhin eine Nachricht an alle Berechnungsserver einer Gruppe mit der Bitte, ein Angebot (*bid)* für den Berechnungsauftrag abzugeben. Die Gruppenzugehörigkeit eines Client-Daemons ist dabei eine konfigurierbare Eigenschaft des Systems.
3. Die Server berechnen ihr Angebot, den sogenannten Affinitätswert, basierend auf ihrer Verfügbarkeit für den Berechnungsauftrag, und schicken ihr Angebot an den Client-Daemon zurück.
4. Der Client-Daemon wartet eine bestimmte Zeit auf das Eintreffen der angeforderten *Angebote* und wählt dann den Server mit dem besten *Angebot* aus.
5. Übertragung der Programm- und Datendateien (falls nötig)
6. Programmausführung auf dem entfernten Rechnersystem
7. Rückübertragung der Ergebnisdatei, die im Arbeitsverzeichnis des Benutzers abgelegt wird

Kann der Client-Daemon keinen geeigneten Berechnungsserver ermitteln, wird der Berechnungsauftrag entweder zwischengespeichert oder lokal berechnet. Der HP Task Broker ist ein einfacher Ansatz zur Lastverwaltung in heterogenen Workstation-Umgebungen, der für viele Anwendungsfälle ausreicht. Da sich das System jedoch vollständig auf die initiale Prozeßmigration beschränkt, ist es für langlaufende Anwendungen nicht geeignet, insbesondere wenn nicht alle Anwendungen der Kontrolle des Task Brokers unterliegen. Diese Limitation wurde in den nachfolgenden, kommerziellen Systemen aufgehoben, da diese zumindest prinzipiell in der Lage sind, eine Migration eines bereits gestarteten Prozesses durchzuführen.

2.4.2 Condor-basierte Systeme

Condor [LiLM88] ist eine Lastverwaltungsumgebung für homogene Workstationnetze und arbeitet nach dem Load-Sharing-Prinzip [Sven90][ZZWD93]. Beim Load-Sharing wird im Vergleich zur dynamischen Lastbalancierung nicht versucht, die Arbeitslast möglichst gleichmäßig auf die Systemressourcen zu verteilten, sondern es wird lediglich zu vermeiden versucht, daß Systemressourcen völlig unbelastet sind. Als Bearbeitungsaufträge betrachtet das System langlaufende Hintergrundprozesse.

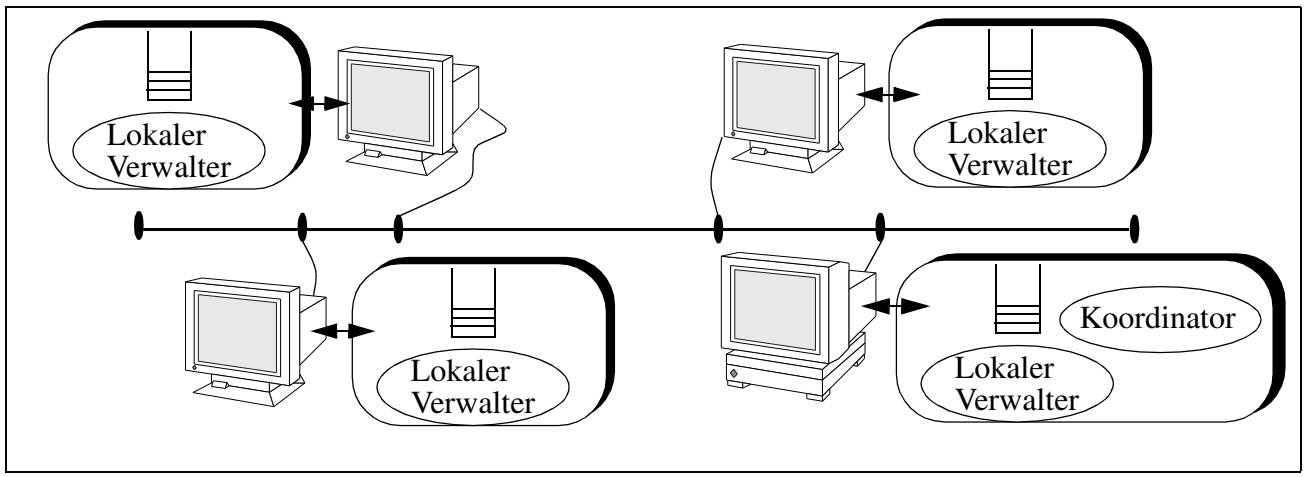


Abbildung 13: Komponentenstruktur des Condor-Systems.

Das Condor-System ist folgendermaßen aufgebaut (siehe Abbildung 13). Auf jedem Knoten des Systems gibt es einen lokalen Verwalter und eine Auftragswarteschlange. Auf einem Knoten des Systems läuft zusätzlich noch der zentrale Koordinator. Der zentrale Koordinator fragt den Lastzustand der verschiedenen Knoten im System in regelmäßigen Abständen (2 Minuten) ab und trifft, basierend auf dieser Information, seine Lastverwaltungsentscheidungen. Die lokalen Verwalter sind nur für ihren Knoten verantwortlich. Unter anderem überprüfen die lokalen Verwalter in regelmäßigen Abständen (30 Sekunden), ob ihr Knoten wieder lokal benutzt wird. Sobald ein Knoten wieder lokal benutzt wird, suspendiert der lokale Verwalter alle auf seinem Knoten laufenden Hintergrundprozesse und erwartet vom zentralen Koordinator die Adresse einer freien Workstation, um die Hintergrundprozesse dorthin auszulagern. Zur Migration der Prozesse wird ein Checkpointing-Verfahren verwendet.
Eines der Ziele des Condor-Systems war, die Lastverwaltung anwendungstransparent zu realisieren. Dazu wird für jeden Bearbeitungsauftrag, der nicht lokal ausgeführt wird, auf der lokalen Workstation ein *Shadow*-Prozeß initiiert, der für die Abarbeitung aller Unix-Systemaufrufe zuständig ist.
Aufgrund seiner Eigenschaften wurde das Condor-System als Ausgangsbasis für mehrere kommerzielle Lastverwaltungswerkzeuge verwendet. Sowohl der IBM LoadLeveler [IBM95] als auch das CODINE-System von Genias Software [PaBH95] basieren auf dem Condor-System. Da die beiden kommerziellen Systeme sich in vielerlei Hinsicht ähnlich sind, wird nachfolgend nicht weiter zwischen diesen Systemen differenziert.
Sie unterscheiden sich jedoch in mehreren Punkten zum Teil erheblich von ihrem Ausgangssystem Condor. Im Gegensatz zu Condor unterstützten diese Systeme:

- Ressourcendefinitionsdateien
- Heterogene Hardware-Umgebungen
- Parallele Anwendungen
- Interaktive Anwendungen

Basierend auf der Beschreibung des Ressourcenbedarfs kann der Lastverwalter eine effizientere Zuweisung der Prozesse auf die verfügbaren Ressourcen durchführen und somit Lastungleichgewichte schon im Vorfeld vermeiden.
Die Unterstützung von heterogenen Hardware-Umgebungen bedeutet jedoch auch, daß die Systeme als Lastausgleichsmechanismus in erster Linie mit der initialen Prozeßzuweisung arbeiten. Die Checkpoint- und Migrationsmechanismen, über die auch die kommerziellen Systeme verfügen, werden hauptsächlich für Wartungszwecke bzw. für Zwischensicherungen bei langlaufenden Anwendungen verwendet. Dies bedeutet jedoch keine wesentliche Einschränkung gegenüber dem Ausgangssystem Condor. In [PaBH95] wurde z.B. untersucht, wie lange eine Migration eines 30 MByte großen Prozesses, basierend auf dem Checkpointing-Verfahren des Condor-Systems Version 3.3, dauert. Auf einer Testumgebung, die aus über Ethernet verbundenen Workstations bestand, ergab sich eine Transferzeit von ungefähr 80 Sekunden. Dabei entfallen allein 48 Sekunden auf die Übertragung der 30 MByte über das Kommunikationsmedium Ethernet (jeweils 24 Sekunden für die Übertragung der Daten von der Workstation zum Datenserver bzw. vom Datenserver zur Workstation),
Deshalb kann der originale Lasttransferansatz des Condor-Systems für den Zweck der dynamischen Lastbalancierung als nicht geeignet bezeichnet werden.

Kapitel 3

Das Konzept der Informationsebenen

Im Laufe der letzten zwei Jahrzehnte wurde eine Vielzahl von dynamischen Lastbalancierungsansätzen [GaGJ78][DoEG96] entwickelt. Dabei wurde das Problem der dynamischen Lastbalancierung aus den unterschiedlichsten Blickwinkeln betrachtet. So wurde untersucht, welche besonderen Eigenschaften einen betriebssystemintegrierten, dynamischen Lastbalancierer [BaWh88][DoOu91] auszeichnen, und welche besonderen Vorteile ein anwendungsintegrierter, dynamischer Lastbalancierer [ChYL95][SaEr87][SeGS95] bietet. Auch wurden verschiedenste Lastbalancierungsansätze im Hinblick auf ihre Eignung für unterschiedliche Hardware-Plattformen [AnHC90][BlPa94][Bokh93] [Bond93][DRCR92][EHMS95] bzw. auf ihre Anwendungscharakteristiken [HwCh87][LiCL90] hin untersucht. Insbesondere bei den Arbeiten im Zusammenhang mit verteilten Lastbalancierungsansätzen wurden viele Untersuchungen durchgeführt, die sich mit dem Informationsfluß zwischen Lastbalancierungsinstanzen beschäftigen [BaSh85][BaWa88][Cybe89][HuSu93][Wu95]. In der in Abschnitt 2.3.1 eingeführten Taxonomie wird der Informationsfluß zwischen den Lastbalancierungsinstanzen innerhalb des Kriterium *Informationsbasis* behandelt.
In engem Zusammenhang mit dem Informationsfluß zwischen den Lastbalancierungsinstanzen stehen auch die Arbeiten, die sich mit der Frage auseinandersetzen, wie bei einem völlig verteilten Lastbalancierer verhindert werden kann, daß einzelne Knoten durch unkontrollierten Lastausgleich über- bzw. unterbelastet werden [LuLa95].
Obwohl sich viele Arbeiten mit dem Informationsfluß innerhalb eines verteilten Lastbalancierers beschäftigt haben, wurde die einem dynamischen Lastbalancierer zur Verfügung stehende Information bislang eher undifferenziert verwendet. Während die Simulationsstudien in der Regel zur Bestimmung der Lastzustände nur sehr einfache Informationen verwenden (z.B. die Anzahl der zur Verarbeitung anstehenden Aufträge), wird in realen Lastbalancierungsumgebungen die zur Entscheidungsfindung verwendete Information häufig durch das zugrundeliegende Betriebssystem vorgegeben. Auf andere Informationsquellen als die des Betriebssystems greifen nur die wenigsten Ansätze zurück, obwohl von den Anwendern bzw. den Anwendungen durchaus sehr wertvolle Informationen bereitgestellt werden können.
Bevor nun detailliert auf die Definition der Informationsebenen und deren Bedeutung für die dynamische Lastbalancierung eingegangen wird, wird kurz die Fage diskutiert welche Informationenfür einen dynamischen Lastverwalter relevant sind. Relevant sind prinzipiell sämtliche Informationen,

die sich auf eine knappe und somit kritische Ressource beziehen. Welche Ressourcen als kritisch einzustufen sind, hängt grundsätzlich vom Zielsystem (Hardware, Betriebssystem) sowie von der jeweiligen Anwendung ab. Typische Informationen bzgl. der Ressourcenauslastung, die von heutigen Betriebssystemen [OSF92] zur Verfügung gestellt werden, sind die aktuelle Hauptspeicher-, Prozessor- oder Bussauslastung.
Basierend auf dem Tatbestand, daß in der Vergangenheit die einem dynamischen Lastbalancierer zur Verfügung stehende Information undifferenziert betrachtete wurde, beschäftigt sich diese Arbeit mit der Frage, welche Informationen einem dynamischen Lastbalancierer prinzipiell zur Verfügung stehen und wie sich deren Verwendung auf die Effizienz eines dynamischen Lastbalancierers auswirkt. Zur Beantwortung dieser Frage wird in einem ersten Schritt eine Klassifikation der zur Verfügung stehenden Information durchgeführt. Das Ergebnis sind die im nachfolgenden definierten vier Informationsebenen.

3.1 Definition der Informationsebenen

Bei der Klassifikation der verfügbaren Information wird darauf geachtet, daß es erstens in einfacher Weise möglich ist, die Informationsverwendung bestehender Lastbalancierungssysteme einzuordnen, und zweitens, daß die identifizierten Informationsebenen möglichst strikt gegeneinander abgegrenzt sind. Ausgehend von diesen Rahmenbedingungen werden die vier ermittelten Informationsebenen wie folgt definiert.

Definition 2: Die **Informationsebene 0** umfaßt alle vom Betriebssystem durch Messung, Zählung, Beobachtung, etc. zur Laufzeit erfaßten Lastdaten.

Mit Hilfe der Informationsebene 0 ist es möglich, einen rein reaktiven Lastbalancierer zu realisieren, d.h. einen Lastbalancierer, der das System beobachtet und beim Auftreten von Lastungleichgewichten seine Lasttransferentscheidungen trifft. Da ein Lastbalancierer dieser Stufe keinerlei Informationen über das prinzipielle bzw. zukünftige Lastverhalten einer Anwendung erhält bzw. ableiten kann, ist er gezwungen, aus dem bisherigen Lastverhalten einer Anwendung auf deren zukünftiges Lastverhalten zu schließen. Ein Lastbalancierer der Ebene 0 setzt somit zumindest implizit ein konstantes Systemverhalten voraus (Lokalitätsprinzip). Durchläuft eine parallele Anwendung im Laufe der Berechnung jedoch Phasen mit unterschiedlichem Lastverhalten, stößt dieses Vorgehen an seine Grenzen. Trotzdem werden in vielen Bereichen der dynamischen Ressourcenverwaltung für die Entscheidungsfindung ausschließlich Informationen der Ebene 0 verwendet, da die meisten Anwendungen über hinreichend lang andauernde Bearbeitungsphasen verfügen und somit die erzielbaren Ergebnisse für praktische Zwecke ausreichen. Als Beispiele für dynamische Ressourcenverwaltungsprobleme, bei denen mit Hilfe von einfachen Informationen der Ebene 0 bereits sehr gute Ergebnisse erzielbar sind, sind die virutelle Hauptspeicherverwaltung [Zoma95] sowie die Verwaltung von parallelen Sekundärspeichermedien [ScWZ94] zu nennen. In Abschnitt 3.2 werden einige Informationen der Ebene 0 exemplarisch vorgestellt.

Definition 3: Die **Informationsebene 1** umfaßt alle statisch bzw. statistisch verfügbaren Informationen über die Anwendungs- und Betriebssystemcharakteristik.

Aufbauend auf die Ebene 0 liefert die Informationsebene 1 Informationen über das prinzipielle Lastverhalten einer Anwendung. Unter statische Informationen fallen dabei sämtliche invarianten Informationen über eine Anwendung, die sich z.B. aufgrund des verwendeten Verarbeitungsalgorithmus ergeben. Dazu gehören, falls die Anwendung ein fest vorgegebenes Kommunikationsverhalten aufweist, die Informationsbeziehungen der Anwendungsprozesse untereinander. Demhingegen müs-

sen statistische Informationen über das Lastverhalten einer Anwendung mittels einer Auswertung von einer ausreichend großen Anzahl von Anwendungsläufen ermittelt werden. Somit kann ein dynamischer Lastbalancierer unter Verwendung der statischen und vor allem unter Verwendung der statistischen Informationen dieser Ebene (vgl. Abschnitt 3.3) wesentlich bessere Abschätzungen über das zukünftige Lastverhalten einer Anwendung machen. Selbstverständlich hängt die Qualität der Lastentwicklungsprognosen entscheidend von der Qualität der Informationen der Ebene 1 ab. Der wesentliche Vorteil der Informationsebene 1 besteht darin, daß diese Informationen bereits vor dem Start der Anwendung verfügbar sind, d.h. zu einem Zeitpunkt, zu dem der Lastbalancierer der Ebene 0 noch keinerlei Informationen über die Anwendung und deren Lastverhalten besitzt.
Der große Vorteil der Informationsebene 1, daß deren Informationen bereits vor dem Start der Anwendung für den dynamischen Lastverwalter verfügbar sind, bedeutet natürlich gleichzeitig, daß diese Informationen mit einer zum Teil erheblichen Unsicherheit behaftet sind. Aufgrund dieser Unsicherheit ist die Informationsebene 1, speziell bei Anwendungen mit einem sehr dynamischen Lastverhalten (Ressourcenbedarf verändert sich sowohl während eines Anwendungslaufes als auch zwischen verschiedenen Läufen), nicht als alleinige Entscheidungsgrundlage geeignet, sondern immer als Ergänzung der Ebene 0 zu sehen.
Während die Ebenen 0 und 1 sich zur Gewinnung der entscheidungsrelevanten Informationen weitgehend auf die Beobachtung der Anwendung(en) beschränkten, beziehen die nachfolgenden beiden Ebenen Informationen direkt aus der Anwendung. Dadurch sind Sie in der Lage, sowohl Informationen über das invariante (datenunabhängige) als auch über das variante (datenabhängige) Anwendungsverhalten zu erhalten.

Definition 4: Die **Informationsebene 2** umfaßt alle Angaben, die zur Laufzeit von der Anwendung über ihren aktuellen Bearbeitungszustand verfügbar gemacht werden.

Die Informationsebene 2 ermöglicht es einem dynamischen Lastbalancierer, seine durch Systembeobachtung gewonnenen, prozeßbezogenen Lasteinschätzungen der Ebene 0 zu überprüfen und gegebenenfalls anhand der Anwendungsinformationen zu korregieren. Mit anderen Worten, die Informationsebene 2 erhöht im wesentlichen die Zuverlässigkeit der aus der Informationsebene 0 verfügbaren Lastwerte.

Definition 5: Die **Informationsebene 3** umfaßt alle Angaben, die von der Anwendung über ihr voraussichtliches zukünftiges Lastverhalten zur Laufzeit verfügbar gemacht werden.

Im Unterschied zu den niedrigeren Ebenen befähigt die Informationsebene 3 einen dynamischen Lastbalancierer zu einem vorausschauenden Handeln, wobei im Gegensatz zur Informationsebene 1 dieses Handeln nicht auf statischen bzw. statistischen Informationen beruht, sondern das tatsächliche Lastverhalten berücksichtigt. Basierend auf dieser Information ist ein Lastbalancierer in der Lage, vor dem Eintreffen eines Lastungleichgewichts seine Lasttransferentscheidungen zu treffen, ohne daß das Eintreffen des Lastungleichgewichts statisch bzw. statistisch vorhersehbar sein muß.
Das auf die Informationsebenen aufbauende Konzept der Lastbalancierungsebene ist in Abbildung 14 dargestellt. Basierend auf dem Laufzeitsystem, das noch als Bestandteil des Betriebssystems angesehen werden kann, stellt die Lastbalancierungsebene 0 die Grundfunktionalität eines dynamischen Lastbalancierers zur Verfügung. Aufgabe der Lastbalancierungsebenen 1 bis 3 ist es nun, die Grundfunktionalitat der Ebene 0 weiter auszubauen bzw. anzureichern. So versetzt die Ebene 1 einen dyna-

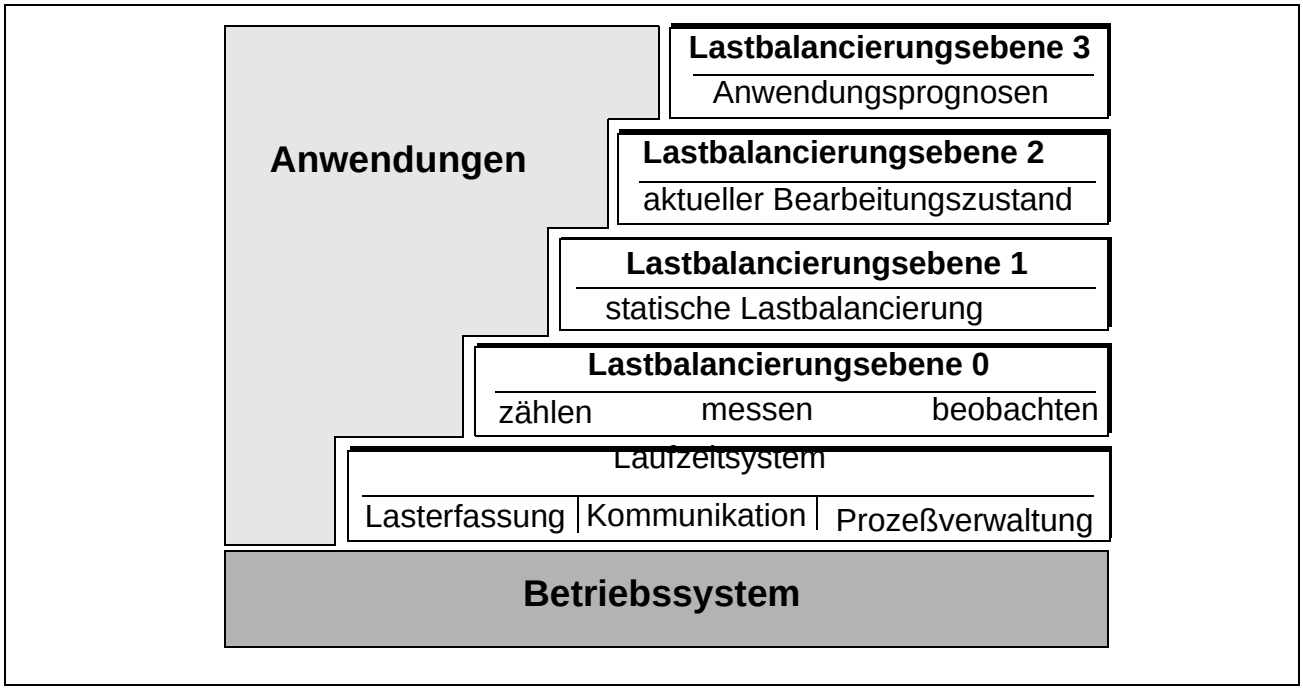


Abbildung 14: Das Konzept der Lastbalancierungsebenen.

mischen Lastbalancierer in die Lage, bessere initiale Plazierungsentscheidungen zu treffen. Die Ebene 2 ermöglicht es einem Lastbalancierer, seine durch Messung, Beobachtung etc. gewonnenen Lastdaten zu validieren, und die Ebene 3 schließlich befähigt einen dynamischen Lastbalancierer zu einem vorausschauenden Handeln. Es muß jedoch darauf hingewiesen werden, daß es sich hierbei um eine rein konzeptionelle Sichtweise handelt. Auf die tatsächliche Umsetzung der Informationsebenen im PaLaBer-System wird erst in Kapitel 5 eingegangen.
Um die vier Informationsebenen in den nachfolgenden Abschnitten noch etwas stärker voneinander abgrenzen zu können, werden die folgenden Unterscheidungskriterien eingeführt:

- **Informationsquelle**:
  Die Informationsquelle gibt die Herkunft der Information an. Für einen dynamischen Lastbalancierer gibt es im wesentlichen zwei wichtige Informationsquellen, das Betriebssystem und die Anwendungen. Unter „Anwendung“ verstehen wir hier das Anwendungsprogramm und evtl. zusätzliche Angaben über die Anwendung in einer Ressourcendefinitionsdatei.
- **Zeitbezug**:
  Der zeitliche Bezug der Information gibt den Zeitraum an, über den die Information etwas aussagt bzw. für den sie gültig ist. Die gröbst mögliche Unterteilung unterscheidet zwischen Informationen, die vergangenes Verhalten bschreiben, solchen, die die Gegenwart charakterisieren und solchen, die die Zukunft betreffen.
- **Informationsqualität**:
  Die Qualität der Information gibt den Detaillierungsgrad der Information an. Dabei wird zwischen detaillierter Information und allgemeinen Einschätzungen unterschieden.
- **Aktualität**:
  Die Aktualität der Information gibt den Neuheitsgrad der Information an. Unterschieden wird dabei zwischen sehr aktueller, aktueller und nicht aktueller Information.
- **Verläßlichkeit**:
  In engem Zusammenhang mit der Aktualität der Information steht deren Verläßlichkeit. Dieses

Kriterium gibt die Korrektheit bzw. die Vertrauenswürdigkeit der Information an. Die Verläßlichkeit einer Information kann hoch, mittel oder niedrig sein.

- **Häufigkeit der Benutzung**:
  Die Benutzungshäufigkeit gibt darüber Auskunft, ob die Information nur zu bestimmten Entscheidungen oder ständig von einem dynamischen Lastbalancierer verwendet wird, die möglichen Ausprägungen sind regelmäßig und unregelmäßig.
  Informationen, die ein dynamischer Lastbalancierer regelmäßig verwendet, sind z.B. alle Informationen, die in die Berechnung des Lastzustandes eingehen. Hingegen werden Informationen, die im Zusammenhang mit der Auswahlentscheidung benötigt werden, nur unregelmäßig verwendet.
- **Häufigkeit der Verfügbarkeit**:
  Die Verfügbarkeit gibt an, ob die Information einem dynamischen Lastbalancierer permanent oder nur sporadisch zur Verfügung steht. In engem Zusammenhang damit steht auch die Frage, ob der Lastbalancierer die Information bei Bedarf anfordern kann (permanente Verfügbarkeit, *pull-Modell*) oder ob er die Information zu unvorhersehbaren Zeitpunkten übergeben bekommt (sporadische Verfügbarkeit, *push-Modell*).
- **Informationscharakter**:
  Mit Hilfe dieses Kriteriums wird die Information in Prognosen und Tatsachen unterteilt. Der Informationscharakter steht dabei natürlich in engem Zusammenhang mit dem zeitlichen Bezug der Information.
- **Gültigkeit**:
  Die Gültigkeit einer Information gibt die Zeitdauer an, für die die Information gültig ist, die drei möglichen Ausprägungen sind Punktdaten, Intervalldaten und Invarianten. Die Gültigkeit einer Information darf dabei nicht mit deren zeitlichen Bezug (Vergangenheit, Gegenwart, Zukunft) verwechselt werden.

Ausgehend von diesen Unterscheidungskriterien werden in den nachfolgenden Abschnitten die vier Informationsebenen stärker differenziert und typische Informationsgrößen aus den Ebenen exemplarisch klassifiziert. Jeder Abschnitt schließt mit einem kurzen Literaturüberblick.

3.2 Die Informationsebene 0

Die Informationsebene 0 bildet die Ausgangsbasis für jeden dynamischen Lastbalancierer, da nur sie zuverlässige Informationen über den tatsächlichen Systemzustand bereitstellen kann. Für die Informationsebene 0 kommt nur das *Betriebssystem* als Informationsquelle in Frage. Der zeitliche Bezug, der vom Betriebssystem zur Verfügung gestellten Information, reicht dabei von der *Vergangenheit* bis in die *Gegenwart*. So gibt es Informationen, die vom Betriebssystem nur über ein gewisses Zeitintervall gemittelt angeboten werden, wie etwa die mittlere Prozessorwarteschlangenlänge in Unix-Betriebssystemen, bzw. Informationen, die den exakten aktuellen Zustand widerspiegeln. Die Hauptspeicherauslastung ist zum Beispiel eine Information, die zumindest von den Unix-Betriebssystemen, immer aktuell bereitgehalten wird. In jedem Fall handelt es sich bei Betriebssysteminformationen um *detaillierte* Informationen über den Systemzustand. Abhängig von der Bereitstellung der Systeminformationen durch das Betriebssystem variiert sowohl die Aktualität der Information von *sehr aktuell* bis *nicht aktuell* als auch die Verläßlichkeit der Information von *hoch* bis *niedrig*. Da jedoch nur von der Ebene 0 Informationen über den tatsächlichen Systemzustand zu bekommen sind, werden diese Informationen *regelmäßig* verwendet, zumal sie vom Betriebssystem auch *permanent* bereitgestellt werden und abrufbereit sind. Insgesamt handelt es sich bei den Informationen dieser Ebene

um Tatsachen, die sowohl Punktdaten als auch Intervalldaten umfassen.
Die heutigen Betriebssysteme stellen eine Vielzahl von sowohl prozeßbezogenen als auch knotenbezogenen Informationen der Ebene 0 zur Verfügung. So liefert allein die Unix-Routine **getrusage()** [OSF92] zwölf verschiedene Lastinformationen über einen einzelnen Prozeß. Die Lastinformationen reichen dabei von der konsumierten Prozessorzeit über den Hauptspeicherbedarf bis hin zur Anzahl der Kontextwechsel. In Tabelle 11 und Tabelle 12 sind exemplarisch eine prozeßbezogene und eine knotenbezogene Information der Ebene 0 anhand der neun Kritierien eingeordnet. Während die Einordnung der prozeßbezogenen Information weitgehend selbsterklärend ist, bedarf die Einordnung der knotenbezogenen Information gewisser Erläuterung.

Kriterium	**Ausprägung**	**Erklärung**
Informationsquelle	Betriebssystem	Betriebssystem verwaltet diese Information
Zeitbezug	Gegenwart	augenblicklich belegte Hauptspeicherseiten
Informationsqualität	detailliert	exakte Anzahl der belegten Speicherseiten
Aktualität	sehr aktuell	Information wird bei Bedarf abgerufen
Verläßlichkeit	hoch	System garantiert Korrektheit
Benutzung	regelmäßig	Information geht in die Berechnung des Lastzustandes ein
Verfügbarkeit	permanent	Information ist über Systemaufruf abzufragen
Charakter	Tatsache	System garantiert Korrektheit
Gültigkeit	Punktdaten	Information beinhaltet keine zeitliche Entwicklung

Tabelle 11: Hauptspeicherbedarf eines Anwendungsprozesses.

Kriterium	**Ausprägung**	**Erklärung**
Informationsquelle	Betriebssystem	Betriebssystem verwaltet diese Information
Zeitbezug	Vergangenheit/Gegenwart	Information wird über einen Zeitraum gemittelt
Informationsqualität	detailliert	Anzahl der lauffähigen Prozesse pro Zeiteinheit
Aktualität	aktuell	Information wird ständig ermittelt
Verläßlichkeit	mittel	Information wurde aufbereitet (gemittelt)
Benutzung	regelmäßig	Der Prozessor stellt die wichtigste Ressource dar, weshalb diese Information permanent verwendet wird
Verfügbarkeit	permanent	Information wird vom Betriebssystem ständig erfaßt und aufbereitet
Charakter	Tatsache	Information liefert den Ist-Zustand
Gültigkeit	Intervalldaten	Information beinhaltet zeitliche Entwicklung

Tabelle 12: Mittlere Prozessorwarteschlange.

Zur Beurteilung der Prozessorauslastung verwenden viele Lastbalancierungsumgebungen [Ludw93] die mittlere Länge der Prozessorwarteschlange, d.h. die Anzahl der Prozesse, die sich in einem lauffähigen Zustand befinden. Da dieser Lastindikator immer über ein Zeitintervall gemittelt wird (im Betriebssystem Unix werden dazu drei verschiedene Zeitintervalle von 5 Sekunden, 30 Sekunden und 60 Sekunden verwendet), reicht der zeitliche Bezug dieser Information von der Vergangenheit bis in die Gegenwart.
Da es sich bei der Informationsebene 0 um das Informationsfundament eines jeden dynamischen Lastbalancierers handelt, wurden in der Vergangenheit auch eine größere Anzahl von Untersuchungen [ZhFe87][Kunz91] bzgl. der Auswirkungen verschiedener Informationen dieser Ebene auf die Effizienz der dynamischen Lastbalancierung durchgeführt. Die wahrscheinlich umfassenste Untersuchung dieser Ebene stammt von Kunz [Kunz91]. Kunz untersucht in seiner Arbeit mit Hilfe eines stochastischen Lernautomaten die Auswirkungen verschiedener Informationen der Ebene 0 auf die Effizienz der dynamischen Lastbalancierung. Da in dieser Arbeit als Zielsystem ein Unix-Workstationnetz verwendet wurde, wurden für die Untersuchung exemplarisch die folgenden unter Unix verfügbaren Lastinformationen benutzt:

- Anzahl der Prozesse in der Prozessorwarteschlange
- Größe des freien Hautpspeichers
- Prozessor-Kontextwechselrate
- Rate der Betriebssystemaufrufe
- Anzahl der Prozesse in der Prozessorwarteschlange innerhalb der letzten Minute
- Prozessorleerlaufzeit

In einer ersten Untersuchungsreihe wurde jeweils eine der Lastkenngrößen als Entscheidungsgrundlage verwendet. Die besten Laufzeitergebnisse wurden dabei mit der ersten Lastkenngröße „Anzahl der Prozesse in der Prozessorwarteschlange“ erzielt. Dieses Ergebnis stimmt auch mit der Arbeit von Ferrari und Zhou [FeZh86][ZhFe87] überein, die ebenfalls zu dem Ergebnis kommen, daß die Warteschlangenlänge einer Ressource in den meisten Fällen die beste Lastkenngröße darstellt.
In einer zweiten Untersuchungsreihe wurden Kombinationen der sechs Lastkenngrößen als Entscheidungsgrundlage verwendet. Kunz kam dabei zu dem Ergebnis, daß keine Laufzeitverbesserungen gegenüber der ersten Untersuchungsreihe erzielt werden konnten. Dieses Ergebnis stimmt ebenfalls mit den Ergebnissen verschiedener Simulationsstudien überein. So kommen z.B. Eager et al. [EaLZ86/2] zu dem Ergebnis, daß bereits wenige Lastinformationen ausreichen, um drastische Laufzeitgewinne zu erzielen.
Zur Einordnung dieser Ergebnisse muß jedoch darauf hingewiesen werden, daß diese Untersuchungen nicht mit parallelen Anwendungen bzw. Lastgeneratoren durchgeführt wurden. Diese Lastkenngrößenvergleiche haben ergeben, daß sich z.B. die Verwendung von Kommunikationsbeziehungen zwischen den Prozessen der parallelen Anwendung positiv auf das Laufzeitergebnis auswirken kann [Milo94]. Dieses Ergebnis wird in Kapitel 6 durch umfangreiche Messungen bestätigt.

3.3 Die Informationsebene 1

Aufbauend auf der Ebene 0 verwendet die Informationsebene 1 sowohl das *Betriebssystem* als auch die *Anwendungen* als Informationslieferanten. Da es sich bei den Informationen der Ebene 1 um statische bzw. statistische Informationen handelt, ist der zeitliche Bezug die *Gegenwart* und die *Zukunft*. Dabei variiert die Informationsqualität der Ebene 1 erheblich, d.h. es sind sowohl sehr *detaillierte*,

z.B. die Kommunikationspartner eines Prozesses, als auch sehr *umfassende* Informationen erhältlich. So werden in verschiedenen Lastbalancierungssystemen die Anwendungsprozesse auf Grund statistischer Informationen bzgl. ihrer Gesamtlaufzeit in Kurz- und Langläufer unterteilt, wobei nur Prozesse, die zur Gruppe der Langläufer zählen, als Migrationskandidaten in Frage kommen.
Ausgehend von der Tatsache, daß es sich bei den Informationen der Ebene 1 um statische bzw. statistische Sachverhalte handelt, sind diese Informationen *nicht aktuell,* und somit ist die Verläßlichkeit *niedrig*. Da diese Ebene ein großes Informationsspektrum abdeckt, ist die Häufigkeit der Benutzung starken Schwankungen unterworfen, die Verfügbarkeit der Information ist jedoch aufgrund ihrer Natur *permanent*. Insbesondere sind die Informationen der Ebene 1 häufig bereits vor dem Start einer Anwendung verfügbar. Insgesamt handelt es sich bei dieser Ebene sowohl um *Prognosen* als auch um *Tatsachen*, die zeitlich *invariante* Sachverhalte beschreiben.
In Tabelle 13 ist exemplarisch die Information über die statistische Laufzeitabschätzung eines Anwendungsprozesses anhand der entwickelten Unterscheidungskriterien dargestellt. Dabei handelt es sich um eine Information, die sowohl von der Anwendung als auch vom Betriebssystem bereitgestellt werden kann.
In Tabelle 14 ist das Zugriffsverhalten eines Anwendungsprozesses beim Lesen einer sequentiellen Datei klassifiziert. Anhand dieses Beispiels lassen sich die Vor- und Nachteile der Informationsebene 1 gut aufzeigen. Öffnet ein Prozeß eine Datei zum sequentiellen, lesenden Zugriff, ist aufgrund der Zugriffssemantik sofort bekannt, welcher Datenblock als erster von der Platte gelesen wird, welcher als zweiter usw.. Diese sehr detaillierte Information kann ein Betriebssystem im Zusammenhang mit einer effizienten Ressourcenverwaltung in vielerlei Hinsicht verwenden. Allerdings ist diese Information mit einer erheblichen Unsicherheit behaftet. Denn obwohl bekannt ist, welcher Datenblock als erster in den Hauptspeicher transferiert werden muß, ist nicht bekannt, ob Daten aus der Datei eingelesen werden, d.h. ob überhaupt ein Datenblock transferiert werden muß.
Obwohl die Informationen der Ebene 1 überwiegend im Zusammenhang mit der statischen Lastbalancierung und dem dynamischen Remapping verwendet werden, gibt es auch einige Arbeiten, die sich mit den Auswirkungen der Informationen der Ebene 1 auf die Effizienz der dynamischen Lastbalancierung auseinandersetzen [LeEl93] [YuLL91]. Ein sehr interessanter Ansatz, Informationen aus der Ebene 1 für einen dynamischen Lastbalancierer verfügbar zu machen, stammt von Devarakonda et al. [DeIy89]. Sie haben von über 65 000 Unix-Prozessen auf einer VAX 11/780 den Ressourcenbedarf ermittelt, um ausgehend davon den zukünftigen Ressourcenbedarf einer Anwendung zu bestimmen. Dazu wurden zu jedem Prozeß die folgenden Ressourcendaten erfaßt:

- Prozessorbedarf
- Hauptspeicherbedarf
- Ein-/Ausgabeaufkommen
- Gesamtlaufzeit der Anwendung

Die ersten drei Resssourcenwerte spannen bei diesem Ansatz einen dreidimensionalen Raum auf. Der Ressourcenbedarf eines Prozesses kann somit als Punkt in diesem Raum dargestellt werden. Auf diese Weise wurden für jede Anwendung die ermittelten Messungen erfaßt und unter Verwendung eine Clusteranalyse in mehrere Cluster aufgeteilt. Um die erhaltenen Cluster in Beziehung zu setzen, wurden dann in einem zweiten Berechnungsschritt Übergangswahrscheinlichkeiten zwischen den Clustern ermittelt. Dazu wurden sämtliche Meßläufe einer Anwendung nach dem Beendigungszeitpunkt sortiert und in aufsteigender Reihenfolge die Übergänge zwischen den Clustern erfaßt. Das Ergebnis der Berechnung war ein Zustandsübergangsdiagramm, mit dessen Hilfe der zukünftige Ressourcenbedarf einer neu zu startenden Anwendung abgeschätzt werden konnte.
Dieses Verfahren wurde von den Autoren zusammen mit einem zentralen Lastbalancierungsansatz

Kriterium	Ausprägung	Erklärung
Informationsquelle	Betriebssystem/Anwendung	Information kann sowohl von der Anwendung als auch vom Betriebssystem erfaßt werden
Zeitbezug	Zukunft	Information steht bereits vor dem Start der Anwendung zur Verfügung
Informationsqualität	detailliert	vorausgesetzt, es wird nicht nur zwischen Kurz- und Langläufern unterschieden
Aktualität	nicht aktuell	Information basiert auf vorangegangenen Anwendungsläufen
Verläßlichkeit	niedrig	aktuelle Aufrufparamter, Daten etc. werden nicht berücksichtigt
Benutzung	regelmäßig	Information wird für die Schätzung der zukünftigen Knotenauslastung benötigt
Verfügbarkeit	permanent	
Charakter	Prognose	
Gültigkeit	Invariante	

Tabelle 13: Statistische Laufzeitabschätzung eines Anwendungsprozesses.

Kriterium	Ausprägung	Erklärung
Informationsquelle	Betriebssystem	Das Dateisystem wird vom Betriebssystem verwaltet
Zeitbezug	Gegenwart/Zukunft	ergibt sich aus der Definition für die Ebene 1
Informationsqualität	umfassend	(siehe obige Erläuterung)
Aktualität	nicht aktuell	
Verläßlichkeit	niedrig	
Benutzung	unregelmäßig	
Verfügbarkeit	permanent	
Charakter	Prognose	
Gültigkeit	Invariante	

Tabelle 14: Zugriffsmuster auf eine sequentielle Datei.

auf seine Leistungsfähigkeit hin überprüft. Dabei stellte sich heraus, daß der zentrale Lastbalancierer unter zusätzlicher Verwendung der statistischen Ressourcenabschätzung Laufzeitgewinne von bis zu 35% erzielen konnte.

3.4 Die Informationsebene 2

Aufgrund ihrer Definition kommen für die Informationsebene 2 als Informationsquelle nur die Anwendungen in Frage. Da die Anwendungen eine Vielzahl von Informationen über ihren *gegenwärtigen* Bearbeitungszustand an den Lastbalancierer weiterleiten können, reicht die Palette der mögli-

chen Informationsqualitäten von *Detail*-Informationen (Anzahl der Kommunikationspartner in der aktuellen Bearbeitungsphase) bis hin zu *umfassenden*, generellen Aussagen (vgl. Tabelle 15). Der große Vorteil der Informationsebene 2 gegenüber der Informationsebene 1 ist der hohe Aktualitäts- und Verläßlichkeitsgrad dieser Information. Da diese Information von der Anwendung zur Verfügung gestellt wird, ist sie nur *sporadisch* verfügbar und nicht vom Lastverwalter beliebig abrufbar. Die Benutzung dieser Information durch einen dynamischen Lastbalancierer ist unregelmäßig, da die prozeßbezogene Information nicht für die Berechnung des Lastzustandes benötigt wird. Insgesamt handelt es sich hierbei um Tatsacheninformationen, die sich jeweils auf ein Intervall, die aktuelle Bearbeitungsphase, beziehen.
In Tabelle 15 ist ein Beispiel für eine umfassende Information der Ebene 2 wiedergegeben. Ein Prozeß, der sich in einer Ein-/Ausgabephase befindet, ist aufgrund des erhöhten Verwaltungsaufwandes kein geeigneter Migrationskandidat [Milo94]. Der wesentliche Vorteil dieser Informationsebene besteht darin, daß sie es dem Lastbalancierer ermöglicht, seine durch Messung, Beobachtung etc. erfaßten prozeßbezogenen Lastdaten mit den Informationen von der Anwendung zu vergleichen und somit zu bestätigen bzw. zu korrigieren. Diese Möglichkeiten der Informationsebene 2 sind insbesondere im Zusammenhang mit Informationen der Ebene 0 mit einem Zeitbezug in die Vergangenheit von Bedeutung (z.B. mittlere Länge der Prozessorwarteschlange).
Die Informationen der Ebene 2 werden in nahezu allen anwendungsintegrierten Lastbalancierungsansätzen verwendet [ChYL95][SaEr87]. Diese Information wird dabei implizit verwendet, da bei diesen Ansätzen eine dynamische Lastbalancierung häufig nur zu bestimmten Zeitpunkten angestoßen wird, in denen entweder der Lastausgleich mit relativ geringem Aufwand durchgeführt werden kann, oder zu denen der Lastbalancierer auf die benötigten, entscheidungsrelevanten Informationen zugreifen kann.

Kriterium	**Ausprägung**	**Erklärung**
Informationsquelle	Anwendung	ergibt sich aus der Definition für die Ebene 2
Zeitbezug	Gegenwart	Anwendungsprozeß meldet den aktuelle Ist-Zustand
Informationsqualität	umfassend	
Aktualität	sehr aktuell	Anwendungsprozeß betritt im Augenblick der Meldung die entsprechende Bearbeitungsphase
Verläßlichkeit	hoch	Anwendung kennt ihr Verhalten
Benutzung	unregelmäßig	Information wird bei der Auswahl eines Migrationskandidaten benötigt
Verfügbarkeit	sporadisch	Information ist nur während eines Wechsels der Bearbeitungsphase verfügbar
Charakter	Tatsache	
Gültigkeit	Intervalldaten	Information bezieht sich immer auf eine Phase

Tabelle 15: Anwendungsprozeß meldet den Eintritt in eine Ein-/Ausgabephase.

3.5 Die Informationsebene 3

Diese Informationsebene bringt einem dynamischen Lastbalancierer einen echten Informationsvorsprung, denn die Informationen der Ebene 3 stammen von der *Anwendung* und betreffen deren voraussichtliches *zukünftiges* Lastverhalten. Wie bereits bei der Ebene 2 erstreckt sich die verfügbare Information von sehr *detaillierter* Information (Abschätzung der Restlaufzeit) bis hin zu generellen, *umfassenden* Einschätzungen (vorzeitige Ankündigung der Terminierung) des zukünftigen Lastzustandes. Dabei ist es selbstverständlich, daß aufgrund des Zeitbezuges diese Information *sehr aktuell,* jedoch *nicht besonders verläßlich* ist. Obwohl die Anwendungen diese Information nur *sporadisch* zur Verfügung stellen können, eignen sie sich aufgrund ihrer Struktur für eine *regelmäßige* Benutzung. Insgesamt handelt es sich bei diesen Informationen um *intervallbezogene Prognosen* mit großen Leistungspotentialen für einen dynamischen Lastbalancierer.
In Tabelle 16 ist die Klassifikation der Abschätzung der Restlaufzeit eines Anwendungsprozesses dargestellt. Diese Information ist für ein dynamischen Lastbalancierer sehr wichtig, um abschätzen zu können, ob sich für einen Anwendungsprozeß eine Migration noch lohnt, d.h. sich positiv auf seine Gesamtlaufzeit auswirkt.
Trotz ihrer großen Bedeutung für einen dynamischen Lastbalancierer wurde die Informationsebene 3 in den bisherigen Lastbalancierungsansätzen weitgehend ignoriert. Eines der wenigen Lastbalancierungssysteme, das Anwendungsprognosen für die Entscheidungsfindung verwendet, ist das HiCon-System [Beck95]. Das HiCon-System ist eine dynamische Lastbalancierungsumgebung für Client-/Serveranwendungen. Zum Informationsaustausch zwischen den Servern und dem Client stellt das System einen virtuellen, gemeinsamen Speicher zur Verfügung. Der virtuelle Speicher ist dabei in Partitionen aufgeteilt, auf die die Anwendungen über entsprechende Sperranforderungen zugreifen können. Übergibt der Client dem Lastbalancierer einen neuen Berechnungsauftrag, so hat er die Möglichkeit, dem Lastbalancierer zusätzlich noch Informationen über den Ressourcenbedarf des Auftrags mitzuteilen. Dazu gehören Informationen bzgl.

- Prozessorbedarf
- Anzahl und Identität der benötigten Datenpartitionen

Diese Informationen kann der Lastbalancierer einerseits für seine Plazierungsentscheidung ver-

Kriterium	**Ausprägung**	**Erklärung**
Informationsquelle	Anwendung	(siehe obige Erläuterungen)
Zeitbezug	Gegenwart/Zukunft	
Informationsqualität	detailliert	
Aktualität	sehr aktuell	
Verläßlichkeit	mittel/gering	
Benutzung	regelmäßig	
Verfügbarkeit	sporadisch	
Charakter	Prognose	
Gültigkeit	Intervalldaten	

Tabelle 16: Abschätzung der Restlaufzeit eines Anwendungsprozesses.

wenden und andererseits an das Laufzeitsystem weiterreichen. Dieses hat dadurch die Möglichkeit, die benötigten Datenpartitionen auf dem Server verfügbar zu machen. Leider sind keine darauf

fokussierten Vergleichsmessungen verfügbar, so daß für dieses konkrete System nicht gesagt werden kann, inwieweit sich diese Informationen auf das Laufzeitergebnis auswirken.

Das Ziel dieses Kapitels war die Definition der Informationsebenen sowie die Motivation für die in den nachfolgenden Kapiteln durchgeführten Untersuchungen. Dabei wurde bei der Definition der Ebenen besonderen Wert auf deren praktische Umsetzbarkeit gelegt, insbesondere sollte es möglich sein, bestehende dynamische Lastbalancierungsansätze einordnen zu können.
Auf die konkrete technische Umsetzung dieser Ebenen wurde im diesem Kapitel bewußt verzichtet, da diese vom jeweilig verwendeten Lastbalancierungssystem abhängig ist - folgerichtig wird auf diesen Punkt im Kapitel 5, in dem die hierarchische Lastbalancierungsumgebung PaLaBer vorgestellt wird, ausführlich eingegangen.

Kapitel 4

Beschreibung des Zielsystems

In diesem Kapitel werden die wichtigsten Eigenschaften und Charakteristiken des Zielsystems Intel Paragon XP/S [Inte92][Inte93] vorgestellt. Obwohl der im nächsten Kapitel vorgestellte Lastbalancierungsansatz sehr flexibel ist und über eine Vielzahl von Hardware-Plattformen hinweg anwendbar ist, müssen für eine konkrete und effiziente Implementierung Details des Zielsystems berücksichtigt werden. So müssen für jedes Zielsystem von neuem die relevanten Ressourcenengpässe durch Messungen und Beobachtungen ermittelt und daraufhin untersucht werden, ob ein dynamischer Lastbalancierer positiv auf diese Ressourcenengpässe einwirken kann.

4.1 Beschreibung der Hardware

Der Parallelrechner Intel Paragon XP/S [BBDEGK94][BeBD95] ist ein MIMD - Rechner mit verteiltem Speicher und einem regulären Verbindungsnetzwerk. Die Verarbeitungsknoten sind in einem zweidimensionalen Rechteckgitter angeordnet. Das System besteht aus homogenen Knoten. Jeder Knoten (vgl. Abbildung 15) besteht aus folgenden Komponenten:

- einem Intel i860 XP Anwendungsprozessor
  (50 MHz, 42 Mips, 75 MFLOPS, 64 Bit-Arithmetik)
- einem Intel i860 XP Kommunikationsprozessor
- zwischen 16 und 128 MB lokalem Hauptspeicher
- einem Performance Monitor
- zwei DMA - Controllern
- einem Network Interface Controller (NIC)
- einem Expansion Port

Der Anwendungsprozessor ist für die Abarbeitung der Anwendungsprozesse zuständig. Auf jedem Knoten können gleichzeitig mehrere Anwendungsprozesse laufen. Der Kommunikationsprozessor ist für die Nachrichtenverarbeitung, Synchronisation sowie für globale Operationen zuständig. Beide Prozessoren haben Zugriff auf den lokalen Hauptspeicher. Der eigentliche Datenaustausch zwischen

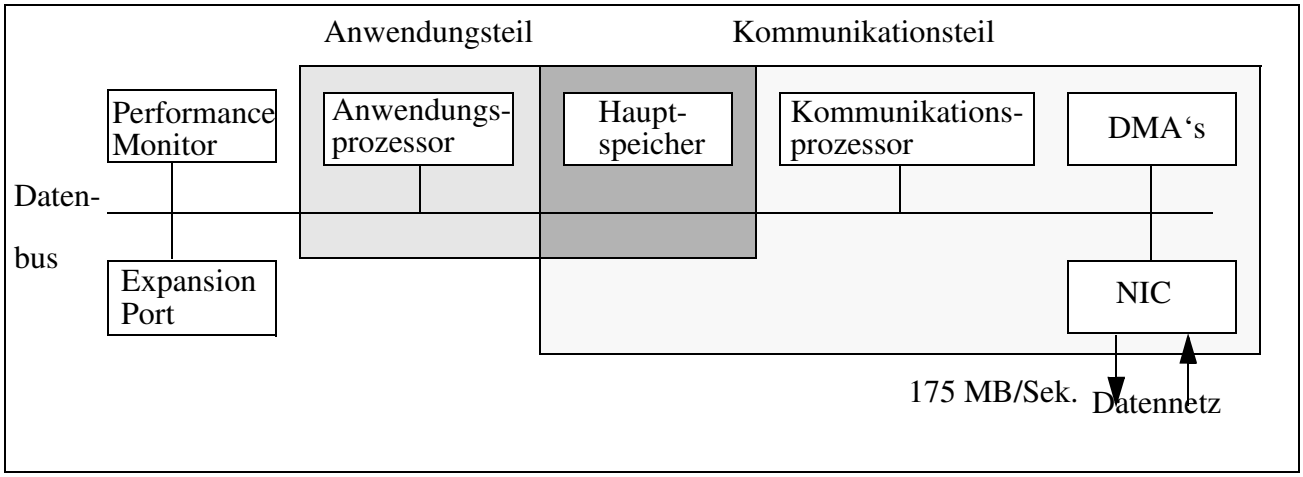


Abbildung 15: Knotenarchitektur [EsKn93].

dem Datennetzwerk und dem lokalen Hauptspeicher erfolgt durch den Network Interface Controller, der von den zwei DMA-Controllern unterstützt wird. Der Performance Monitor ermittelt durch Beobachtung des knotenlokalen Datenbusses die Lastkennwerte des jeweiligen Knotens. Diese Lastwerte werden vom Nachrichtenprozessor gelesen und an einen Knoten der Service-Partition (siehe Abschnitt 4.2) übermittelt. Der Expansion Port dient zum Anschluß von Ein-/Ausgabegeräten. Alle Komponenten sind über einen 32-Bit-Adreßbus und einen 64-Bit-Datenbus (400 MB/Sek.) verbunden.

Die Intel Paragon XP/S verfügt über zwei Kommunikationsnetzwerke, ein Diagnosenetzwerk und ein Datennetzwerk. Das Diagnosenetzwerk wird während des Bootvorgangs und, wie der Name schon andeutet, zu Diagnosezwecken verwendet. Der Informationsaustausch zwischen den Anwendungsprozessen erfolgt über das Datennetzwerk (16 Bit, 200 MB/Sek. voll-duplex). Das Datennetzwerk (vgl. Abbildung 16) besteht aus Mesh Router Controllers (MRC) und Hochgeschwindigkeitskanälen. Die MRC‘s sind in einem zweidimensionalen Rechteckgitter angeordnet. Benachbarte MRC‘s sind dabei immer über zwei Kanäle miteinander verbunden. An jeden MRC kann ein Verarbeitungsknoten mit Hilfe des NIC angeschlossen werden. Die MRC‘s führen das Nachrichtenrouting autonom, d.h. unabhängig von den Verarbeitungsknoten, durch. Da die Datenübertragung zwischen dem NIC und dem MRC 175 MB/Sek. beträgt, kann zwischen zwei beliebigen Verarbeitungsknoten nicht die maximale Übertragungsleistung des Datennetzwerkes von 200 MB/Sek. erreicht werden.

Beim Datennetzwerk wird als Übertragungsstrategie das *Wormhole Routing* [NiMc93] angewandt. Das Wormhole Routing ist ein paketorientiertes Vermittlungsprotokoll, bei dem jedes Paket (maximal 2 KB) noch weiter in sogenannte *flits* (flow control digits) aufgeteilt wird. Die Größe eines *flits* beträgt bei der Intel Paragon 16 bit, die zwischen zwei benachbarten MRC‘s parallel übertragen werden. Die zu einem Paket gehörigen *flits* werden dabei in einer Pipeline vom Senderknoten zum Empfänger übertragen. Das erste *flit* (header flit) bestimmt den Pfad durch das Netzwerk, die restlichen *flits* (data flits) folgen dann diesem Pfad. Wird das *header flit* blockiert, da ein Kanal bereits durch ein anderes Paket exklusiv belegt ist, wird die gesamte Übertragung gestoppt.

Da beim *Wormhole Routing* prinzipiell die Gefahr eines Deadlocks besteht, findet bei der Intel Paragon ein deterministischer Routing-Algorithmus Anwendung. Dabei wird ein Paket zuerst in horizontaler Richtung und anschließend in vertikaler Richtung durch das zweidimensionale Gitter geroutet, d.h. pro Übertragung ist maximal ein Richtungswechsel erlaubt. Zur Umsetzung des Algorithmus werden pro Paket zwei *header flits* verwendet. Das erste legt die Orientierung und Anzahl der *hops* in horizontaler Richtung fest, das zweite *header flit* in vertikaler Richtung. Aufgrund der pipelineartigen Übertragung beim *Wormhole Routing* ist die Übertragungszeit einer Nachricht

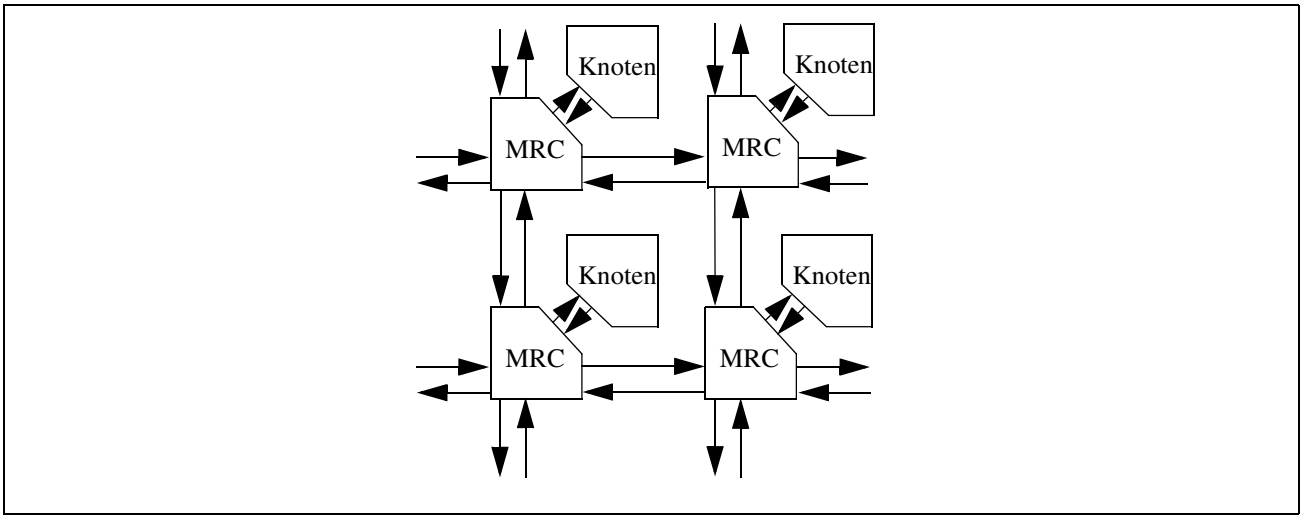


Abbildung 16: Verbindungsnetzwerk.

nahezu unabhängig von der Entfernung zwischen Sender- und Empfängerknoten. In Tabelle 17 sind die Übertragungszeiten in Abhängigkeit der Distanz und der Nachrichtenlänge aufgeführt.
Die Abweichung der Übertragungszeiten bei der Distanz 0, d.h. bei der Kommunikation innerhalb eines Knotens, ergibt sich nicht zuletzt aus der Notwendigkeit eines Prozeßwechsels. Befinden sich zwei Prozesse auf einem Knoten muß der sendende Prozeß den Prozessor abgeben, damit der andere Prozeß die Nachricht empfangen kann. Mit anderen Worten, bei der Kommunikation innerhalb eines Knotens werden die Kommunikationszeiten im Wesentlichen durch die Zeiten für den Kontextwechsel bestimmt. Diese Leistungseinbußen wirken sich jedoch nur dann voll auf die Laufzeiten einer parallelen Anwendung aus, wenn zwischen zwei Prozessen eine sogenannte Ping-Pong-Kommunikation stattfindet. Eine Ping-Pong-Kommunikation liegt genau dann vor, wenn der Empfänger einer Nachricht sofort nach dem Empfangen eine Anwortnachricht schickt und der ürsprüngliche Sender sich sofort in einer Empfangsroutine blockiert. In realen parallelen Anwendungen finden jedoch zwischen dem Empfangen ein Nachricht und dem Senden einer Antwortnachricht häufig aufwendigere Berechnungen statt, so daß ein sofortiger Kontextwechsel nicht erforderlich ist.
Insgesamt läßt sich die Intel Paragon mit Hilfe des in Kapitel 1 eingeführten Erlanger Klassifikationssystems für Rechnerarchitekturen darstellen als (1x1, 1x1, 32x1) + (84x2, 1x2, 64x3), wobei das erste Tripel die Diagnosestation und das zweite Tripel den eigentlichen Parallelrechner beschreibt.

Nachrichtenlänge	Distanz 0	Distanz 1	Distanz 5	Distanz 10
0 KB	522 µs	36 µs	41 µs	41 µs
5 KB	583 µs	155 µs	156 µs	154 µs
10 KB	621µs	298 µs	326 µs	324 µs
20 KB	1137µs	505 µs	552 µs	548 µs
30 KB	1611µs	706 µs	774 µs	765 µs

Tabelle 17: Übertragungszeiten in Abhängigkeit der Knotenentfernung.[a]

a. Diese Messungen wurden mit Hilfe der synchronen Kommunikationsroutinen unter dem Betriebssystemrelease R.1.3.4 durchgeführt.

4.2 Das Partitionierungskonzept

Die Tatsache, daß die Intel Paragon aus homogenen Verarbeitungsknoten aufgebaut ist, bedeutet nicht, daß alle Knoten die gleichen Aufgaben zu erledigen haben bzw. im gleichen Modus betrieben werden. Stattdessen werden die Knoten in sogenannte Partitionen [Inte94] aufgeteilt. Eine Partition ist eine Menge von Knoten, die im gleichen Modus betrieben werden. Standardmäßig müssen mindestens drei Partitionen angelegt sein:

- die Root-Partition,
- die Service-Partition und
- die Compute-Partition.

Die *Root-Partition* beinhaltet alle Verarbeitungsknoten des Systems. Alle weiteren Partitionen sind somit Unterpartitionen der Root-Partition.
In der *Service-Partition* werden alle gängigen Benutzerdienste wie Editoren, Übersetzer und Unix-Shells betrieben. Der Betriebsmodus der Service-Partition ist vergleichbar mit dem von Unix-Workstations, d.h. als Prozessorzuteilungsstrategie wird ein Zeitscheibenverfahren (~100 ms) verwendet. Die Service-Partition umfaßt normalerweise auch die I/0-Knoten zum Anschluß von Platten und externen Netzwerken.
Die *Compute-Partition* dient zur Abarbeitung der parallelen Anwendungen, wobei jede Anwendung in einer eigenen Unterpartition abläuft. Zu diesem Zweck kann die Compute-Partition zur Laufzeit hierarchisch in Unterpartitionen aufgeteilt werden. Die Compute-Partition wird im sogenannten Gang-Scheduling-Modus betrieben. In diesem Betriebsmodus werden alle Prozesse, die zu einer parallelen Anwendung gehören, auf allen Knoten der Unterpartition als Einheit verplant. Daraus folgt, daß eine parallele Anwendung alle Verarbeitungsknoten der Unterpartition exklusiv zur Verfügung hat.

4.3 Beschreibung der Softwarestruktur

Die Intel Paragon XP/S wird mit dem verteilten Betriebssystem OSF1/AD [Hibb93] betrieben. OSF1/AD ist die „Advanced Development“ Mehrprozessorversion des OSF1, das von der Open Software Foundation entwickelt wurde. Wie bereits OSF1 basiert auch OSF1/AD auf dem Mach3-Mikrokernel [TeRa87][BKLL93][ZiKr93], der von der Carnegie Mellon University entwickelt wurde. Da es sich bei der Intel Paragon XP/S um eine Shared-Nothing-Architektur handelt, wurde die NORMA Version (NO Remote Memory Access) des Mach-Mikrokernel verwendet. Die Softwarestruktur des Betriebssystems ist in Abbildung 17 dargestellt. Die verschiedenen Betriebssystemkomponenten werden nachfolgend kurz erklärt.
Der Mach-Mikrokernel stellt im wesentlichen 5 Basismechanismen zur Erzeugung höherer Systemdienste zur Verfügung:

- Prozesse
- Threads
- Ports
- Messages
- Speicherobjekte

Ein Mach-*Prozeß* ist ein passives, abstraktes Gebilde, das selbst keine bzw. nur sehr wenige Operationen ausführen kann. Ein Prozeß stellt lediglich die Arbeitsumgebung für Threads zur

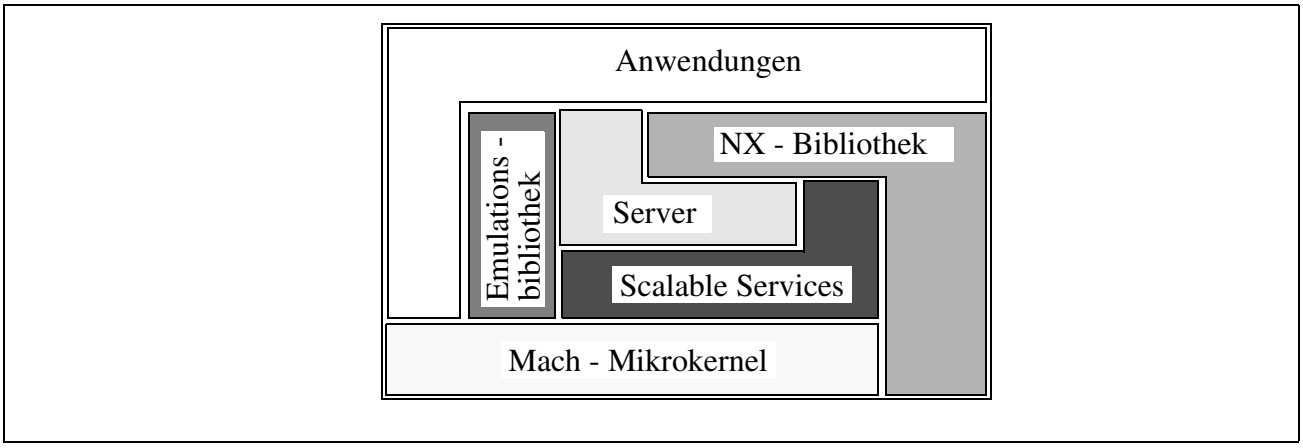


Abbildung 17: Softwarestruktur.

Verfügung, indem er Ressourcen (Adreßraum, Port Name Space, Statistiken, Kernel-interne Parameter) für die Threads verwaltet. Jeder Prozeß enthält dabei mindestens einen Thread.
Threads sind die eigentlichen Aktivitätsträger in Mach. Innerhalb eines Prozesses können mehrere Threads konkurrierend ausgeführt werden. Da Threads keinen eigenen Adreßraum besitzen - dieser wird ja vom Prozeß zur Verfügung gestellt - haben alle Threads eines Prozesses Zugriff auf den gemeinsamen Adreßraum. Ein klassischer Unix-Prozeß ist somit ein Mach-Prozeß mit einem Thread als Aktivitätsträger.
Damit auch Threads, die zu unterschiedlichen Prozessen gehören, Informationen austauschen können, stellt Mach das *Port-Konzept* zur Verfügung. Ein Port ist ein unidirektionaler Kanal und ist immer einem Prozeß zugeordnet. Um auf einen Port lesend oder schreibend zugreifen zu können, benötigt ein Thread das entsprechende Zugriffsrecht. Mach unterscheidet drei Zugriffsrechte: Leserecht, Schreibrecht und das Recht des *send-once*. Zugriffsrechte können bei der Erzeugung eines neuen Prozesses vererbt werden oder explizit mit einer Nachricht weitergegeben werden. Ein weiterer Vorteil des Port-Konzeptes ist die Realisierung der ortstransparenten Kommunikation, d.h. der Sender einer Nachricht muß nicht wissen, auf welchem Knoten des Systems sich der Empfänger befindet, da die Adressierung nur über die PortID erfolgt. Durch diese Eigenschaft ist es in Mach sehr einfach, benutzertransparent verteilte Dienste zu realisieren.
Während in Unix Nachrichten als untypisierte Byteströme realisiert sind, verwendet Mach typisierte Nachrichten (*Messages*). Eine Message kann alle Mach bekannten Datentypen und darauf basierende Typen enthalten.
Der letzte Basismechanismus im Sinne der obigen Aufzählung, den Mach zur Verfügung stellt, sind die *Speicherobjekte*. Jeder Speicherbereich, der von einem Prozeß angefordert wird, wird eindeutig einem Speicherobjekt zugeordnet. Jedes Speicherobjekt wird zu seiner Verwaltung einem *Memory-Manager* zugewiesen, wobei in unterschiedlichen Memory-Managern unterschiedliche Ressourcenverwaltungsstrategien verwendet werden können. Auf Speicherobjekte können mehrere Prozesse Zugriff haben (shared memory), diese Prozesse müssen dabei nicht auf dem gleichen Knoten laufen. Die Abbildung des Speicherobjektes in die Adreßräume der verschiedenen Prozesse erfolgt dabei durch einen sogenannten External-Memory-Manager (XMM), der als Anwendungsprozeß außerhalb des Mikrokernels realisiert werden kann.
Da Mach nur die fünf aufgeführten Basismechanismen zur Verfügung stellt, müssen alle höheren Systemdienste (Dateisystem, ...) von Betriebssystemservern, die als normale Anwendungsprozesse realisiert sind, bereitgestellt werden. Auf jedem Paragonknoten läuft somit neben dem Mikrokernel noch ein OSF1/AD-Server, der die gesamte Unix-Funktionalität bereitstellt. Damit ein Unix-Prozeß die Dienste des Betriebssystemservers transparent nutzen kann, wird an jeden Prozeß noch die Emu-

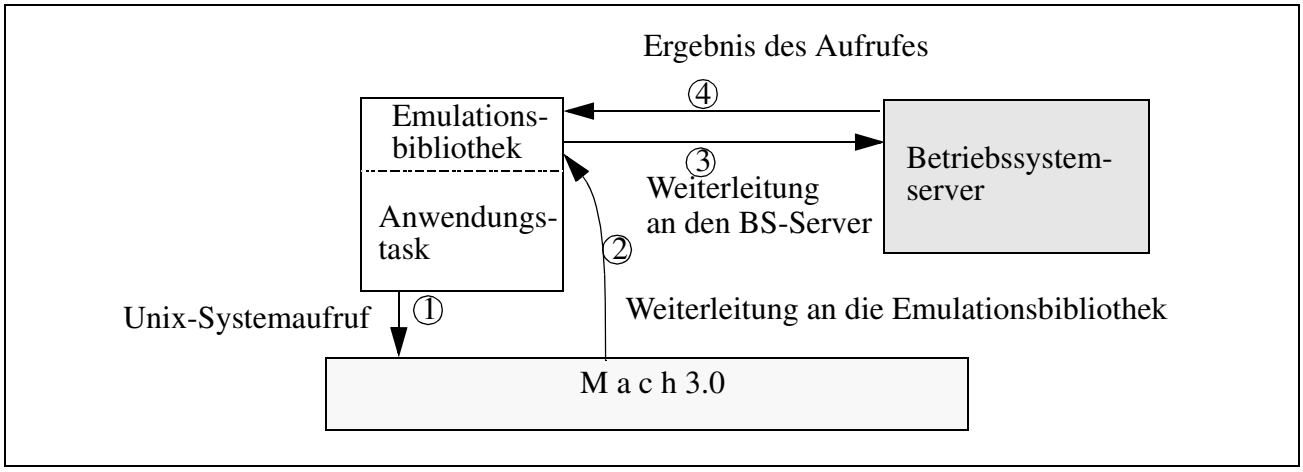


Abbildung 18: Aufruf des Betriebssystemservers unter Mach 3.0.

lationsbibliothek gebunden, die die Betriebssystemaufrufe an den Server weiterleitet.Der Ablauf eines Unix-Systemaufrufs ist in Abbildung 18 dargestellt.
Das verteilte Betriebssystem OSF1/AD realisiert mit Hilfe der *Scalable-Services* ein sogenanntes *Single-System-Image*, d.h. die verteilte Realisierung der Betriebssystemdienste auf den Knoten des parallelen Systems sind für den Anwender transparent. Dies bedeutet beispielsweise, daß Prozeßkennungen systemweit eindeutig sind, daß von jedem Knoten des Systems auf das Dateisystem zugegriffen werden kann, etc.
Die letzte wichtige Softwarekomponente in Abbildung 17 ist die *NX-Bibliothek* von Intel [Inte94] [Pier94][PiRe95]. Diese Bibliothek stellt sowohl Kommunikationsroutinen und Prozeßverwaltungsroutinen zur Erstellung von parallelen Anwendungen sowie ein paralleles Dateisystem (PFS) zur Verfügung. Aus Effizienzgründen setzt diese Bibliothek teilweise direkt auf der Hardware auf. Während mit dem Unix-Dateisystem (UFS) der Durchsatz eines I/O-Knotens auf 500 KB/Sek. begrenzt ist, kann mit Hilfe des parallelen Dateisystems der Durchsatz pro I/O-Knoten auf über 2 MB/Sek. gesteigert werden. Das PFS unterstützt drei Arten der Parallelität [BeBD95]:

1. Striping der Daten innerhalb eines RAID-Systems (Redundant Array of Inexpensive Disks)
2. Striping der Daten über alle beteiligten RAID-Systeme hinweg
3. Paralleler Zugriff auf die Daten von mehreren Prozessorknoten aus

4.4 Der verwendete Lasttransfermechanismus

Zur Realisierung des Lasttransfers gibt es eine Vielzahl von Möglichkeiten. Bei der Auswahl eines Lasttransfermechanismuses muß zwischen der Portabilität des Lastverwaltungssystems und der Benutzertransparenz abgewogen werden. Portable Lasttransferansätze wie die Auftragsmigration [ASLV95][Beck95], Datenmigration [GHWZ94] und Objektmigration [BuMa93][Kalé90] erfordern in aller Regel eine Neuimplementierung der parallelen Anwendung, um sie mit den jeweiligen Lastbalancierungssystemen unterstützen zu können. Dagegen sind die Threadmigration [CaKO94] bzw. die Prozeßmigration [ArFi89][DoOu91][Gait90][Osse92] weitgehend benutzertransparent. Es besteht jedoch die Einschränkung, daß sie nicht portabel und auf eine homogene Rechnerumgebung beschränkt sind, wobei es in jüngster Zeit einige interessante Ansätze im Bereich der Prozeßmigration in heterogenen Rechnernetzen [Pleier96] gibt. Da mit der in Kapitel 5 vorgestellten hierarchischen Lastbalancierungsumgebung PaLaBer das Ziel verfolgt wird, bereits existierende, grobgranulare, nachrichtengekoppelte, parallele Anwendungen zu unterstützen, ohne diese neu implementieren zu müssen, kommt als Lasttransfermechanismus nur die Prozeßmigration in Frage.

In diesem Abschnitt wird der Lasttransfermechanismus beschrieben, der im PaLaBer-System zur Umsetzung der Migration eines bereits laufenden Prozesses verwendet wird. Zum Transfer von bereits gestarteten Prozessen über Knotengrenzen hinweg wird die nx_nfork()-Routine, die Bestandteil der NX-Bibliothek von Intel ist, verwendet.
Die Lasttransferroutine arbeitet in zwei Phasen. In der ersten Phase wird auf dem Zielknoten ein neuer Unix-Prozeß generiert. Dazu wird ein neuer Mach-Prozeß mit drei Threads (Anwendungsthread, NX-Thread und Emulationsthread) erzeugt und anschließend beim zuständigen Betriebssystemserver auf dem Zielknoten die dazugehörige Unix-Prozeßabstraktion angelegt. In der ersten Phase werden jedoch noch keine Daten vom Quell- auf den Zielknoten transferiert. Der Kindprozeß erhält lediglich Zugriffsrechte auf die Hauptspeicherseiten des Elternprozesses auf dem Quellknoten. Die Generierung eines neuen Unix-Prozesses auf dem Zielknoten benötigt dabei zwischen 700 und 800 Millisekunden. Diese Zeit wird im Zusammenhang mit der Prozeßmigration auch als *Freezing Time* bezeichnet, da in dieser Zeit weder der Eltern- noch der Kindprozeß aktiv sind.
Zu Beginn der zweiten Phase wird der Elternprozeß terminiert und der Kindprozeß gestartet. Da im Laufe der ersten Phase keinerlei Datenseiten vom Elternprozeß an den Kindprozeß transferiert werden, treten beim Start des Kindprozeß sofort Seitenfehler auf. Diese Seitenfehler veranlassen den External-Memory-Manager (XMM) die entsprechenden Datenseiten vom Elternprozeß zum Kindprozeß zu transferieren und die Zugriffsrechte auf die übertragenen Seiten beim Elternprozeß zu löschen. Das Nachladen einer einzelnen Datenseite (8 KB) benötigt im Durchschnitt zwischen 3 und 5 Millisekunden.
Diese Seitennachladestrategie hat den Vorteil, daß nur das sogenannte Working-Set eines Prozesses transferiert wird. Da das Working-Set eines Prozesses häufig deutlich kleiner ist als sein tatsächlich belegter Hauptspeicher, kann durch diese Strategie die Freezing Time deutlich verringert werden. Diese Seitennachladestrategie hat jedoch auch Nachteile. So verursacht z.B. auch ein bereits migrierter Prozeß noch durch das Nachladen der Hauptspeicherseiten Last auf dem Quellknoten der Migration. Außerdem bedeutet die Generierung eines neuen Unix-Prozesses in diesem Zusammenhang, daß der verwendete Lasttransfermechanismus keine vollständige Prozeßmigration darstellt, da sich durch diesen Mechanismus z.B. die Prozeßnummer ändert. D.h. basierend auf diesem Lasttransfermechanismus können somit sämtliche Betriebssystemdienste nicht unterstützt werden, die als Aufrufparameter die Prozeßkennung benötigen, z.B. Unix-Signals (*interrupt-basierte Ereignisverarbeitung*). Für die betrachtete Anwendungsklasse stellt dies jedoch keine wesentliche Einschränkung dar.

4.5 Eignung einer Mikrokernelarchitektur für die Lastbalancierung

In diesem Abschnitt wird auf die Eignung der Mach-Mikrokernel-Architektur für die dynamische Lastbalancierung eingegangen. Neben den Erfahrungen, die im Rahmen dieser Arbeit gemacht wurden, gehen auch die Erkenntnisse aus der Arbeit von Milojicic [Milo94] in diese Beurteilung ein.
In der Arbeit von Milojicic wurde insbesondere der Aufwand für die Realisierung bzw. Integration eines Migrationsdienstes in den Mach-Mikrokernel untersucht. Dazu wurden zwei Migrationsserver auf Anwendungsebene und ein Migrationsserver im Mikrokernel realisiert. Für die beiden Migrationsserver auf Anwendungsebene mußten dabei nur 300 Programmzeilen im Mach-Quelltext verändert werden.
Es muß in diesem Zusammenhang jedoch daraufhin gewiesen werden, daß in der erwähnten Arbeit nur der Aufwand für die Realisierung der Migration eines Mach-Prozesses untersucht wurde, nicht jedoch die Realisierung einer Prozeßmigration auf Unix-Ebene, die auch Modifikationen des Betriebssystemservers erfordert hätte. Besonders hilfreich bei der Realisierung der Migrationsserver

hat sich hierbei das Nachrichtenkonzept des Mach-Kernels (NORMA IPC) und der External-Memory-Manager (XMM) erwiesen. Der Mach-NORMA IPC ist in Kombination mit der objektorientierten Struktur des Mikrokernels besonders bei der Realisierung der anwendungstransparenten Migration hilfreich, da der Informationsaustausch zwischen den verschiedenen Objekten (Speicherobjekte, Threads, Prozesse, etc.) über expliziten Nachrichtenaustausch erfolgt und die Adressierung der Objekte über ihre Portnummer und nicht über ihre aktuelle Lokation im System erfolgt. Die Umsetzung von Portnummer zur aktuellen Lokation geschieht in transparenter Weise durch den NORMA IPC. Die Funktionalität des XMM, Speicherobjekte zwischen mehreren Prozessen, die sich nicht notwendigerweise auf dem selben Rechenknoten befinden müssen, zugreifbar zu machen, ist besonders bei der Übertragung des Speicherkontextes von Vorteil.
Neben diesen beiden unbestreitbaren Vorteilen des Mach-Mikrokernels für die dynamische Lastbalancierung weist der Mach-Mikrokernel jedoch einige zum Teil erhebliche Nachteile auf:

- Nicht alle Prozeßzustände sind für den Anwender zugänglich (z.B. die Kernelports der Prozesse und Threads).
- Nicht alle für die dynamische Lastbalancierung relevanten Lastinformationen, die im Mikrokernel prinzipiell verfügbar sind, werden erfaßt bzw. für den Anwender verfügbar gemacht. Z.B. sammelt der Mikrokernel Informationen über das Working-Set eines Knotens, jedoch nicht über das Working-Set eines Prozesses. Ebenfalls sind die Lastinformationen bezüglich des IPC nicht ausreichend.
- Die Prozeßmigration als Lasttransfermechanismus reicht nicht bei allen Anwendungen aus. Speziell bei Anwendungen, die intensiven Gebrauch der Funktionalität des Betriebssystemservers machen, muß auch die Prozeßabstraktion transferiert werden. Beim OSF1/AD ergibt sich etwa das Problem, daß die Prozeßzustände über drei Stellen verteilt im System vorliegen. Im Mikrokernel liegen alle Mach-prozeßbezogenen Zustände, im Emulationsthread und dem Betriebssystemserver liegen die Unix-prozeßbezogenen Zustände. Aufgrund dieser verteilten Zustände ist die Unix-Prozeßmigration wesentlich aufwendiger zu realisieren als eine Mach-Prozeßmigration.
- Viele für die dynamische Lastbalancierung relevanten Lastinformationen sind nur im jeweiligen Betriebssystemserver verfügbar. Aufgrund seiner Beschränkung auf die Basismechanismen Prozesse, Threads, Ports, Messages und Speicherobjekte verfügt der Mikrokernel z.B. über keinerlei Information bezüglich der Ein-/Ausgabeaktivitäten eines Unix-Prozesses. Diese Information ist nur über den Betriebssystemserver verfügbar, wobei ein Betriebssystemserveraufruf um den Faktor 3 bis 4 teurer ist als ein vergleichbarer Mikrokernelaufruf.

Insgesamt kann allerdings die Schlußfolgerung gezogen werden, daß der Mach-Mikrokernel eine geeignete Plattform für die Realisierung einer dynamischen Lastbalancierungsumgebung darstellt. Insbesondere die zugrundeliegenden Konzepte für den Nachrichtenaustausch zwischen den Prozessen und die flexiblen Zugriffsmöglichkeiten auf Speicherobjekte erleichtern die Realisierung eines anwendungstransparenten Lasttransfersmechanismuses.

Kapitel 5

PaLaBer - eine hierarchische Lastbalancierungsumgebung

In diesem Kapitel wird die hierarchische Lastbalancierungsumgebung PaLaBer [Poll95][PoReWa97] vorgestellt. Hierbei handelt es sich um einen neuen Lastbalancierungsansatz, der auf seine Leistungsfähigkeit in Kapitel 6 untersucht wird. Da für die Untersuchungen im Zusammenhang mit den Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung in Kapitel 7 ein tieferes Verständnis für die Arbeitsweise des Systems hilfreich ist, wird in diesem Kapitel die Lastbalancierungsumgebung ausführlich beschrieben.
Im Vergleich zu Kapitel 2, in dem verschiedene Lastbalancierungsstrategien, sowie die an sie gestellten Anforderungen vorgestellt wurden, wird in diesem Kapitel eine Lastbalancierungsumgebung vorgestellt. Eine Lastbalancierungsumgebung besteht dabei aus einer statischen Lastbalancierungsstrategie zur initialien Plazierung der Arbeitslasteinheiten, einer dynamischen Lastbalancierungsstrategie, einem Kommunikationsmodul, einem Programmiermodell sowie Analysewerkzeugen (vgl. dazu Kapitel 6) zur Unterstützung des Entwicklungsprozesses.
Im ersten Abschnitt wird auf die prinzipiellen Anforderungen an eine dynamische Lastbalancierungsumgebung eingegangen, bevor dann die Arbeitsweise der Lastbalancierungsumgebung PaLaBer, die verwendeten Protokolle und die Umsetzung der Informationsebenen im Detail beschrieben werden.

5.1 Anforderungen an eine dynamische Lastbalancierungsumgebung

Ziel dieses Abschnittes ist es aufzuzeigen, welche Aufgaben von einer dynamischen Lastbalancierungsumgebung prinzipiell unterstützt werden können und welche Aufgaben davon von der hierarischen Lastbalancierungsumgebung PaLaBer unterstützt werden.
Das Lastverwaltungsproblem läßt sich nach [Heis94] in die folgenden fünf Teilprobleme aufgliedern, wobei der Autor implizit davon ausgeht, daß das parallele System im Space-Sharing Modus (vgl. Kapitel 1) betrieben wird.

- **Aufteilungsproblem**
  Das Aufteilungsproblem wird auch als quantitatives Partitionierungsproblem bezeichnet. Das

Aufteilungsproblem beschäftigt sich mit der Frage: Wie viele Prozessoren sollen einer parallelen Anwendung für ihre Abarbeitung zugeteilt werden?

- **Prozessorverwaltungsproblem**
  Das Prozessorverwaltungsproblem, das auch als qualitatives Partitionierungsproblem bezeichnet wird, beantwortet im Anschluß an das Aufteilungsproblem die Frage: Welche Prozessoren sollen der parallelen Anwendung zugeteilt werden?
- **Einbettungsproblem**
  Während sich die beiden vorangegangenen Probleme mit der Zuordnung von kompletten Programmen zu Prozessormengen beschäftigt haben, beschäftigt sich das Einbettungsproblem mit der Zuordnung von Prozessen zu Prozessoren. Dabei geht das Einbettungsproblem von einer injektiven Abbildung von Prozessen auf Prozessoren aus, d.h. die Anzahl der freien Prozessoren muß mindestens so groß sein wie die Anzahl der Prozesse der parallelen Anwendung.
  Das Einbettungsproblem hat somit die Aufgabe, unter Berücksichtigung der Kommunikationsbeziehungen bzw. Datenaffinitäten etc., eine möglichst gute Abbildung der Prozesse auf die verfügbare Hardware zu ermitteln.
- **Kontraktionsproblem**
  Beim Kontraktionsproblem, einer Verallgemeinerung des Einbettungsproblems, wird die Einschränkung der injektiven Abbildung aufgegeben. Dementsprechend ist die Lösung des Kontraktionsproblems ein zweistufiger Vorgang. Im ersten Schritt werden Prozeßcluster gebildet, wobei die Anzahl der Prozeßcluster der Anzahl der freien Prozessoren entspricht, und im zweiten Schritt findet dann wieder eine injektive Abbildung von Prozeßclustern zu Prozessoren statt.
- **Migrationsproblem**
  Die vorangegangenen vier Teilprobleme der Lastverwaltung befaßten sich mit der statischen Zuordnung von Programmen bzw. Prozessen zu Prozessoren. Die Zielsetzung des Migrationsproblems hingegen ist die dynamische Zuordnung von Prozessen zu Prozessoren. Eine dynamische Zuordnung wird erforderlich, falls sich zur Laufzeit der parallelen Anwendung die Anzahl der Prozesse ändert und/oder das Laufzeitverhalten der Prozesse signifikanten Veränderungen unterliegt.
  Wichtig in diesem Zusammenhang ist, daß das Migrationsproblem nicht losgelöst von den vorangegangenen Problemen gesehen werden darf, da eine möglichst gute initiale Plazierung der Anwendungsprozesse einen entscheidenden Einfluß auf die Lösung des Migrationsproblems ausübt.

Vier der fünf Teilprobleme des Lastverwaltungsproblems werden von der hierarchischen Lastbalancierungsumgebung PaLaBer unterstützt. Im einzelnen werden das Prozessorverwaltungsproblem, das Einbettungsproblem, das Kontraktionsproblem und das Migrationsproblem von der PaLaBer-Umgebung unterstützt. Lediglich das Aufteilungsproblem muß vom Anwender vor dem Start der Anwendung gelöst werden. Allerdings wird er auch bei diesem Problem von der Lastbalancierungsumgebung insofern unterstützt, als der Anwender als Programmier- und Ablaufschnittstelle eine virtuelle Maschine zur Verfügung gestellt bekommt. „Virtuelle Maschine“ bedeutet in diesem Zusammenhang eine Maschine, die über mehr Knoten verfügt als das tatsächliche Zielsystem. Dadurch hat der Anwender die Freiheit, eine Anzahl von Prozessoren zu wählen (Allokation einer virtuellen Partition), die optimal für seine spezifische Anwendung ist. Die Abbildung der virtuellen Knoten auf die real vorhandenen erfolgt dabei transparent für den Anwendung durch die Lastbalancierungsumgebung.

Im Laufe der nachfolgenden Abschnitte wird nun u.a. erläutert, wie die PaLaBer-Umgebung diese Teilprobleme des allgemeinen Lastverwaltungsproblems unterstützt.

5.2 Aufbau und Arbeitsweise der Lastbalancierungsumgebung

In diesem Abschnitt werden der Aufbau und die prinzipielle Arbeitsweise der PaLaBer-Umgebung beschrieben, bevor in den nachfolgenden Abschnitte detailliert auf die einzelnen Komponenten und Protokolle eingegangen wird. Dieser Abschnitt soll somit als roter Faden für die nachfolgenden Detailbeschreibungen dienen.
In Kapitel 2, in dem eine Übersicht über die verschiedensten Lastbalancierungsansätze vorgestellt wurde, wurde aufgezeigt, daß die Stärke der dezentralen Lastbalancierungsstrategien ihre Skalierbarkeit ist, während die zentralen Ansätze durch die Akkumulation aller entscheidungsrelevanter Informationen in einer Instanz - zumindest theoretisch - in der Lage sind, optimale Entscheidungen zu treffen. Bei einem hierarchischen Ansatz zur Lösung der Lastverwaltungsaufgabe wird nun versucht, die Stärken der dezentralen sowie der zentralen Ansätze zu vereinigen, ohne deren Schwächen mit zu übernehmen. Dazu wird versucht, möglichst viele rechenzeitintensiven Protokolle in den unteren Stufen der Hierarchie auszuführen, während in den oberen Schichten der Hierarchie die Möglichkeit besteht, Enscheidungen basierend auf komplexeren, akkumulierter Informationen zu treffen.
Im Vergleich zu vielen in der Literatur beschriebenen hierarchischen Lastbalancierungsansätzen, die häufig nur eine zweistufige Struktur aufweisen, verwendet das PaLaBer-System eine vollständige Baumstruktur (siehe Abbildung 19 (a)), bestehend aus einem Wurzelknoten, einer oder mehreren inneren Komponenten und mehreren Blattkomponenten[1]. Jeder Knoten des Baumes besteht dabei aus zwei Hauptkomponenten (siehe Abbildung 19(b)): einer Lastbalancierungskomponente und einer Komponente zur Realisierung der ortstransparenten Kommunikation, die in einem der nachfolgenden Abschnitte ausführlich beschrieben wird. Die Interaktion der Anwendungsprozesse mit der Lastbalancierungshierarchie erfolgt mittels einer Anwendungsbibliothek (PaLib), die an jeden Anwendungsprozeß gebunden wird. Diese Anwendungsbibliothek enthält sowohl Routinen für die Kommunikation als auch für die Prozeßverwaltung und wird in Abschnitt 5.9 und Anhang B ausführlich beschrieben. Bevor in den nachfolgenden Unterabschnitten die Aufgaben der einzelenen Komponenten im Baum skizziert werden, wird zuerst ein typischer Anwendungslauf kurz beschrieben.
Die parallelen Anwendungen gelangen über die Wurzelkomponente in das System. Dazu beobachtet der Wurzelknoten permanent das System. Sobald die Auslastung der einzelnen Bearbeitungsknoten einen gewissen Schwellwert unterschreitet, versucht die Wurzel diese Unterlast durch das Starten einer neuen parallelen Anwendung auszugleichen. Da die Wurzel, wie in den nachfolgenden Abschnitten noch ausführlich dargestellt wird, ihre Entscheidungen basierend auf nicht aktuellen Lastinformationen treffen muß, startet sie die Anwendungsprozesse nicht direkt, sondern übergibt lediglich die Prozeßbeschreibungen an ihren untergeordnenten Knoten.
Die inneren Komponenten, die innerhalb ihres Teilbaumes als zentrale Instanzen agieren, reichen analog zur Wurzel die Prozeßbeschreibungen, unter Berücksichtigung der aktuellen Lastsituation in den Teilbäumen, an ihre untergeordneten Komponenten weiter.
Der eigentliche Start der Anwendungsprozesse erfolgt erst in den Blattkomponenten der Hierarchie. Die Blattkomponenten entscheiden dabei autonom darüber, ob sie einen Anwendungsprozeß sofort starten oder für eine spätere Migration zwischenspeichern. Die Hauptaufgabe der Blattkomponenten ist jedoch die Durchführung der Migrationsprotokolle. Dabei werden sowohl zwischengespeicherte

1. Für eine formale, graphentheoretische Definiton eines korrekten Lastbalancierungsbaumes im Sinne des PaLaBer-Systems wird auf den Anhang A verwiesen.

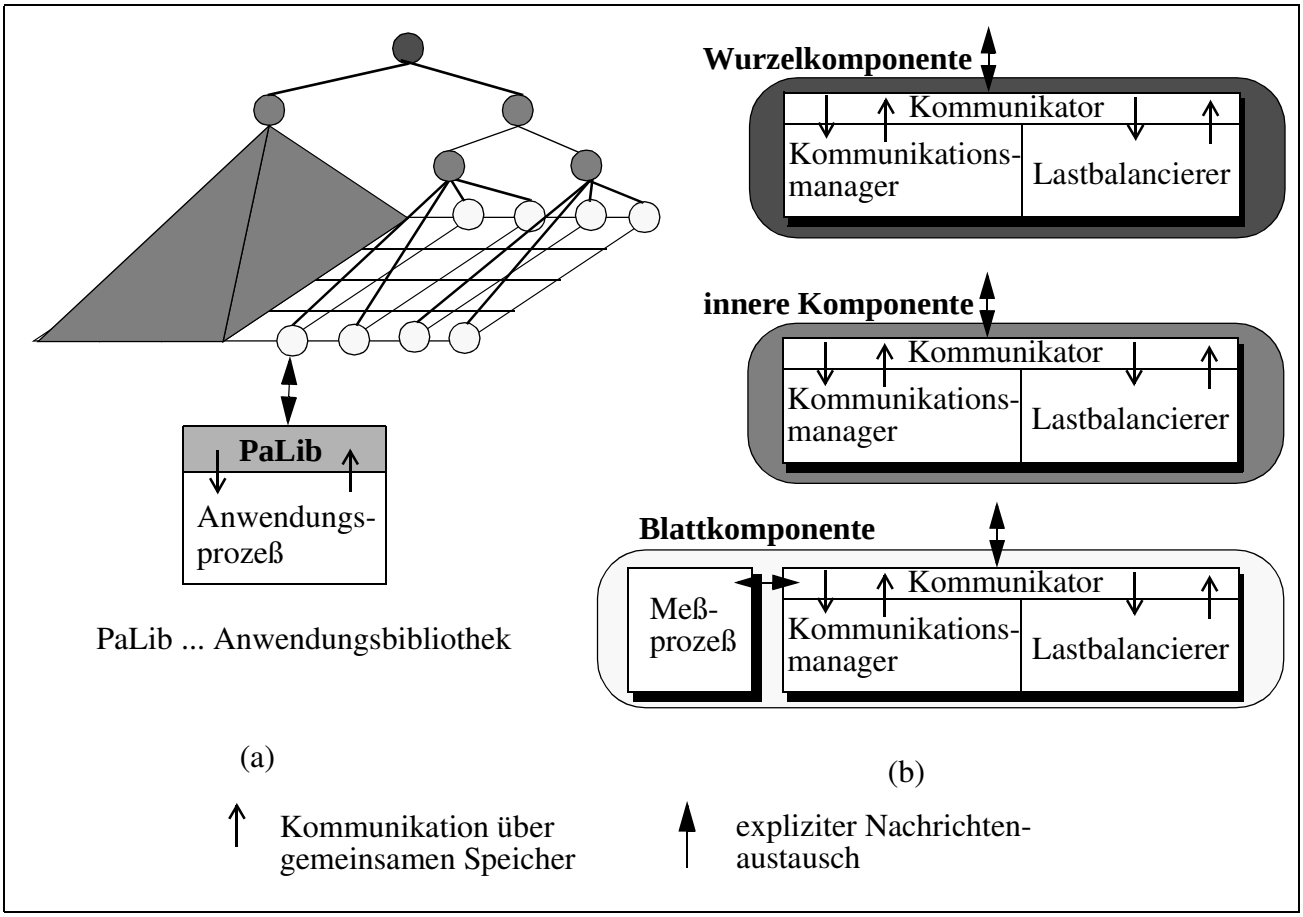


Abbildung 19: Aufbau der hierarchischen Lastbalancierungsumgebung.

Prozeßbeschreibungen als auch bereits gestartete Anwendungsprozesse migriert.
Anmerkung: Die eigentliche Kommunikation zwischen den Knoten des Baumes sowie zwischen den Komponenten eines Knotens erfolgt über den in Abbildung 19 (b) dargestellten Kommunikator, der die Basisfunktionalitäten zum Senden und Empfangen von Nachrichten bereitstellt.

5.2.1 Aufgaben der Wurzelkomponente

Die Wurzelkomponente ist die einzige zentrale Instanz im gesamten System. Da jedoch alle rechenzeitintesiven Protokollteile in den unteren Schichten der Hierarchie abgearbeitet werden, stellt sie keinen Engpaß bzgl. der Skalierbarkeit des Systems dar. Die Wurzel hat im wesentlichen drei Aufgaben zu erfüllen:

1. Starten der parallelen Anwendungen
   Wie bereits erwähnt, beobachtet die Wurzelkomponente permanent das System. Fällt die Systemauslastung unter einen vorgegebenen, adaptiven Schwellwert, versucht die Wurzel eine neue Anwendung entsprechender Größe zu starten. Zur Aufteilung der Anwendungprozesse bzw. ihrer Prozeßbeschreibungen auf die verschiedenen Teilbäume des Systems verwendet sie einen einfachen Clusteringalgorithmus (siehe Abschnitt 5.5).
2. Berechnung der adaptiven Schwellwerte
   Jede Komponente im Lastbalancierungsbaum berechnet autonom ihren Lastzustand. Dazu vergleicht sie ihre Lastwerte mit vorgegebenen Schwellwerten. Da konstante Schwellwerte der hohen Dynamik eines großen parallelen/verteilten Systems nicht gerecht werden können,

werden im PaLaBer-System Schwellwerte verwendet, die sich aus der aktuellen Systemlast ableiten lassen. Aufgabe der Wurzel ist es, aus den Lastwerten im System unter Berücksichtigung gewisser Rahmenbedingungen (minimale Systemauslastung) diese adaptiven Schwellwerte zu berechnen (vgl. Abschnitt 5.4).

3. Protokollkoordination zwischen den Teilbäumen
   Um die inneren Komponenten möglichst einfach und somit effizient halten zu könen, übernimmt die Wurzelkomponente die Koordination der Protokollflüsse über Teilbaumgrenzen hinweg.

5.2.2 Aufgaben der inneren Komponenten

Die inneren Komponenten, von denen es in jeder Systemkonfiguration mindestens eine Instanz geben muß, sind für die eigentliche dynamische Lastbalancierung verantwortlich, d.h. die inneren Komponenten entscheiden, ob und zwischen welchen Knoten des System ein Lastausgleich stattfinden soll. Um den gewählten Lastbalancierungsansatz auch für sehr große, parallele und verteilte Systeme verwenden zu können, kann es beliebig viele Stufen von inneren Komponenten im Lastbalancierungsbaum geben. Dadurch wird verhindert, daß eine einzelne innere Komponente zum Systemengpaß wird. Untersuchungen haben jedoch ergeben (siehe Kapitel 6), daß eine innere Komponente problemlos bis zu 60 untergeordnete Komponenten verwalten kann. Um den Lastbalancierungsbaum an die jeweilige Hardware- und Anwendungscharakteristik anpassen zu können, muß der Baum nicht notwendigerweise ausbalanciert sein. Im wesentlichen haben die inneren Komponenten die folgenden drei Aufgaben:

1. Plazierung der Anwendungsprozesse in ihrem Teilbaum
   Startet die Wurzelkomponente eine neue parallele Anwendung, dann erfolgt dies durch die Weiterleitung der Prozeßbeschreibung an die inneren Komponenten. Die inneren Komponenten leiten dann ihrerseits diese Beschreibungen an ihre untergeordneten Komponenten weiter. Die Weiterleitung der Prozeßbeschreibung erfolgt dabei in zwei Schritten. Im ersten Schritt versuchen die inneren Komponenten sicherzustellen, daß jeder Blattkomponente des Systems mindestens ein Anwendungsprozeß zugeteilt ist (*Load Sharing*[1]). Im zweiten Schritt wird dann versucht, die Anzahl der Anwendungsprozesse pro Blattkomponente ausgeglichen zu halten. Da die gesamten Migrationsprotokolle in den Blattkomponenten ausgeführt werden, wird bei der Plazierungsstrategie in den inneren Komponenten in Kauf genommen, daß einzelne Blattkomponenten durch eine Zuweisung überbelastet werden.
2. Aufbereitung der Lastwerte in ihrem Teilbaum
   Um den Berechnungsaufwand in den höheren Stufen der Lastbalancierungsstrategie in Grenzen zu halten, bereitet jede Stufe die Lastwerte ihrer untergeordneten Komponenten auf und reicht nur die aggregierten Lastwerte zusammmen mit dem ermittelten Lastzustand des Teilbaumes an ihre übergeordnete Komponente weiter. In den nachfolgenden Abschnitten wird ausführlich auf die Fragen eingegangen, welche Informationen zur Berechnung des Lastzustandes verwendet werden bzw. wie aus den ermittelten Lastkenngrößen der Lastzustand ermittelt wird.

1. In dieser Arbeit wird der englische Begriff „Load Sharing“ anstelle der deutschen Bezeichnung „Lastverteilungsverfahren“ verwendet, da es sich beim englischen Begriff um eine allgemein akzeptierte Bezeichnung handelt.

3. Lokationsentscheidungen, die ihren Teilbaum betreffen
   Treten im Laufe der Bearbeitung Lastungleichgewichte auf, die die inneren Komponenten nicht durch die Zuweisung neuer Anwendungsprozesse ausgleichen können, versuchen sie, durch die Vermittlung von über- und unterbelasteten Blattkomponenten einen Lastausgleich zu erreichen. Die eigentliche Auswahlentscheidung, welche Anwendungsprozesse zwischen diesen Blattkomponenten ausgetauscht werden sollen, wird von den beteiligten Blattkomponenten getroffen.

5.2.3 Aufgaben der Blattkomponenten

Auf jedem Knoten des Systems auf dem Anwendungsprozesse laufen, muß eine Blattkomponente vorhanden sein, da die Blattkomponenten als „Ansprechpartner“ für die Anwendungsprozesse fungieren. Die Lastbalancierungshierarchie als solche ist für die Anwendungsprozesse transparent. Die Blattkomponenten sind im wesentlichen für vier Aufgaben verantwortlich:

1. Prozeßverwaltung[1]
   Bekommt eine Blattkomponente von ihrer übergeordneten Komponente die Beschreibung eines zu startenden Anwendungsprozesses, entscheidet sie autonom, ob sie den Prozeß unmittelbar startet oder zuerst zwischenspeichert. Dazu schätzt die Blattkomponente den zukünftigen Lastzustand ihres Knotens ab. Würde der Knoten durch den neuen Anwendungsprozeß nicht überlastet, startet sie den Prozeß.
2. Prozeßmigration
   Die wichtigste Aufgabe der Blattkomponenten ist die Abarbeitung der Migrationsprotokolle. Das PaLaBer-System unterstützt dabei sowohl die Migration von noch nicht gestarteten Prozessen als auch die Verschiebung von laufenden Anwendungsprozessen. Beide Protokolle werden in Abschnitt 5.7 ausführlich beschrieben.
3. Erfassung und Aufbereitung der Lastwerte auf den lokalen Knoten
   Zur Erfassung der Lastwerte auf den lokalen Knoten verwenden die Blattkomponenten einen eigenständigen Meßprozeß (siehe Abbildung 19 (b)), der in regelmäßigen Zeitabständen die knotenbezogenen Lastwerte erfaßt und an die Blattkomponente weiterleitet. Alle prozeßbezogenen Informationen werden von den Blattkomponenten selbständig bei Bedarf erfaßt.
4. Interaktion mit den Anwendungsprozessen
   Die Blattkomponenten fungieren als Ansprechpartner für die Anwendungsprozesse, d.h. jede Interaktion zwischen Anwendungsprozessen und dem PaLaBer-System erfolgt über die Blattkomponenten. Zu diesem Zweck wird an jeden Anwendungsprozeß die Anwendungsbibliothek PaLib gebunden (vgl. Abbildung 19 (a)). Diese Bibliothek enthählt sowohl Routinen für die Interprozeßkommunikation als auch für die Prozeßverwaltung. Immer dann, wenn ein Anwendungsprozeß eine dieser Routinen aufruft (z.B. um einen anderen Anwendungsprozeß zu starten), besteht die Möglichkeit der Interaktion zwischen dem Anwendungsprozeß und dem Lastbalancierer. Dies bedeutet auch, daß ein Anwendungsprozeß nur dann zur Migration aufgefordert werden kann, wenn er sich innerhalb einer PaLib-Routine befindet. Für eine weitergehende Beschreibung dieser Bibliothek siehe Abschnitt 5.9.

1. Unter den Begriff Prozeßverwaltung fallen alle Tätigkeiten, die im Zusammenhang mit dem Start bzw. der Terminierung eines Anwendungsprozesses erledigt werden müssen.

Basierend auf der in Kapitel 2 eingeführten Taxonomie für dynamische Lastbalancierunsstrategien läßt sich das PaLaBer-System wie in Tabelle 18 dargestellt einordnen. Auf die fünf Klassifikationskriterien wird in den nachfolgenden Abschnitten noch ausführlich eingegangen.

Informationsbasis	Transfer-entscheidung	Lokations-entscheidung	Auswahl-entscheidung	Migrationsraum
global	dezentral	dezentral	dezentral	global

Tabelle 18: Klassifikation des PaLaBer-Systems.

5.3 Informationsfluß und -aufbereitung

Ziel dieses Abschnittes ist es aufzuzeigen, *wer*, *wann*, *welche* Information zur Verfügung gestellt bekommt. Die Frage, *wie* aus diesen Informationen dann in den verschiedenen Stufen der Lastbalancierungshierarchie der tatsächliche Lastzustand ermittelt wird, ist Bestandteil des nächsten Abschnittes.
Bevor jedoch auf die drei oben gestellten Fragen eingegangen wird, muß die Frage nach der Last bzw. dem vom PaLaBer-System unterstützten Lastmodell beantwortet werden. Als Ausgangsbasis für die Definition des Lastbegriffes und des daraus abgeleiteten Lastmodells dient das in Kapitel 1 bereits vorgestellte Anwendungsmodell. Die formale Definition des vom PaLaBer-System unterstützten Anwendungsmodell lautet:

Definition 6: Das unterstützte Anwendungsmodell besteht aus einer Gruppe von nicht statischen TIG‘s, die konkurrent abgearbeitet werden. Dieses Anwendungsmodell wird mit *multiple non-static TIG* (MNTIG) bezeichnet.

Aufbauend auf diese Definition des Anwendungsmodells läßt sich in Anlehnung des Lastbegriffs aus dem Bereich der Wartesysteme die Last definieren als:

Definition 7: Last ist die Gesamtheit der zu einem Zeitpunkt t noch durch das Lastbalancierungssystem abzuarbeitenden non-static TIG‘s.

Nachdem nun definiert ist, was im Zusammenhang mit dem PaLaBer-System unter dem Begriff Last zu verstehen ist, kann die Definition des Lastmodells angegangen werden. Aufgabe des Lastmodells ist es dabei, die im System vorhandene Last zu modellieren und somit einem dynamischen Lastbalancierer bzw. dessen Lastbewertungskomponente zugänglich zu machen. Da das vom PaLaBer-System unterstützte Anwendungsmodell vergleichbar ist mit dem Anwendungsmodell des ENX-Systems (vgl. [Ludw93]) wird dessen Definition des Lastmodells übernommen, da diese für praktische Zwecke völlig ausreichend ist.

Definition 8: Als Lastmodell bezeichnen wir die Menge der zur Lastbewertung herangezogenen Meßgrößen mit ihren relativen Gewichtungen zueinander und der Art ihrer Verknüpfung.

Nach diesem kurzen Ausflug in die formalen Definitionen, zurück zu den Fragen wer, wann, welche Information im Lastbalancierungsbaum bekommt. In Kapitel 2 wurde bereits darauf hingewiesen, daß bei völlig dezentralen Lastbalancierungsansätzen die Informationsverbreitung eine kritische Leistungsgröße darstellt, während bei zentralen Ansätzen die Informationssammlung und -aufbereitung einen entscheidenden Einfluß auf die gesamte Systemleistung ausübt. Beim PaLaBer-System ergibt sich der Informationsfluß und somit die Frage wer welche Information bekommt aus der baumförmi-

gen Struktur. Jede Komponente tauscht nur mit ihrer übergeordneten Komponente und ihrer untergeordneten Komponenten Informationen aus, wobei auf jeder Stufe der Hierarchie die Information für die nächste Stufe bereits vorverarbeitet wird. Prinzipiell muß bei den erfaßten Lastdaten zwischen knotenbezogenen Lastinformationen und prozeßbezogenen Informationen unterschieden werden.

5.3.1 Knotenbezogene Lastinformationen

Im Zusammenhang mit den knotenbezogenen Lastinformationen ist es wichtig, daß es im PaLaBer-System zwei gegenläufige Informationsströme gibt. Während die erfaßten Lastdaten von den Blattkomponenten über die inneren Komponenten zur Wurzel geschickt werden, wandern die daraus abgeleiteten adaptiven Schwellwerte, die als Entscheidungsparameter in die Berechnung des Lastzustandes eingehen (siehe nächster Abschnitt), von der Wurzel über die inneren Kompoenten zu den Blättern.
Die Blattkomponenten erfassen mit Hilfe des Meßprozesses in regelmäßigen Abständen alle knotenbezogenen Lastwerte. Im einzelnen erfassen die Blattkomponenten die folgenden knotenbezogenen Informationen:

- Anzahl der nicht auslagerbaren Hauptspeicherseiten
- Anzahl der belegten und auslagerbaren Hauptspeicherseiten
- Anzahl der freien Hauptspeicherseiten (L_{Mem})
- Anzahl der Hauptspeicher-Seitenfehler
- Anzahl der ein- bzw. ausgelagerten Hauptspeicherseiten im Meßintervall
- Prozessorleerlaufzeit im letzten Meßintervall (L_{CPU})
- Anzahl der bereits gestarteten Anwendungsprozesse (L_{Proc})
- Anzahl der zwischengespeicherten Anwendungsprozesse (L_{Queue})
- Anzahl der initiierten Migrationen (NS_{MIG})
- Anzahl der vorzeitig angekündigten Terminierungen (NS_{AT})[1]
- tatsächliche Länge des Meßintervalls

Nicht berücksichtigt werden die Ein-/Ausgabeaktivitäten sowie das Kommunikationsaufkommen auf dem jeweiligen Knoten. Die Ein-/Ausgabeaktivitäten auf dem jeweiligen Knoten werden nicht berücksichtigt, da der dynamische Lastbalancierer keine Einflußmöglichkeit auf diesen Ressourcenengpaß hat bzw. das in Kapitel 4 vorgestellte parallele Dateisystem (PFS) die zur Verfügung stehenden Ressourcen bereits nahezu optimal ausnutzt. Die Kommunikation, die in erster Linie für die Auswahl eines geeigneten Migrationskandidaten benötigt wird, wird in Verbindung mit den prozeßbezogenen Informationen behandelt.
Bevor nun eine Blattkomponente diese Lastinformationen an ihre übergeordnete Komponente weiterleitet, werden sämtliche intervallbezogenen Größen (z.B. Prozessorleerlaufzeit im letzten Meßintervall) auf ein 1-Sekunden-Intervall normiert. Diese Normierung ist notwendig, da die Länge des tatsächlichen Meßintervalls vom vorgegebenen Meßintervalls durch betriebssystembedingte Verzögerungen abweichen kann.
Bekommt eine innere Komponente die oben aufgeführen Lastinformationen von einer ihrer untergeordenten Komponenten zugeschickt, speichert sie diese „Rohdaten“ in einer Liste ab. Anschließend berechnet sie basierend auf den Lastinformationen ihrer Kinder erstens die

1. Auf diese Größe wird erst im Zusammenhang mit den höheren Informationsebenen eingegangen.

durchschnittlichen Lastwerte in ihrem Teilbaum und zweitens die maximale Prozessorleerlaufzeit aller Anwendungsknotens in ihrem Teilbaum. Ausgehend von diesen aufbereiteten Daten ermittelt die innere Komponente den Lastzustand ihres Teilbaumes. Sobald sie genau so viele Lastnachrichten bekommen hat wie sie untergeordnete Komponenten besitzt, übermittelt sie die aufbereiteten Lastdaten zusammen mit dem ermittelten Lastzustand an ihre übergeordnete Komponente. Da die inneren Komponenten nicht jede Laständerung im Teilbaum sofort an ihre übergeordnete Komponente weiterleiten, arbeiten sie bereits mit den neuen Lastwerten, bevor sie diese nach oben weiterleiten.
Kommen die Lastinformationen schließlich in der Wurzel des Lastbalancierungsbaumes an, berechnet die Wurzel die bereits mehrfach erwähnten adaptiven Schwellwerte, die anschließend über die inneren Komponenten an die Blattkomponenten weitergeleitet werden, sowie den Lastzustand des gesamten Systems.
Nachdem nun für die knotenbezogenen Lastinformationen aufgezeigt wurde, *wer welche* Informationen bekommt, wird jetzt auf die Frage eingegangen, *wann* wer welche Informtion bekommen soll. Bei den knotenbezogenen Informationen wird die Informationsverbreitung entscheidend durch die Länge des Meßintervalles bestimmt, d.h. der Zeitabstand zwischen dem der Meßprozeß die knotenbezogenen Lastinformationen sammelt und an seine Blattkomponente weiterleitet. Die Länge des Meßintervalls hat dabei entscheidenden Einfluß auf die Leistungsfähigkeit eines dynamischen Lastbalancierers. Ein großes Meßintervall verursacht wenig Systemzusatzlast, hat jedoch den Nachteil, daß der Lastbalancierer seine Entscheidungen aufgrund alter Lastinformationen treffen muß. Ein kleines Meßintervall hat hingegen den Vorteil, daß der Lastbalancierer seine Entscheidungen auf der Basis aktueller Informationen treffen kann. Allerdings erkauft er sich diesen Vorteil mit einer erheblichen Zusatzlast. Zudem bedeutet ein kleines Meßintervall, daß auch Systembelastungen die z.B. durch eine Lastverschiebung entstehen, erfaßt werden und die Knotenlast zum Teil erheblich verfälschen können. Vergleichsmessungen haben ergeben, daß für das PalaBer-System zusammen mit seinem unterstützten Anwendungsmodell ein Meßintervall von 2,7 Sekunden die besten Laufzeitergebnisse für die in Kapitel 6 vorgestellten Anwendungen erzielt.
Damit werden die besten Ergebnisse mit einer Intervallänge erzielt, die in etwa doppelt so lang ist, wie die durchschnittliche Zeit für eine Migration eines bereits laufenden Prozesses (Freezing-Time inkl. der Übertragung des Working-Sets). Dieses Ergebnis ist insofern nicht überraschend, da das Abtasttheorem von Shannon besagt, daß man ein Signal (in diesem Fall die permanenten Änderungen in der Systemlast) mit der doppelten Frequenz abtasten muß, die man noch erfassen möchte. Die Umkehrung des Abtasttheorems kann in Anlehnung dazu verwendet werden, die minimale Länge eines Meßintervalls zu bestimmen, da die Verfälschungen der Lastwerte durch die Lastverschiebungen nicht erfaßt werden sollen.

5.3.2 Prozeßbezogene Lastinformationen

Neben den knotenbezogenen Lastinformationen, die in regelmäßigen Zeitabständen erfaßt werden, ermitteln die Blattkomponenten bei Bedarf noch prozeßbezogene Lastinformationen über ihre Anwendungsprozesse. Damit ist auch bereits die Frage nach dem *wann* beantwortet. Prozeßbezogene Information wird nur bei Bedarf, d.h. beim Auftreten von Lastungleichgewichten erfaßt. Diese Informationen werden zur Auswahl eines geeigneten Migrationskandidaten sowie im Zusammenhang mit den höheren Informationsebenen benötigt. Im einzelnen erfassen die Blattkomponenten die folgenden Informationen über ihre Anwendungsprozesse:

- Gesamtlaufzeit des Anwendungsprozesses

- Konsumierte Prozessorzeit
- Anzahl der belegten Hauptspeicherseiten
- Kommunikationsbeziehungen des Anwendungsprozesses
  - Adressen der Kommunikationspartner
  - Anzahl der ausgetauschen Nachrichten

Diese Lastwerte werden, mit Ausnahme der Kommunikationsbeziehungen, nur in den Blattkomponenten gehalten. Lediglich die Kommunikationsbeziehungen der Anwendungsprozesse werden an die übergeordneten inneren Komponenten weitergeleitet, die diese Information in Verbindung mit ihrer Lokationsentscheidung verwenden.

5.4 Bestimmung des Lastzustandes im Lastbalancierungsbaum

Nachdem im vorigen Abschnitt die Fragen beantwortet wurden, *wer, wann, welche* Information zur Verfügung gestellt bekommt, ist es das Ziel dieses Abschnittes, zu klären, *wie* diese Informationen zur Ermittlung des Lastzustandes verwendet werden. Unabhängig davon, ob der Lastzustand für einen einzelnen Anwendungsknoten, einen Teilbaum oder für das gesamte System berechnet werden soll, unterscheidet das PaLaBer-System drei Lastzustände: *unterbelastet*, *normal belastet* und *überbelastet*.
Im Vergleich zu vielen anderen Ansätzen (siehe z.B. [Beck95][Ludw93]), bei denen durch entsprechende Normierung der Meßgrößen mit anschließender Gewichtung versucht wird, den Lastzustand mit Hilfe einer geschlossenen Formel zu berechnen, verwendet die PaLaBer-Umgebung dazu einen mehrstufigen Ansatz (siehe Abbildung 20). Dieser Ansatz finden in allen Stufen der Lastbalancierungshierachie Anwendung. In einem ersten Schritt werden die ermittelten Meßgrößen auf ressourcenbezogene Lastzustände abgebildet, so wird z.B. auf Basis der ermittelten Prozessorleerlaufzeit festgelegt, ob sich die Ressouce Prozessor im Lastzustand *unterbelastet*, *normal belastet* oder *überbelastet* befindet. Die so ermittelten ressourcenbezogenen Lastwerte dienen dann als Zugriffsindizes für die in Tabelle 19 dargestellte Lastmatrix, die als Entscheidungsmatrix zu verstehen ist. Der aus der Lastmatrix ermittelte Lastzustand $L_{General}$ wird anschließend mittels mehrerer Zusatzregeln auf den endgültigen Lastzustand abgebildet. Dieser Berechnungsvorgang wird in den nachfolgenden Unterabschnitten detailliert beschrieben (siehe Abbildung 20 und Tabelle 19).

5.4.1 Ermittlung der ressourcenbezogenen Lastzustände

Zur Ermittlung des Lastzustandes werden in einem ersten Schritt die von den Blattkomponenten erfaßten Lastwerte auf vier ressourcenbezogene Lastzustände (L_{CPU}, L_{Proc}, L_{Mem}, L_{Queue}) abgebildet. Dazu werden jedoch nicht alle von den Blattkomponenten ermittelten Lastwerte verwendet. So hat sich z.B. herausgestellt, daß die Anzahl der Hauptspeicher-Seitenfehler keine geeignete Meßgröße darstellt. Diese Größe ist ungeeignet, da nicht unterschieden wird, ob die Seitenfehler durch Anwendungsprozesse verursacht werden, die sich auf dem lokalen Knoten befinden oder durch einen bereits migrierten Anwendungsprozeß (vgl. Abschnitt 4.4). Tatsächlich Anwendung finden die folgenden Meßgrößen, die mittels der bereits mehrfach erwähnten adaptiven Schwellwerte (siehe nachfolgende Beschreibungen) auf die vier Lastzustände abgebildet werden:

- Anzahl der freien Hauptspeicherseiten (L_{Mem})
- Prozessorleerlaufzeit im letzten Meßintervall (L_{CPU})
- Anzahl der bereits gestarteten Anwendungsprozesse (L_{Proc})

- Anzahl der zwischengespeicherten Anwendungsprozesse (L_{Queue})

Die systemweit gültigen, adaptiven Schwellwerte werden in der Wurzelkomponente berechnet. Insgesamt werden zur Bestimmung des Lastzustandes vier Schwellwerte verwendet, für jeden oben aufgeführten Lastkennwert ein Schwellwert. Während die Schwellwerte für die Hauptspeicherauslastung und für die Anzahl der zwischengespeicherten Werte pro Komponente konstant sind, werden die Schwellwerte bzgl. der Prozessorauslastung bzw. bzgl. der Anzahl der bereits gestarteten Anwendungsprozesse dynamisch zur Laufzeit berechnet. Die Formeln zur Berechnung dieser beiden adaptiven Schwellwerte lautet wie folgt:

$threshold_{CPU}$ = min(Øprozentuale Leerlaufzeit, Grenzwert)

Grenzwert = 1%,[a] falls es keine leeren Blattkomponenten gibt.

Grenzwert = 10%, sonst.

$threshold_{Proc}$ = max($\lceil$ ØProzeßanzahl $\rceil$,1)

a. Die konstanten Werte für die Untergrenze der Prozessorauslastung wurden empirsch ermittelt.

Für beide Schwellwerte wurden untere Grenzen festgelegt, um eine minimale Systemauslastung garantieren zu können. Für den Schwellwert $threshold_{CPU}$ wurden zwei verschiedene untere Grenzen angegeben. Für den Fall, daß alle Blattkomponenten mindestens einen Anwendungsprozeß zugewiesen bekommen haben, wird eine Prozessorauslastung von 99% angestrebt. Damit der Lastbalancierer auf globale Lastungleichgewichte schneller reagieren kann, wird die angestrebte Prozessorauslastung für den Fall, daß es einzelne völlig unausgelastete Blattkomponenten gibt, auf 90% reduziert. Dadurch wird die Wahrscheinlichkeit erhöht, daß stärker belastete Knoten als überbelastet eingestuft werden.
Durch den Vergleich der Meßgrößen mit den vier Schwellwerten werden die vier Lastzustände ermittelt. Die Lastzustandsvariablen L_{Proc} und L_{CPU} können dabei die Werte *unterbelastet*, *normal belastet* oder *überbelastet* annehmen. Hingegen sind die Lastzustandsvariablen L_{Mem} und L_{Queue} nur zweiwertig und können die Werte *frei* und *nicht frei* bzw. *leer* und *nicht leer* annehmen.

5.4.2 Beschreibung der Lastmatrix

Die vier Lastzustände L_{Proc}, L_{CPU}, L_{Mem} und L_{Queue} dienen als Index für die Lastmatrix. Die Lastmatrix verknüpft die vier Lastzustände zum Lastzustand $L_{General}$, der mittels mehrerer Zusatzregeln auf den endgültigen Lastzustand L_{Final} abgebildet wird.
Die Lastmatrix zeichnet sich durch ihre Einfachheit und Anschaulichkeit aus. So ist es mit Hilfe der Lastmatrix in einfacher Weise möglich, sowohl Charakteristiken des Rechnersystems als auch der betrachteten Anwendungsklasse zu berücksichtigen, wie in der nachfolgenden Diskussion der 12 Regeln (siehe Tabelle 19) aufgezeigt wird.

Regel 1:

Von allen Ressourcen, die von der Lastbalancierungsumgebung PaLaBer verwaltet werden, ist der Prozessor die wichtigste Ressource. Ist die Prozessorressource überbelastet, ist der Lastzustand des Knotens bzw. des Teilbaums, unabhängig vom Lastzustand der restlichen Ressourcen, *überbelastet*.

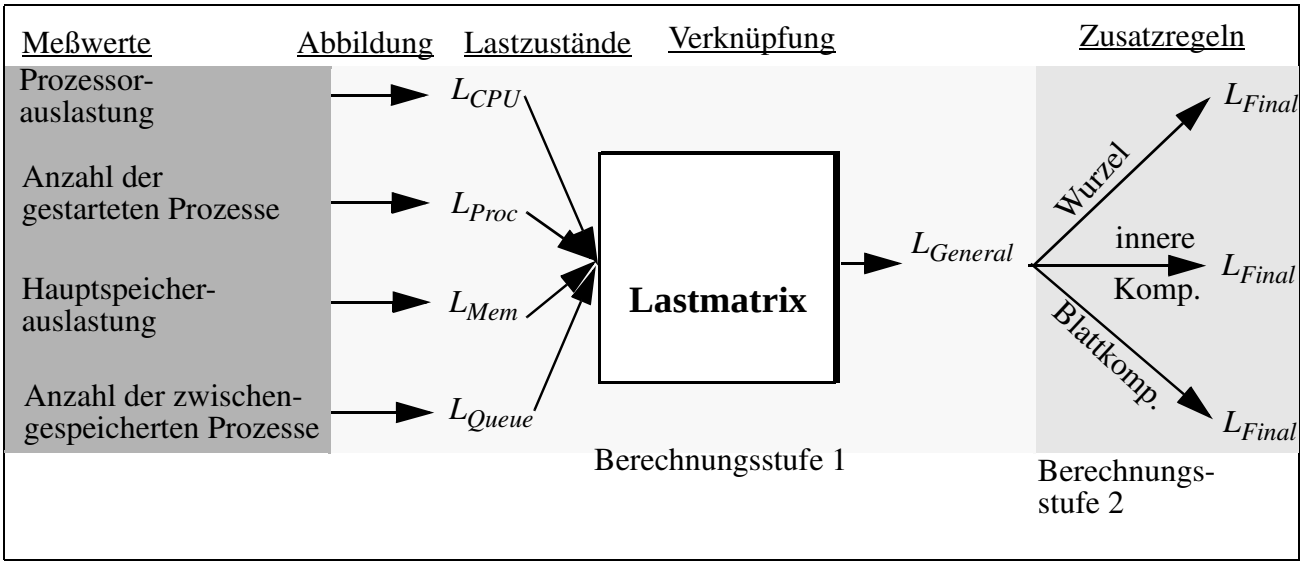


Abbildung 20: Schematischer Ablauf der Lastzustandsberechnung.

Regel	L_{CPU}	L_{Proc}	L_{Queue}	L_{Mem}	$L_{General}$
1	überbelastet				überbelastet
2	normal belastet	unterbelastet			normal belastet
3	normal belastet	normal belastet			normal belastet
4	normal belastet	überbelastet	leer		normal belastet
5	normal belastet	überbelastet	nicht leer		überbelastet
6	unterbelastet	unterbelastet	leer	frei	unterbelastet
7	unterbelastet	unterbelastet	nicht leer		normal belastet
8	unterbelastet	unterbelastet		nicht frei	normal belastet
9	unterbelastet	normal belastet	leer		unterbelastet
10	unterbelastet	normal belastet	nicht leer		normal belastet
11	unterbelastet	normal belastet		nicht frei	normal belastet
12	unterbelastet	überbelastet			normal belastet

Tabelle 19: Beschreibung der Lastmatrix

Regel 2:

Der Prozessor als wichtigste Ressource ist normal ausgelastet und keine der anderen Ressourcen stellt einen Engpaß dar, somit ist der Lastzustand insgesamt *normal belastet*.

Regel 3:

Begründung ist analog zu Regel 2.

Regel 4:

In dieser Konfiguration ist die Anzahl der Prozesse überdurchschnittlich hoch. Da der Prozessor jedoch normal ausgelastet ist, ist der gesamte Lastzustand als normal zu bezeichnen. Eine

solche Lastsituation kann z.B. dadurch eintreten, daß sich die Prozesse häufig global synchronisieren und somit wenig Prozessorzeit in Anspruch nehmen.

Regel 5:
In dieser Lastsituation ist der Prozessor normal ausgelastet und es sind noch zwischengespeicherte Prozesse vorhanden. Da ein Starten der Prozesse die wichtigeste Ressource - den Prozessor - mit großer Wahrscheinlichkeit überbelasten würden, wird als Lastzustand *überbelastet* angenommen.

Regel 6:
Sämtliche Ressourcen sind unterbelastet, der Lastzustand ist folgerichtig *unterbelastet*.

Regel 7:
In dieser Lastzustandskombination ist sowohl die Prozessorressource unterbelastet als auch die Anzahl der Anwendungsprozesse unter dem Systemdurchschnitt. Nachdem die Komponente jedoch noch über zwischengespeicherte Anwendungsprozesse verfügt und somit selbständig auf die Unterlast reagieren kann, ist der Lastzustand der Komponente *normal belastet*.

Regel 8:
In dieser Lastzustandkombination geht Information über das Zielsystem (vgl. Kapitel 4) ein. Obwohl das Zielsystem über eine virtuelle Speicherverwaltung verfügt, muß aus Leistungsgründen vermieden werden, daß es zu einer größeren Anzahl von Seitentransfers kommt. D.h. sobald die Hauptspeicherauslastung über dem Schwellwert liegt, ist der minimale Lastzustand, den die Komponenten noch annehmen können, *normal belastet*.

Regel 9:
Der Prozessor ist nicht ausgelastet und da keine noch nicht gestarteten Prozesse vorhanden sind, wird der Lastzustand *unterbelastet* angenommen.

Regel 10:
Da noch nicht gestartete Prozesse vorhanden sind, kann die Unterbelastung des Prozessors selbständig behoben werden. Der Lastzustand *normal belastet* ist somit korrekt.

Regel 11:
Die Prozessorressource ist *unterbelastet*, da jedoch die Ressource Hauptspeicher an der kritischen oberen Marke angekommen ist, wird der Lastzustand *normal belastet* angenommen.

Regel 12:
In diese Lastzustandskombination gehen sowohl Informationen über das Zielsystem als auch über die unterstützte Anwendungschrakteristik ein. Bei dieser Lastzustandskombination befindet sich ein Knoten bzw. Teilbaum im Lastzustand *normal belastet,* obwohl die Prozessorressource *unterbelastet* ist, da auf dem Knoten bzw. Teilbaum überdurchschnittlich viele Anwendungsprozesse vorhanden sind. Für das Auftreten einer solchen Lastzustandskombination gibt es prinzipiell drei Gründe:

1. Durch eine ungünstige Plazierung der Anwendungsprozesse wird das Kommunikationssystem zum Engpaß.
2. Die Anwendungsprozesse haben grundsätzlich nur einen geringen Prozessorbedarf.
3. Die Anwendungsprozesse befinden sich gegenwärtig in einer Synchronisationsphase.

Aufgrund des leistungsfähigen Kommunikationssystems der Zielmaschine, die eine virtuelle Vollvermaschung anbietet, scheidet die erste Möglichkeit für diese Lastzustandskombination aus. Die zweite Möglichkeit scheidet aufgrund der Eigenschaften der betrachteten Anwendungen, die sehr prozessorintensiv sind, aus. Basierend auf diesen Eigenschaften kommt nur die dritte Möglichkeit in Betracht, weshalb der Lastbalancierer eines Knotens bzw. Teilbaums bei dieser Lastzustandskombination in den Zustand normal belastet wechselt.

5.4.3 Beschreibung der Zusatzregeln

Da in die Umsetzung der vier Lastzustände in der Lastmatrix keine spezifischen Eigenschaften der Blattkomponenten, inneren Komponenten oder der Wurzelkomponente eingehen, wird der Lastzustand $L_{General}$ mittels mehrerer Zusatzregeln, die diesen Spezifika Rechnung tragen, auf den endgültigen Lastzustand L_{Final} abgebildet.

Die Zusatzregeln für die Blattkomponenten lauten:

Regel 13:

War eine Blattkomponente innerhalb der letzten t_{Action} Sekunden an einer Lastbalancierungsaktion (Prozeßstart, Prozeßmigration) beteiligt, kann die Blattkomponente maximal den Lastzustand normal belastet annehmen. In Abhängigkeit der Lastbalancierungsaktion liegt die Zeitspanne t_{Action} zwischen 1,5 (Prozeßstart) und 2 Meßintervallen (Prozeßmigration). Es werden für den Prozeßstart bzw. Prozeßmigration unterschiedliche Zeitspannen verwendet, da sich ihr Einfluß auf den Lastzustand eines Knotens unterschiedlich schnell auswirkt.

Diese Zusatzregel verhindert, daß ein Knoten auf ein Lastungleichgewicht überreagiert. Die nachfolgenden Regeln 14 und 15 werden in erster Linie im Hinblick auf die höheren Informationsebenen benötigt (vgl. Abschnitt 5.10). Speziell die Informationsebene 3 manipuliert den Lastkennwert für die Anzahl der laufenden Prozesse im voraus. Diese Manipulation kann unter Umständen zu inkonsistenten Lastzustandskombinationen führen, die durch die zwei folgenden Zusatzregeln bereinigt werden.

Regel 14:

Befindet sich kein Anwendungsprozeß auf einem Knoten, ist der Lastzustand des Knotens *unterbelastet*.

Regel 15:

Befindet sich genau ein Anwendungsprozeß auf einem Knoten, ist der maximale Lastzustand des Knotens *normal belastet*.

Die Zusatzregeln für die inneren Komponenten lauten:

Regel 16:

Ist die Anzahl der unterbelasteten Blattkomponenten im Teilbaum kleiner als die Anzahl der überbelasteten Blattkomponenten, ist der gesamte Teilbaum und somit die innere Komponente im Lastzustand *überbelastet*. In diesem Fall ist die Komponente nicht in der Lage, das Lastungleichgewicht selbständig auszugleichen.

Regel 17:

Ist die Anzahl der Anwendungsprozesse kleiner als die Anzahl der Blattkomponenten im Teilbaum, ist der Lastzustand des Teilbaums *unterbelastet*.

Regel 18:

Entspricht die Anzahl der Anwendungsprozesse der Anzahl der Blattkomponenten im Teilbaum, befindet sich die innere Komponente maximal im Lastzustand *normal belastet*.

Die Zusatzregeln für die Wurzelkomponente entsprechen den Regeln 17 und 18 der inneren Komponenten.

5.5 Starten einer parallelen Anwendung

Das Starten einer parallelen Anwendung verläuft in zwei Phasen. In der ersten Phase wird von der Wurzelkomponente autonom entschieden, ob eine neue Anwendung gestartet werden soll und wenn ja, welche. In der zweiten Phase werden dann die Prozesse der ausgewählten Anwendung, unter Berücksichtigung der Lastsituation im System, auf die Teilbäume der Lastbalancierungshierarchie verteilt.
Wie in Abschnitt 5.3 ausführlich beschrieben, bekommt die Wurzelkomponente in regelmäßigen Zeitabständen von ihren untergeordneten Komponenten Lastinformationsnachrichten. Ausgehend von diesen Lastinformationen berechnet die Wurzel die Fehlmenge der Anwendungsprozesse. Die Prozeßfehlmenge ergibt sich dabei wie folgt.

$$\text{Prozeßfehlmenge} = N_{IDEAL} - N_{RUN} - N_{QUEUE} + N_{AT}$$

mit	N_{IDEAL}	...	Knotenanzahl der virtuellen Maschine
	N_{RUN}	...	Anzahl der laufenden Anwendungsprozesse
	N_{QUEUE}	...	Anzahl der zwischengespeicherten Anwendungsprozesse
	N_{AT}	...	Anzahl der Anwendungsprozesse, die ihre Terminierung bereits vorzeitig angekündigt haben (vgl. Abschnitt 5.10)

Ist die Prozeßfehlmenge größer Null, versucht die Wurzelkomponente, durch das Starten einer neuen parallelen Anwendung die Prozeßfehlmenge zu verringern. Dabei achtet die Komponente jedoch darauf, daß die Prozeßfehlmenge nicht negativ wird.
In der zweiten Phase werden die Prozesse der ausgewählten Anwendung auf die Teilbäume der Lastbalancierungshierarchie verteilt. Das PaLaBer-System unterstützt dabei zwei verschiedene Verteilungsstrategien.
Bei der ersten Verteilungsstrategie, die im Hinblick auf die Untersuchungen in Kapitel 7 benötigt wird, werden die Anwendungsprozesse, unter Berücksichtigung der Prozeßfehlmenge in den Teilbäumen, zufällig, d.h. ohne Berücksichtigung der statischen Plazierungsinformation, auf die Teilbäume verteilt.
Die zweite Verteilungsstrategie macht Gebrauch von der statischen Plazierungsinformation, die der Anwender durch Plazierung der Prozesse innerhalb der virtuellen Partition (vgl. Abschitt 5.9) dem System mitteilt. Dabei wird versucht, möglichst große Cluster aus der virtuellen Partition zu schneiden und jeweils einem Teilbaum zuzuweisen (siehe Abbildung 21). Reichen die Prozesse der parallelen Anwendung nicht aus, um die Prozeßfehlmenge im System vollständig auszugleichen, versucht der Lastbalancierer in der Wurzelkomponente zuerst sicherzustellen, daß jeder Teilbaum mindestens so viele Anwendungsprozesse wie Blattkomponenten besitzt (*Load Sharing*). Die verbleibenden

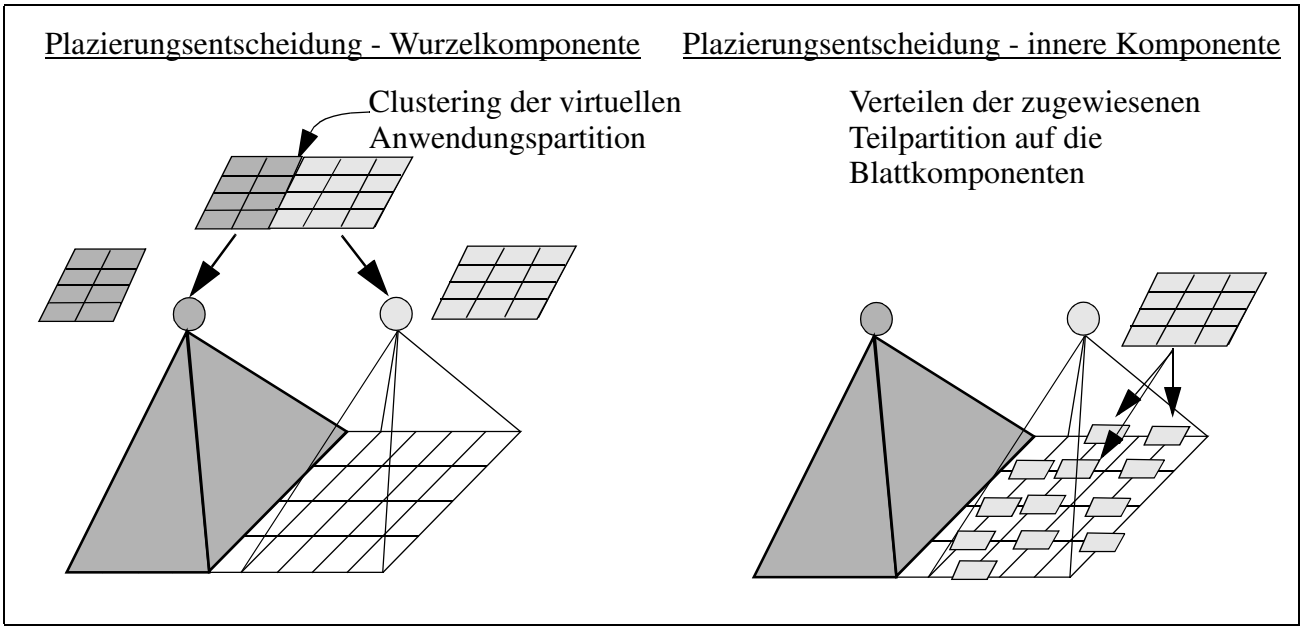


Abbildung 21: Initiale Prozeßplazierung - Clusterverfahren.

Anwendungsprozesse werden dann anschließend so verteilt, daß die Auslastung (Anzahl der Anwendungsprozesse) der Teilbäume annähernd gleich ist (*Lastbalancierung*).

Zur Bestimmmung der Prozeßfehlmenge für einen Teilbaum wird die nachfolgende Formel verwendet.

$$\text{Prozeßfehlmenge im Teilbaum} = NS_{IDEAL} - NS_{RUN} - NS_{QUEUE} - NS_{MIG} + NS_{AT}$$

mit

NS_{IDEAL}	...	Gewünschte Anzahl von Anwendungsprozessen im Teilbaum (Diese Größe entspricht der Anzahl der virtuellen Knoten im Teilbaum)
NS_{RUN}	...	Anzahl der laufenden Anwendungsprozesse im Teilbaum
NS_{QUEUE}	...	Anzahl der zwischengespeicherten Anwendungsprozesse im Teilbaum
NS_{MIG}	...	Anzahl der Migrationen in den Teilbaum (Inkrementierung der Zählervariablen) bzw. aus dem Teilbaum heraus (Dekrementierung der Zählervariablen)
NS_{AT}	...	Anzahl der Anwendungsprozesse, die ihre Terminierung bereits vorzeitig angekündigt haben

Bekommt eine innere Komponente von ihrer übergeordneten Komponente eine Prozeßgruppe zum Start zugewiesen, läuft bei ihr ein analoger Berechnungsvorgang ab. Mit Hilfe der obigen Formel wird für jeden Teilbaum die Prozeßfehlmenge ermittelt und in zwei Schritten (*Load Sharing*, *Lastbalancierung*) die Anzahl der Prozesse pro Teilbaum bestimmt. Zur Verteilung der Anwendungsprozesse auf die Teilbäume stehen ebenfalls zwei Verteilungsstrategien (zufällige Verteilung, Clusterverfahren) zur Verfügung.

5.6 Auswahl von Quell- und Zielknoten

Die Hauptaufgabe der inneren Komponenten ist die Entscheidung, zwischen welchen Blattkomponenten im System ein Lasttransfer stattfinden soll. Diese Lastbalancierungsentscheidung wurde in Kapitel 2 als Lokationsentscheidung bezeichnet. Auf die anschließende Auswahlentschei-

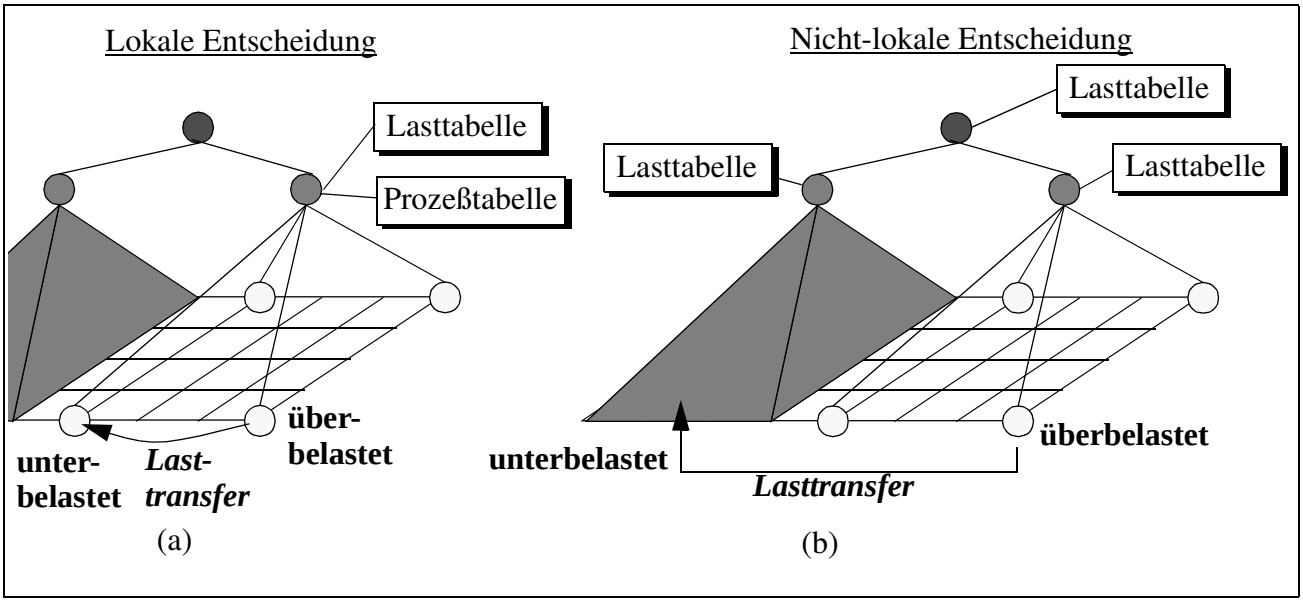


Abbildung 22: Auswahl von Quell- und Zielknoten.

dung, die vollständig in den Blattkomponenten durchgeführt wird, haben die inneren Komponenten keinen Einfluß mehr.

Bei der Durchführung der Lokationsentscheidung muß unterschieden werden (siehe siehe Abbildung 22), ob es sich um eine rein lokale Entscheidung handelt (beide in Frage kommenden Blattkomponenten sind Nachfolger derselben inneren Komponente) oder um eine nicht-lokale Entscheidung (der Lasttransfer erfolgt über Teilbaumgrenzen hinweg).

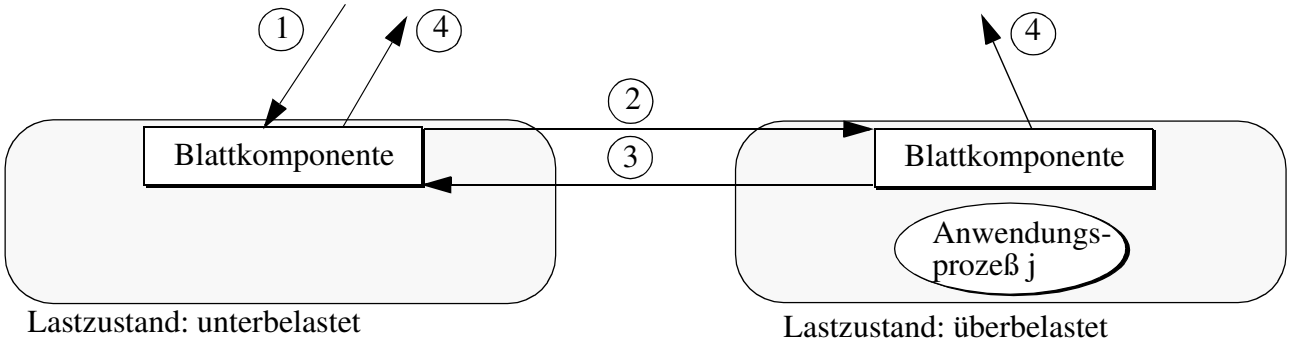
Wie bereits in den vorigen Abschnitten beschrieben, verfügt eine innere Komponente über zwei Tabellen. In der Lasttabelle sind die aggregierten Lastinformationen ihrer unmittelbaren Nachfolger abgelegt. In der Prozeßtabelle sind alle relevanten Informationen der Prozesse enthalten, die sich innerhalb des Wirkungsbereiches einer inneren Komponente befinden. Ausgehend von den Informationen in der Lasttabelle und der Prozeßtabelle ist eine innere Komponente bei einer lokalen Entscheidung somit in der Lage, nicht nur die Lastsituation, sondern auch die Kommunikationsbeziehungen der Anwendungsprozesse bei der Auswahl geeigneter Quell- und Zielknoten eines Lasttransfers zu berücksichtigen. Die Berücksichtigung der Kommunikationsbeziehungen bedeutet dabei, daß versucht wird, Anwendungsprozesse, die miteinander kommunizieren, nicht auf einen Knoten zu legen, da die Kommunikation zwischen Prozessen innerhalb eines Knotens um den Faktor 14.5 teurer ist als über Knotengrenzen hinweg. Die Gründe für die Diskrepanz zwischen der Interprozessor- und Intraprozessorkommunikation wurden in Kapitel 4 ausführlich beschrieben.

Wurde eine passende überbelastete Blattkomponente ausgewählt, wird ihre Adresse der unterbelasteten Blattkomponente mitgeteilt. Um zu verhindern, daß eine Blattkomponente mehrfach als Ziel- bzw. Quellknoten ausgewählt wird, setzt die innere Komponente anschließend den Lastzustand der beiden Knoten in ihrer Lasttabelle auf *normal belastet*. Als Knoten im Lastzustand *normal belastet* nehmen diese beiden Knoten dann, zumindest bis zum Eintreffen ihrer nächsten Lastnachricht, an keinem weiteren Lasttransfer teil.

Im Vergleich zur lokalen Entscheidung wird bei der nicht-lokalen Entscheidung keine prozeßbezogene Information verwendet, d.h. die Entscheidung wird nur aufgrund der Lastdaten in den/dem Knoten getroffen. Stellt eine Komponente - hierbei kann es sich sowohl um eine innere Komponente als auch um die Wurzelkomponente handeln - fest, daß es innerhalb ihres Wirkungsbereichs unter- und überbelastete Teilbäume gibt, fordert sie vom überbelasteten Teilbaum eine Adreßliste aller überbelasteten Blattkomponenten an. Diese sogenannte Knotenliste wird anschließend unter alle unterbelasteten Teilbäumen entsprechend ihrer Prozeßfehlmenge aufgeteilt. Wie bereits im lokalen Fall wird anschließend der Lastzustand der beteiligten Teilbäume auf *normal belastet* gesetzt, um eine Mehrfachaktivierung des Lokationsprotokolls zu vermeiden.

5.7 Migrationsentscheidung

Die Hauptaufgabe der inneren Komponenten ist die Lokationsentscheidung. Die Entscheidung, ob und wenn ja, welche Last zwischen den Knoten ausgetauscht wird, wird autonom von den korrespondierenden Blattkomponenten getroffen. Bevor auf die Umsetzung der Auswahlentscheidung in den Blattkomponenten detailliert eingegangen wird, werden noch einmal kurz die beiden den Blattkomponenten zur Verfügung stehenden Lasttransfermethoden beleuchtet.
Die Blattkomponenten verfügen über zwei Lasttransfermechanismen, die *Migration eines zwischengespeicherten Prozesses* und die *Migration eines laufenden Prozesses*. Während im ersten Fall nur eine Prozeßbeschreibung transferiert werden muß, wird im zweiten Fall ein gesamter Prozeßkontext zwischen den Knoten transferiert. Aus diesem Grund versucht eine überbelastete Blattkomponente, wenn möglich, jede Migrationsanfrage mit der Migration eines zwischengespeicherten Prozesses zu beantworten. In Abbildung 23 ist das entsprechende Migrationsprotokoll graphisch veranschaulicht.



Nachricht 1: Adresse einer überbelasteten Blattkomponente
Nachricht 2: Migrationsanfrage inkl. Last- und Prozeßtabelleninhalt
Nachricht 3: Transfer der Prozeßbeschreibung
Nachricht 4: Senden der aktuellen Last- bzw. Prozeßdaten an die übergeordnete Komponente

Abbildung 23: Protokoll für die Migration eines zwischengespeicherten Prozesses.

Bekommt eine unterbelastete Blattkomponente von ihrer übergeordneten Komponente die Adresse einer überbelasteten Blattkomponente, schickt sie eine Migrationsanfrage zusammen mit den vollständigen Informationen der Last- und Prozeßtabelle an die überbelastete Blattkomponente. Der Empfänger der Migrationsanfrage versucht daraufhin, diese Anfrage mit der Migration eines

zwischengespeicherten Prozesses zu beantworten, vorausgesetzt er befindet sich noch immer im Lastzustand *überbelastet*. Dazu wählt er den ersten noch nicht gestarteten Prozeß aus seiner Prozeßtabelle als Migrand aus. Auf eine aufwendige Auswahlentscheidung im Zusammenhang mit dieser Migration wird aus zwei Gründen verzichtet:

- Eine Migration eines zwischengespeicherten Prozesses verursacht nur geringe Last und kann somit einfach revidiert werden.
- Der dynamische Lastbalancierer verfügt nur über sehr geringe Informationen bzgl. des Lastverhaltens eines noch nicht gestarteten Anwendungsprozesses.

Verfügt eine überbelastete Blattkomponente über keine zwischengespeicherten Anwendungsprozesse, versucht sie die Migrationsanfrage mittels einer Migration eines bereits laufenden Anwendungsprozesses zu beantworten. Da bei dieser Migration der gesamte Kontext eines laufenden Prozesses zwischen zwei Knoten des Systems migriert werden muß, sind nicht nur die Kosten für die eigentliche Übertragung deutlich höher als bei der Migration eines zwischengespeicherten Prozesses, sondern auch das dazugehörige Protokoll deutlich komplexer.
In Abbildung 24 ist das Migrationprotokoll graphisch veranschaulicht. Dabei muß darauf hingewiesen werden, daß die Nachrichten 5 und 6 im Zusammenhang mit den höhren Informationsebenen benötigt werden. Befindet sich ein Anwendungsprozeß zum Zeitpunkt der Migrationsaufforderung in einer Bearbeitungsphase, die für eine Migration nicht geeignet ist, kann der Anwendungsprozeß die Migration ablehnen. Für den unwahrscheinlichen Fall, daß alle Anwendungsprozesse auf einem Knoten eine Migrationsaufforderung ablehnen, verfügt der Lastbalancierer noch über eine strenge Migrationsaufforderung, die von den Anwendungsprozessen nicht abgelehnt werden darf.
Während für die Migration eines zwischengespeicherten Prozesses nur eine sehr einfache Auswahlstrategie verwendet wird, muß für diese Art von Prozeßmigration im Hinblick auf deren Kosten eine komplexe Auswahlstrategie verwendet werden, da diese Auswahl einen erheblichen Einfluß auf die Leistungsfähigkeit des gesamten Lastbalancierungsansatzes hat, wie in Kapitel 7 noch ausführlich aufgezeigt wird. Damit nun zwischen einer unterbelasteten und einer überbelasteten Blattkomponente ein Lasttransfer mittels der Migration eines laufenden Prozesses stattfinden kann, müssen sowohl der Quell- bzw. der Zielknoten als auch der zu migrierende Anwendungsprozeß eine Reihe von Bedingungen erfüllen:

Regel 19:

Unter der Voraussetzung, daß der Zielknoten der Migration mindestens einen Awendungsprozeß besitzt und der Quellknoten der Migration innerhalb der letzten t_{Action} Sekunden an einer Lastbalancierungsaktivität (Prozeßstart, Prozeßmigration) teilgenommen hat, wird die Migrationsaufforderung abgelehnt.

Durch diese Regel wird sichergestellt, daß ein überbelasteter Knoten nicht durch Überreaktion auf seinen Lastzustand direkt in den Lastzustand *unterbelastet* wechselt. Verfügt der potentielle Zielknoten eines Lasttransfers hingegen über keinen Anwendungsprozeß, wird unabhängig von der zeitlichen Restriktion in der Auswahlentscheidung fortgefahren, da auf jeden Fall verhindert werden muß, daß einzelne Knoten unnötig lange ohne Arbeitslast sind.

Regel 20:

Die Grundvoraussetzung dafür, daß sich für einen Anwendungsprozeß die Migration auf einen anderen Knoten lohnt, ist, daß der Prozeß auf dem Zielknoten der Migration deutlich mehr Prozessorzeit bekommt als auf dem lokalen Knoten.

Die auf dem Zielknoten verfügbare Prozessorzeit für den Anwendungsprozeß kann aus der Anzahl der Anwendungsprozesse und der Leerlaufzeit auf dem Zielknoten abgeschätzt werden. Der aktuelle Prozessorbedarf des Anwendungsprozesses wird aus den Parametern Prozessorkonsum während der gesamten Laufzeit des Prozesses, Prozessorbedarf innerhalb der letzten zwei Meßintervalle und einer Prozessorbedarfsschätzung der Anwendung (siehe Umsetzung der Informationsebenen in Abschnitt 5.10) berechnet. Steht dem Prozeß auf dem Zielknoten nicht mindestens 50% mehr Prozessorzeit als auf dem lokalen Knoten zur Verfügung, ist der Prozeß kein geeigneter Migrationskandidat für diesen Zielknoten.

Regel 21:

Die Anzahl der freien Hauptspeicherseiten des Zielknotens muß größer/gleich sein wie die Anzahl der durch den Migrationskandidaten belegten Hauptspeicherseiten. Ansonsten ist der Prozeß kein geeigneter Migrationskandidat für diesen Zielknoten .

Das Zielsystem Intel Paragon XP/S verfügt über eine virtuelle Speicherverwaltung. Aufgrund der unzureichend dimensionierten Ein-/Ausgabeknoten des Systems sollte von dieser Betriebssystemfunktionalität aus Leistungsgesichtspunkten jedoch kein Gebrauch gemacht werden. Deshalb stellt der Lastbalancierer durch das obige Kriterium sicher, daß der Zielknoten durch den zusätzlichen Anwendungsprozeß nicht gezwungen wird, Hauptspeicherseiten auszulagern.

Regel 22:

Ein Anwendungsprozeß muß auf dem lokalen Knoten eine Mindestmenge an Prozessorzeit (ca. 3 Sekunden) konsumiert haben, damit er als Migrationskandidat in Frage kommt.

Durch diese Regel wird sichergestellt, daß erstens eine unproduktive *Prozeßmigration* vermieden wird und zweitens keine sogenannten Kurzläufer migriert werden, die im Laufe ihrer kurzen Berechnung keine Möglichkeit haben, die Migrationskosten auszugleichen (vgl. [Milo94]).

Regel 23:

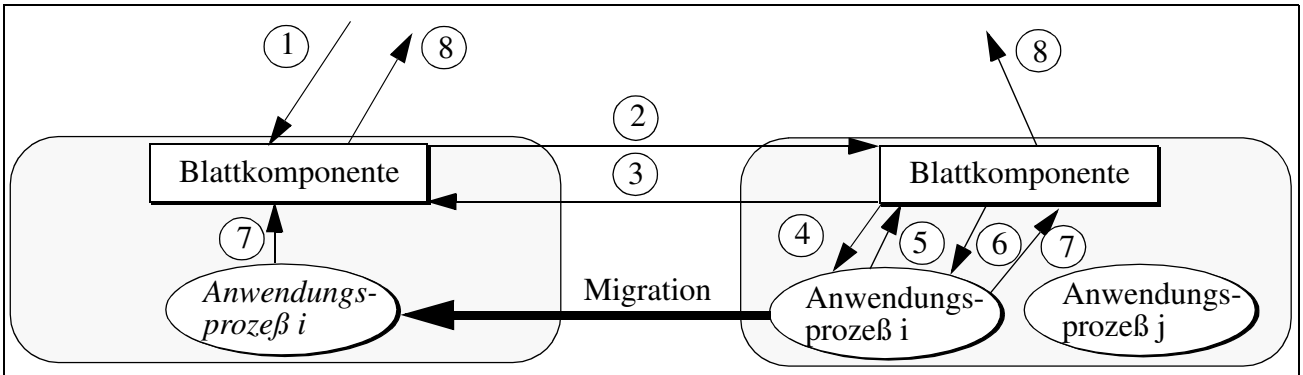
Ein Anwendungsprozeß muß innerhalb der letzten zwei Meßintervalle eine Mindestmenge an Prozessorzeit konsumiert haben, um als Migrationskandidat in Frage zu kommen.

Hat ein Prozeß innerhalb der letzten zwei Meßintervalle keine Prozessorzeit beansprucht, bedeutet dies, daß die Überlast durch andere Anwendungsprozesse auf dem Knoten verursacht wurde bzw. immer noch wird. Eine Migration des Prozesses würde sich deshalb nicht positiv auf die Lastsituation des Knotens auswirken.

Regel 24:

Um zu verhindern, daß ein Quellknoten nach der Migration vom Lastzustand *überbelastet* direkt in den Lastzustand *unterbelastet* wechselt, muß nach der Migration der Prozessorbedarf der verbleibenden Anwendungsprozesse ausreichen, um den Prozessor auszulasten. Dazu wird vom aktuellen Prozessorbedarf der verbleibenden Anwendungsprozesse auf deren zukünftigen Prozessorbedarf geschlossen.

Während die bisherigen Regeln sich ausschließlich auf Informationen der Ebene 0 stützten, basieren die beiden nachfolgenden Regeln auf den Informationen der höheren Ebenen. Für eine ausführliche Diskussion der verwendeten Informationen der höheren Ebenen sowie deren Umsetzung im hierachischen Lastbalancierungssystem PaLaBer wird auf Abschnitt 5.10 verwiesen.



Nachricht 1: Adresse einer überbelasteten Blattkomponente

Nachricht 2: Migrationsanfrage inkl. Last- und Prozeßtabelleninhalt

Nachricht 3: Ablehnung der Migrationsaufforderung

Nachricht 4: Blattkomponente fordert Anwendungsprozeß zur Migration auf

Nachricht 5: Anwendungsprozeß bestätigt die Migrationsaufforderung oder schickt eine Ablehnung der Migrationsaufforderung (siehe Umsetzung der Informationsebenen)

Nachricht 6: Engültige Aufforderung zur Migration

Nachricht 7: Migrand meldet sich bei seiner Blattkomponente an bzw. ab

Nachricht 8: Senden der aktuellen Last- bzw. Prozeßdaten an die übergeordnete Komponente

Abbildung 24: Protokoll für die Migration eines laufenden Prozesses.

Regeln im Zusammenhang mit höheren Informationsebenen

Regel 25:

Verfügt der Lastbalancierer über Informationen bezüglich der Restlaufzeit eines Anwendungsprozesses, wird abgeschätzt, ob der Anwendungsprozeß in der Lage ist, die Migrationskosten im Laufe seiner Restbearbeitung auszugleichen. Da es sich bei dieser Berechnung nur um eine grobe Abschätzung handeln kann (das Working-Set des Anwendungsprozesses kann sich ändern und damit die Größe des zu übertragenden Kontextes), wird bei der Berechnung noch ein Sicherheitszuschlag von 100% berücksichtigt, d.h. der Lastbalancierer geht davon aus, daß die Prozeßmigration doppelt so lange dauert.

Für einen Lastbalancierer der Ebene 0 ist es unmöglich, herauszufinden, ob ein Anwendungsprozeß noch lange genug laufen wird, um die nicht unerheblichen Migrationskosten auszugleichen. In der Literatur wird deshalb häufig eine Heuristik vorgeschlagen, die davon ausgeht, daß ein Prozeß zum Migrationszeitpunkt maximal 50% seiner Gesamtlaufzeit absolviert hat. Durch diese Heuristik wird sichergestellt, daß ein Prozeß in der Anfangsphase seiner Berechnung nicht migriert wird. Dadurch stellt diese Heuristik sicher, daß sogenannte Kurzläufer nicht als Migrationskandidaten ausgewählt werden. Der entscheidende Nachteil dieser Heuristik ist jedoch, daß ein Prozeß nach einer gewissen Zeit immer als Migrationskandidat in Frage kommt.

Regel 26:

Verfügt der Lastbalancierer über Informationen bezüglich des durchschnittlichen zukünftigen Prozessorbedarfs eines Prozesses, wird überprüft, ob der Prozeß auch in Zukunft noch

signifikante Prozessorzeit konsumieren wird, d.h. ob der Prozeß positiv auf die Unterlast des Zielknotens einwirken wird.

Unter allen Prozessen auf dem Quellknoten, die die oben beschriebenen Kriterien erfüllen, wird versucht, einen Anwendungsprozeß auszuwählen, der keinen Kommunikationspartner auf dem Zielknoten hat. Diese Auswahl ist möglich, da die unterbelastete Blattkomponente zusammen mit der Migrationsaufforderung den Inhalt ihrer gesamten Prozeßtabelle mitschickt. Ist es nicht möglich, einen solchen Prozeß zu finden, wird der Anwendungsprozeß mit dem geringsten Kommunikationsaufkommen ausgewählt, der die obigen Regeln erfüllt.

5.7.1 Erweitertes Prozeßzustandsdiagramm

Durch die Einführung der beiden Prozeßmigrationsarten erhöht sich die Anzahl der möglichen Zustände, in denen sich ein Prozeß befinden kann. Ohne das Vorhandensein der Prozeßmigration befindet sich ein Prozeß in einem der fünf Zustände *initiiert*, *bereit*, *aktiv*, *blockiert* oder *terminiert* (vgl. Abbildung 25). In Abbildung 26 ist das erweiterte Prozeßzustandsdiagramm abgebildet. Vom Zustand *initiiert* wechselt ein Prozeß in den Zustand z*wischengespeichert*. In diesem Zustand kann ein Prozeß beliebig oft migriert werden. Wird ein Prozeß von einer Blattkomponente gestartet,

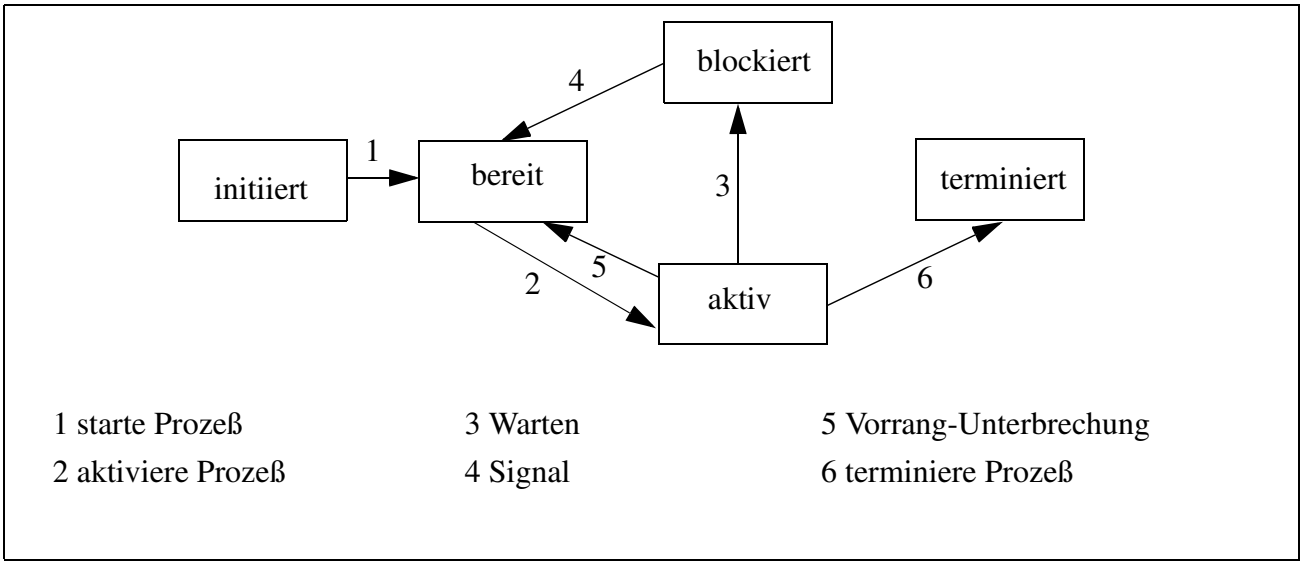


Abbildung 25: Konventionelles Prozeßzustandsdiagramm.

wechselt er in den Zustand *bereit* und unterliegt von nun an der Prozeßverwaltung des Betriebssystems (*aktiv*, *bereit*, *blockiert*, *terminiert*).Aufgrund der höheren Informationsebenen, speziell der Informationsebene 2, unterscheidet das PaLaBer-System jedoch zwischen zwei aktiven Zuständen. Ein Prozeß kann sich entweder im Zustand *aktiv* und somit migrierbar oder im Zustand *aktiv-nicht migrierbar* befinden. Da die Migration eines bereits gestarten Prozesses innerhalb der Anwendungsbibliothek PaLib stattfindet, wechselt ein Prozeß direkt vom Zustand *aktiv* in den Zustand *laufend migrierend*. Da es während der Migration zwei Instanzen desselben Prozesses gibt, die beide der Betriebssytemverwaltung der jeweiligen Knoten unterliegen, gibt es für diese Phase eigene *bereit* bzw. *blockiert* Zustände.

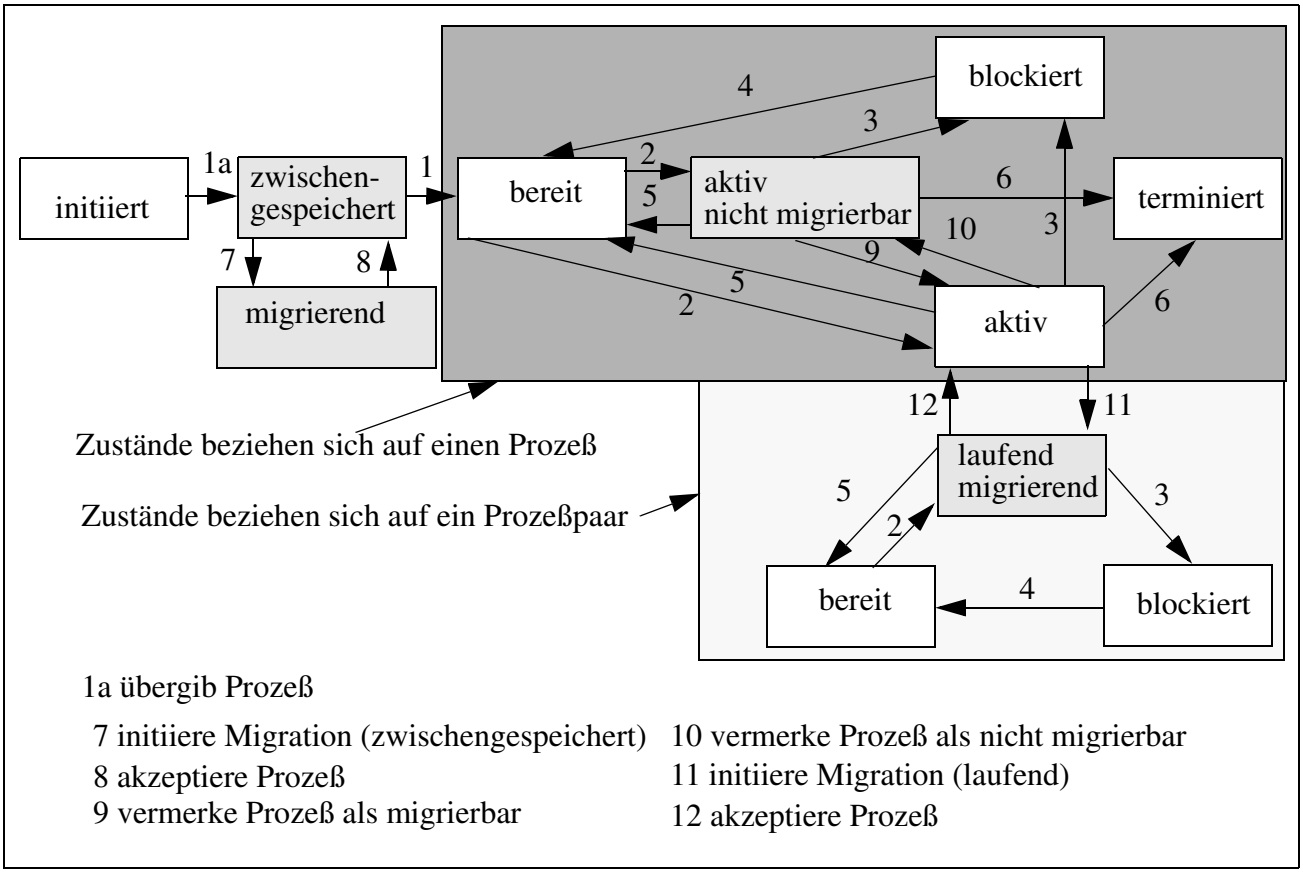


Abbildung 26: Erweitertes Prozeßzustandsdiagramm.

5.8 Ortstransparente Kommunikation

Ortstransparente Kommunikation bedeutet, daß der Sender einer Nachricht nicht die aktuelle Lokation des Empfängers kennen muß [RaJe88]. Statt dessen erfolgt die Adressierung mit Hilfe einer systemweit eindeutigen, virtuellen Adresse, die zur Laufzeit vom Kommunikationssystem auf die reale Adresse abgebildet wird. Die Einführung der Ortstransparenz ist unerläßlich für die anwendungstransparente Prozeßmigration durch einen dynamischen Lastbalancierer. Sie erleichtert jedoch auch die Eingliederung von Mechanismen zur Softwarefehlertoleranz in parallelen und verteilten Systeme und stellt einen erheblichen Programmierkomfort dar, da der Anwendungsprogrammierer von Detailkenntnissen über die Hardware entlastet wird.
Im PaLaBer - System wurde zur Realisierung der Ortstransparenz ein indirekter Kommunikationsansatz (siehe Anhang C) gewählt, da sich nur dieser Ansatz für große parallele und verteilte Anwendungen eignet, insbesondere wenn die Anwendungsprozesse eine große Anzahl von Kommunikationspartnern aufweisen. Aufgrund der hierarchischen Struktur des dyamischen Lastverwalters wurde auch für den Kommunikationsmanager ein hierarchischer Ansatz gewählt. Der hierarchische Kommunikationsmanager besteht wie der Lastverwalter aus drei verschiedenen Komponenten: den Blattkomponenten, mindestens einer inneren Komponente und einer Wurzelkomponente. Wie beim Lastbalancierer ist eine Blattkomponente für einen Knoten verantwortlich, eine innere Komponente für eine Gruppe von Blatt- oder inneren Komponenten. Die Wurzel dient als Protokollkoordinator zwischen den Teilbäumen.
Da es in großen parallelen und verteilten Systemen weder sinnvoll noch möglich ist, jeder Komponente des Kommunikationsmanagers eine vollständige Adreßtabelle zur Verfügung zu stellen,

verfügen die Knoten im PaLaBer-System nur über unvollständige (und bedingt korrekte) Informationen. Neben den Prozeßtabellen (siehe Abbildung 27), die von den Lastbalancierungskomponenten verwaltet werden (vgl. Abschnitt 5.3), besitzen die Blatt- sowie die inneren Komponenten noch Adreßcaches fester Größe. In diesen Pufferspeichern, die nach dem *Least-Recently-Used* Prinzip (kurz LRU) verwaltet werden, halten die Komponenten die Adressen der letzten *n* Kommunikationspartner ihrer lokalen Prozesse.
Damit ein Anwendungsprozeß auch nach einer Migration die Adressen seiner Kommunikationspartner zur Verfügung hat, besitzen auch die Anwendungsprozesse einen Adreßcache, die durch die Anwendungsbibliothek PaLib verwaltet werden.
In den folgenden beiden Unterabschnitten werden die Protokolle zum Versenden einer Anwendungsnachricht und zum Auffinden eines Anwendungsprozesses im Detail beschreiben, da sich diese Protokolle kritisch auf die Leistungsfähigkeit des gesamten Laufzeitsystems auswirken.

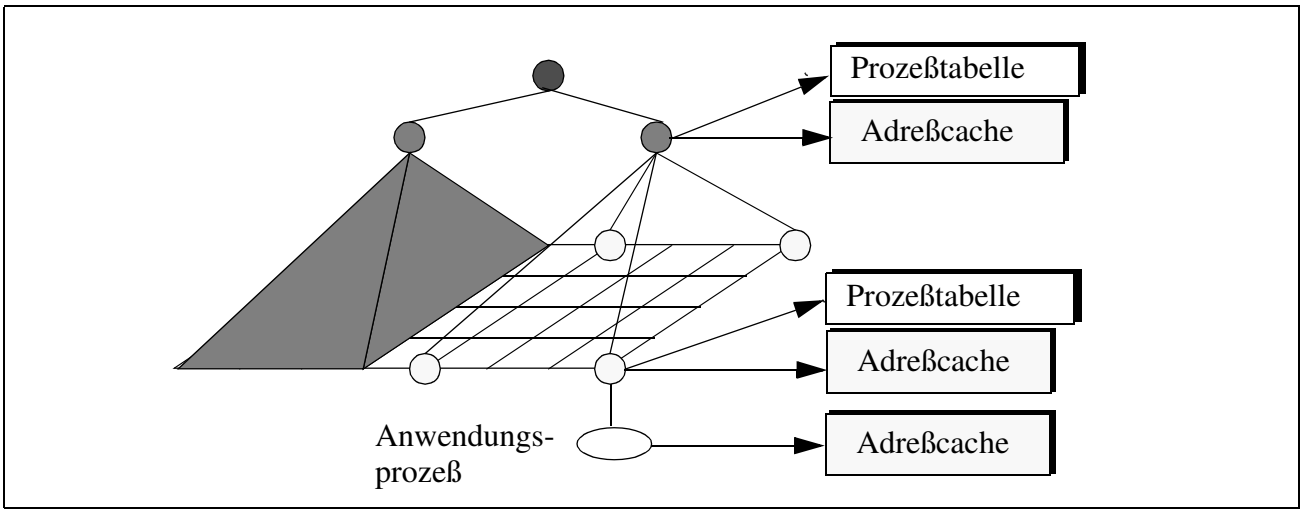


Abbildung 27: Aufteilung der Adreßtabellen im PaLaBer - System.

5.8.1 Protokoll zum Versenden einer Anwendungsnachricht

Für das Protokoll zum Versenden einer Anwendungsnachricht müssen drei Fälle unterschieden werden:

1. Dem Anwendungsprozeß ist die Adresse des Empfängers bekannt, d.h. es fand bereits ein Nachrichtenaustausch zwischen diesen Prozessen statt.
2. Dem lokalen Kommunikationsmanager ist die Adresse des Empfängers bekannt, d.h. es fand bereits ein Nachrichtenaustausch zwischen dem Knoten und dem Empfänger der Nachricht statt.
3. Auf dem Knoten ist die Adresse des Empfängers nicht bekannt. Dieser Fall wird im nächsten Abschnitt gesondert behandelt.

Fall 1: Der Sender kennt die Adresse des Empfängers.

Hat Prozeß A (vgl. Abbildung 28) bereits mit Prozeß B kommuniziert, findet er die vermutliche Adresse von Prozeß B in seinem lokalen Adreßcache. Unter der optimistischen Annahme, daß sich der Empfänger noch immer auf diesem Knoten befindet, schickt er die Nachricht dorthin. Der lokale Kommunikationsmanager auf Knoten 2 aktualisiert daraufhin seinen Adreßcache mit der Adresse von Prozeß A, leitet die Nachricht an den Empfänger weiter, falls dieser sich noch auf dem Knoten

befindet, und schickt dem Sender der Nachricht eine Empfangsbestätigung. Mit Hilfe der Empfangsbestätigung wird die Einhaltung der Nachrichtenreihenfolge gewährleistet. Erst wenn Prozeß A diese Empfangsbestätigung erhalten hat, schickt er wieder eine Nachricht an Prozeß B. Aus Effizienzgründen wartet der Sender natürlich nicht synchron auf diese Empfangsbestätigung. Erst wenn er wieder eine Nachricht an einen Prozeß verschicken will, überprüft er, ob eine Empfangsbestätigung von diesem noch aussteht und wartet gegebenenfalls auf diese. Außerdem kann der Sender mit Hilfe der Empfangsbestätigung seinen Adreßcache aktualisieren.
Befindet sich Prozeß B nicht mehr auf dem Knoten, an den Prozeß A die Nachricht geschickt hat, durchsucht der lokale Kommunikationsmanager auf Knoten 2 seinen Adreßcache nach der neuen Adresse von Prozeß B. Für den unwahrscheinlichen Fall, daß die Information auch nicht mehr im Adreßcache zu finden ist, leitet der Kommunikationsmanager auf Knoten 2 das Suchprotokoll ein.

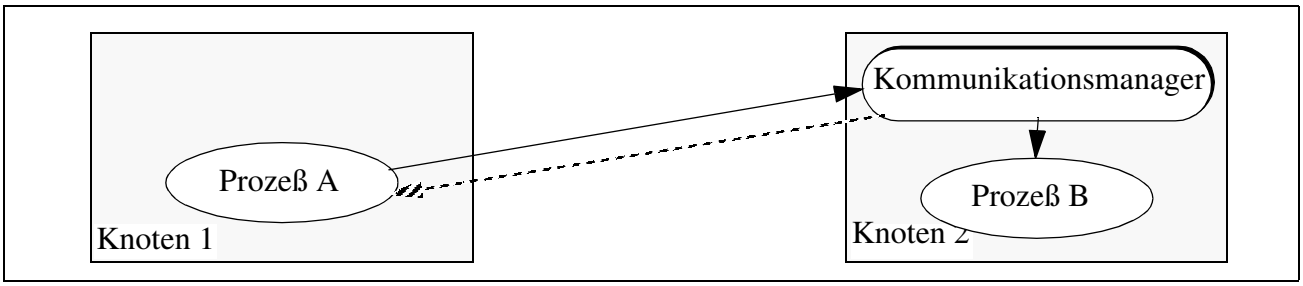


Abbildung 28: Nachrichtenfluß - Sender kennt die Adresse des Empfängers.

Fall 2: Lokaler Kommunikationsmanager kennt die Adresse des Empfängers.

Findet der Sender einer Nachricht die Adresse des Empfängers nicht in seinem prozeßlokalen Adreßcache, schickt er die Anwendungsnachricht an seinen lokalen Kommunikationsmanager (vgl. Abbildung 29). Dieser durchsucht daraufhin seine Prozeßtabelle und den lokalen Adresscache (vgl. Abbildung 27) und leitet die Nachricht, falls er die Adresse in einer seiner Tabellen findet, an den zuständigen Kommunikationsmanager (in Abbildung 29 ist dies der lokale Kommunikationsmanager auf Knoten 2) weiter.

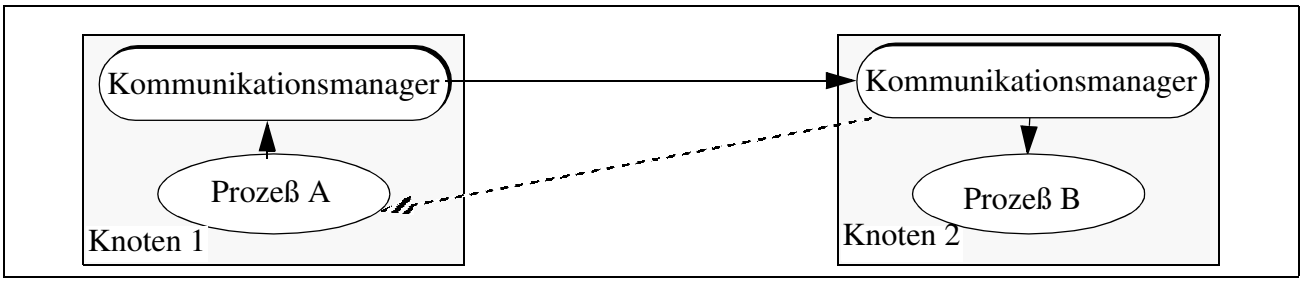


Abbildung 29: Nachrichtenfluß - lokaler Kommunikationsmanager kennt die Adresse.

5.8.2 Protokoll zum Auffinden eines Anwendungsprozesses

Kann ein Kommunikationsmanager auf Ebene der Blattkomponenten eine Anwendungsnachricht nicht zustellen, da er die Adresse des Empfängers weder in seiner Prozeßtabelle noch in seinem Adreßcache findet, startet er das Suchprotokoll. Dazu speichert er die Nachricht lokal zwischen und schickt eine Suchanfrage an seine übergeordnete Komponente. Diese durchsucht daraufhin ihre Prozeßtabelle und ihren Adreßcache. Findet sie die Adresse des Empfängers, leitet sie die Adresse an

ihre untergeordnete Komponente weiter. Ansonsten wird die Suchanfrage nach *oben* weitergeleitet. Kommt die Anfrage schließlich bei der Wurzelkomponente an, leitet sie diese an die anderen Teilbäume weiter, da sie selbst über keine Prozeßtabellen verfügt. Befindet sich ein Prozeß im System, wird seine Adresse spätestens in den Prozeßtabellen der inneren Komponenten der obersten Stufe gefunden, da diese alle Prozesse in ihren Teilbäumen kennen. Kann die Adresse eines Prozesses nicht ermittelt werden, bekommt der Sender eine entsprechende Fehlermeldung. Für das Suchprotokoll werden somit maximal

$$3\ (+1) + 2n + 2(m-1)$$

Nachrichten benötigt. 3(+1) Systemnachrichten werden für die eigentliche Zustellung der Anwendungsnachricht und für die Empfangsbestätigung benötigt. 2n Nachrichten werden für die Weiterleitung der Suchnachricht und des Ergebnisses über die n-Stufen der inneren Komponenten benötigt und 2(m-1) Nachrichten für das Weiterleiten der Suchanfrage und des Ergebnisses in die verschiedenen Teilbäume durch die Wurzelkomponente. Typische Werte für n sind 1 oder 2; m liegt zwischen 1 und 9.

5.8.3 Integration der ortstransparenten Kommunikations in das Betriebssystem

Bislang wurde die Realisierung der Ortstransparenz im PaLaBer - System als zusätzliche Schicht zwischen Anwendung und Betriebsystem beschrieben. Bei dieser Realisierung werden für jede Anwendungsnachricht mehrere Systemnachrichten benötigt. Wird hingegen die Ortstransparenz in das Kommunikationssytem des Betriebssystems integriert, wird für die ortstransparente Kommunikation keine zusätzliche Nachricht benötigt, mit Ausnahme für das Suchprotokoll. Kommunizieren zwei Anwendungsprozesse unter Verwendung eines konventionellen Kommunikationssystems miteinander, bedeutet dies nicht, daß die Nachricht vom Sender direkt zum Empfänger übermittelt wird. Vielmehr übergibt der Sender die Nachricht an das knotenlokale Kommunikationssubsystem. Dieses sendet die Nachricht dann an den Zielknoten. Das Betriebssystem auf dem Zielknoten stellt die Nachricht dem eigentlichen Empfänger zu. Wenn man desweiteren berücksichtigt, daß zwischen den Kommunikationssystemen auf dem Quell- und Zielknoten noch zusätzliche Nachrichten ausgetauscht werden, um z.B. sicherzustellen, daß auf dem Zielknoten ausreichend freier Pufferspeicher zur Verfügung steht ([KoHa96] [Pier94]), gelangt man zu einem vergleichbaren Nachrichtenfluß wie er in Abschnitt 5.8.1 beschrieben ist.

5.9 Die Programmierschnittstelle

Die Interaktion der Anwendungsprozesse mit dem hierarchisch organisierten PaLaBer-System erfolgt über die Anwendungsbibliothek PaLib (siehe Abbildung 19). Diese Bibliothek stellt sowohl Routinen zur Interprozeßkommunikation als auch zur Prozeßverwaltung zur Verfügung, d.h. die parallelen Anwendungen verwenden ausschließlich die Routinen der PaLib für die Interprozeßkommunikation und Prozeßverwaltung. Wann immer nun ein Anwendungsprozeß eine der Routinen der Bibliothek aufruft, besteht die Möglichkeit der Interaktion zwischen Anwendungsprozeß und dem dynamischen Lastbalancierer. Somit findet eine Interaktion nur zu diskreten Zeitpunkten statt[1].
Da die Untersuchungen, die in Kapitel 7 vorgestellt werden, mit bereits existierenden parallelen

1. Eine Realisierung der Interaktion zwischen den Blattkomponenten und dem Anwendungsprozeß mittels eines asynchron agierenden Threads war aufgrund von Limitationen des Betriebssystems leider nicht möglich.

Anwendungen durchgeführt wurden, ist es notwendig, eine bestehende, etablierte Kommunikationsbibliothek zu unterstützen. Da das Zielsystem für die Untersuchungen die Intel Paragon ist, unterstützt das PaLaBer-System das Programmier- und Ablaufmodell der NX-Bibliothek von Intel [Inte94]. Diese Bibliothek enthält sowohl Routinen für die Interprozeßkommunikation als auch für die Prozeßverwaltung. Um nun ein bestehendes „NX-Anwendungsprogramm“ in eine „PaLaBer-Programm“ zu konvertieren, ist es ausreichend, die verwendeten NX-Routinen um das Präfix **pa_** zu erweitern und eine zusätzliche Initialisierungs- und Terminierungsroutine einzubinden. Ansonsten sind keine Modifikationen des Anwendungsprogramms erforderlich, da die gesamte dynamische Lastbalancierung für die Anwendung transparent erfolgt. Auf die Nutzung der höheren Informationsebenen und die dafür zu entwickelnde Anwendungsschnittstelle wird erst in Abschnitt 5.10 eingegangen. Da es sich bei der NX-Bibliothek um eine sehr umfangreiche Bibliothek handelt, deren vollständige Unterstützung durch das PaLaBer-System den Rahmen des Projektes gesprengt hätte, wurden exemplarisch zwölf Routinen der NX-Bibliothek in das PaLaBer-System übernommen. Die Schnittstellendefinitionen dieser Routinen sind im Anhang B beschreiben.
Während sich das Programmier- und Ablaufmodell des PaLaBer-Systems nur marginal von dem der NX-Bibliothek unterscheidet, weicht das Starten einer parallelen Anwendung deutlich ab, da dem Anwender eine virtuelle Maschine zur Verfügung gestellt wird[1]. Die virtuelle Maschine kann dabei deutlich mehr Knoten besitzen als die reale Maschine. Aus dem Verhältnis zwischen virtuellen und realen Knoten ergibt sich für den Lastbalancierer die gewünschte Anzahl von Anwendungsprozessen pro Blattkomponente. Diese Information wird sowohl bei der Berechnung der adaptiven Schwellwerte sowie der Prozeßfehlmenge beim Starten einer parallelen Anwendung benötigt. Basierend auf der virtuellen Maschine kann sich der Anwender beim Start einer parallelen Anwendung eine virtuelle Partition mit einer Knotenanzahl, die optimal für seine Anwendung ist allokieren. Die maximale Knotenanzahl einer virtuellen Partition ist dabei durch die Größe der virtuellen Maschine vorgegeben.

5.10 Nutzung der Informationsebenen im PaLaBer-System

In Kapitel 3 wurde das Konzept der Informationsebenen eingeführt. In diesem Abschnitt wird nun die Nutzung der höheren Informationsebenen im PaLaBer-System vorgestellt. Außerdem wird auf die neu definierte Schnittstelle zwischen dem dynamischen Lastbalancierer und den parallelen Anwendungen eingegangen. Die Informationen der Ebene 0 bzw. deren Verwendung im PaLaBer-System werden in diesem Abschnitt nicht weiter behandelt, da dies bereits Bestandteil der vorigen Abschnitte war.
Auf die erzielbaren Leistungsgewinne, basierend auf den Informationen der höheren Ebenen, wird dann in Kapitel 7 ausführlich eingegangen.

5.10.1 Nutzung der Informationsebene 1

Zur Erinnerung sei gesagt, daß die Informationsebene 1 alle statischen bzw. statistischen Informationen über die Anwendungs- und Betriebssystemcharakteristik beinhaltet. Aus der Vielzahl von möglichen Informationen der Informationsebene 1 verwendet das PaLaBer-System Informationen aus folgenden drei Bereichen:

1. Die virtuelle Maschine wird dabei vom Anwender in einer Konfigurationsdatei definiert.

- **Statische Plazierungsinformation**
  Bei der Übergabe einer parallellen Anwendung kann der Anwender durch die Plazierung der Prozesse innerhalb der virtuellen Partition statische Plazierungsinformation an den Lastbalancierer weitergeben.

Diese Information wird, wie in Abschnitt 5.5 beschrieben, zur Aufteilung der Anwendungsprozesse auf die verschiedenen Teilbäume verwendet. Dadurch hat der Anwender die Möglichkeit sicherzustellen, daß Anwendungsprozesse mit einer hohen Kommunikationsintensität innerhalb des gleichen Teilbaumes plaziert werden.

- **Statische Angaben über die Anzahl und virtuelle Lokation der Kommunikationspartner eines Anwendungsprozesses**
  Über eine neu definierte Schnittstelle hat ein Anwendungsprozeß die Möglichkeit, dem Lastbalancierer sowohl die Anzahl als auch die Adressen seiner wichtigsten Kommunikationspartner mitzuteilen. Die Anwendungsbibliothek PaLib erfaßt daraufhin selbständig Lastdaten (Anzahl der Nachrichten, die zwischen den Kommunikationspartner ausgetauscht werden) über diese Kommunikationsbeziehungen und teilt diese dem Lastbalancierer mit. Der Funktionsaufruf für diese Information lautet:

```
StaticCommunicationPartner(long NumberOfPartners,long ListOfPartners[ ]);
```

Diese Informationen werden vom Lastbalancierer bei der Auswahl eines geeigneten Migrationskandidaten verwendet, indem er versucht, kommunizierende Anwendungsprozesse möglichst nicht auf einen Knoten zu plazieren, da, wie bereits in Kapitel 4 beschrieben, die Intraprozessorkommunikation teurer ist als die Interprozessorkommunikation.

- **Statistische Information über die benötigte Prozessorzeit und den Hauptspeicherbedarf der Anwendungsprozesse**
  Diese Information (Anzahl der belegten Hauptspeicherseiten, Gesamtlaufzeit, konsumierte Prozessorzeit) wird von den Blattkomponenten der Lastbalancierungshierarchie transparent für die Anwendung erfaßt und verwertet. In der prototypischen Realisierung der Lastbalancierungsumgebung wird die Information für jede Anwendung in Abhängigkeit der Knotenanzahl erfaßt.

Die Information über den statistischen Hauptspeicherbedarf eines Anwendungsprozesses verwendet der Lastbalancierer beim Starten eines Anwendungsprozesses zur Abschätzung, ob der Knoten noch ausreichend freie Hauptspeicherseiten für den neuen Anwendungsprozeß hat. Die statistische Information über den Prozessorbedarf wird vom Lastbalancierer in zweifacher Hinsicht verwendet. Erstens kann er mit dieser Information bei der Auswahl eines geeigneten Migrationskandidaten abschätzen, ob der Prozeß in seiner Restlaufzeit noch in der Lage sein wird, die Migrationskosten auszugleichen. Zweitens: Hat ein Anwendungsprozeß seine statistische Laufzeit überschritten und innerhalb der letzten zwei Meßintervalle weniger als 25% seines Prozessorquantums ausgenützt, wird er von seiner zuständigen Blattkomponente so eingestuft, als hätte er seine vorzeitige Terminierung dem Lastbalancierer bereits mitgeteilt (vgl. Nutzung der Informationsebene 3). Ändert sich das Lastverhalten des Anwendungsprozesses nachträglich, wird diese Einstufung vom Lastbalancierer wieder rückgängig gemacht.

5.10.2 Nutzung der Informationsebene 2

Zur Erinnerung sei darauf hingewiesen, daß die Informationsebene 2 alle Angaben umfaßt, die zur Laufzeit von der Anwendung über ihren aktuellen Bearbeitungszustand an den dynamischen

Lastbalancierer weitergegeben werden. Aus der Vielzahl von möglichen Informationen der Informationsebene 2 verwendet das PaLaBer-System Informationen aus den folgenden Bereichen:

- **Anwendungsprozeß befindet sich in einer Kommunikations- bzw. Ein-/Ausgabephase**
  Ein Aufruf der Routinen

  ```
  CommunicationPhase();
  IOPhase();
  ```

  bewirkt in der Anwendungsbibliothek PaLib lediglich das Setzen einer Kontrollmarke. Erst wenn der Anwendungsprozeß eine Migrationsaufforderung bekommt, wird diese Phaseninformation zusammen mit der Migrationsablehnung an den Lastbalancierer weitergeleitet. Der Lastbalancierer vermerkt daraufhin, daß der Anwendungsprozeß momentan nicht migrierbar ist.

Damit der Lastbalancierer jedoch auch in der Lage ist, Prozesse zu migrieren, wenn alle Prozesse auf dem Knoten von sich aus eine Migration ablehnen, verfügt er noch über eine *strenge* Migrationsaufforderung. Diese kann von einem Anwendungsprozess nicht abgelehnt werden. Die strenge Migrationsaufforderung wurde in der Protokolldarstellung in Abbildung 24 auf Seite 83 nicht dargestellt.

- **Anwendungsprozeß befindet sich in einer Rechenphase**
  Wurde in der vorangegangenen Phase eine Migration abgelehnt, wird der Phasenwechsel dem Lastbalancierer von der PaLib transparent für die Anwendung durch das Senden einer Systemnachricht mitgeteilt. Ansonsten bewirkt dieser Aufruf nur das Setzen einer Kontrollmarke in der PaLib-Bibliothek. Der Funktionsaufruf lautet:

  ```
  ComputationPhase();
  ```

Mit Hilfe dieser drei einfachen Schnittstellen kann ein Anwendungsprozeß einem dynamischen Lastbalancierer mitteilen, ob er sich in einer Bearbeitungsphase befindet, die sich für eine Migration eignet.

5.10.3 Nutzung der Informationsebene 3

Zur Erinnerung sei gesagt, daß die Informationsebene 3 alle Angaben von der Anwendung über ihr voraussichtliches, zukünftiges Lastverhalten umfaßt, die zur Laufzeit an den dynamischen Lastbalancierer weitergegeben werden. Aus der Vielzahl von möglichen Informationen der Informationsebene 3 verwendet das PaLaBer-System Informationen aus drei Bereichen:

- **Vorzeitige Ankündigung der Terminierung eines Anwendungsprozesses**
  In der Schlußphase seiner Berechnung teilt der Anwendungsprozeß mit Hilfe der Routine

  ```
  AdvanceTerminationNotice();
  ```

  dem Lastbalancierer seine baldige Terminierung mit. Der Lastbalancierer setzt daraufhin in der Prozeßtabelle eine Kontrollmarke. Sobald der Anwendungsprozeß in zwei aufeinanderfolgenden Meßintervallen weniger als 25% seines Prozessorquantums konsumiert, wird der NS_{AT} - Zähler inkrementiert. Verändert der Anwendungsprozeß sein Lastverhalten deutlich, wird diese Inkrementierung wieder rückgängig gemacht.

Diese Information sollte ca. 3 bis 4 Meßintervalle vor der eigentlichen Terminierung bzw. Einstellung der Berechnung an den Lastbalancierer weitergeleitet werden. Dieser Zeitraum ist völlig ausreichend für den Lastbalancierer, um sie bei Transfer-, Lokations- und Auswahlentscheidungen zu berücksichtigen.

- **Abschätzung der Restlaufzeit eines Anwendungsprozesses**
  Zur Abschätzung der Restlaufzeit eines Prozesses stehen der Anwendung drei Routinen zur Verfügung:

  ```
  StartRunTimeMeasurement();
  EndRunTimeMeasurement();
  RemainingRunTime(Quantum);
  ```

  Die ersten zwei Routinen dienen zur Erfassung der konsumierten Prozessorzeit innerhalb eines repräsentativen Verarbeitungsabschnittes. Mit Hilfe der dritten Routine teilt der Anwendungsprozeß dem Lastbalancierer mit, wie oft er diese Prozessorzeit noch benötigt. Da die gesamte Messung innerhalb der Anwendungsbibliothek PaLib erfolgt, ist die Systemzusatzbelastung durch diese Routinen gering. Außerdem wurden sie so realisiert, daß der Anwendungsprozeß während der Messung beliebig oft migriert werden kann. Die Migrationskosten werden dabei weitgehend herausgerechnet, soweit dies aufgrund der verwendeten Migrationsroutine möglich ist.

Diese Laufzeiterfassung eignet sich natürlich nur für Anwendungen, bei denen ein repräsentativer Verarbeitungsabschnitt mehrfach ausgeführt wird. Für die in Kapitel 6 vorgestellten Anwendungen eignet sich dieses Verfahren jedoch. So haben Vergleichsmessungen ergeben, daß die tatsächliche Laufzeit i.d.R. nur um ca. 10% von der Laufzeitprognose abweicht. Dies resultiert daraus, daß diese Messungen von einem Anwendungsprozeß zur Laufzeit mehrfach wiederholt und so den dynamischen Laufzeitbedingungen angepaßt werden können.

- **Durchschnittlicher Prozessorbedarf eines Anwendungsprozesses innerhalb des nächsten Bearbeitungsabschnittes**
  Zur Abschätzung des Prozessorbedarfes eines Anwendungsprozesses innerhalb der nächsten Bearbeitungsphase stehen drei Routinen zur Verfügung:

  ```
  StartBehaviorMeasurement(double EstimatedCpuBehavior);
  EndBehaviorMeasurement();
  UnknownBehavior();
  ```

  Basierend auf diesen drei Routinen wird ermittelt, wieviel Prozent der Anwendungsprozess von seiner ihm theoretisch zur Verfügung stehenden Prozessorzeit (Zeit / Anzahl der Anwendungsprozesse) genutzt hat. Dazu übergibt die Anwendung zu Beginn der Messung einen Schätzwert (*EstimatedCpuBehavior*) über den voraussichtlichen Prozessorbedarf des Anwendungsprozesses an den Lastbalancierer. Dieser Schätzwert wird dann durch das Meßergebnis am Ende der Messung ersetzt.
  Den durchschnittlichen Prozessorbedarf eines Anwendungsprozesses verwendet die Blattkomponente um ihre voraussichtliche, zukünftige Prozessorauslastung zu bestimmen und gegebenenfalls von ihrer übergeordneten Komponente präventiv Last anzufordern.
  Auch während dieses Meßvorganges kann der Anwendungsprozeß beliebig oft migriert werden, ohne signifikanten Einfluß auf das Meßergebnis zu nehmen.

Insgesamt kann festgestellt werden, daß mit Ausnahme der statischen Plazierungsinformation sämtliche Informationen der höheren Ebenen bereits in den Blattkomponenten aufbereitet werden. Dadurch ist es möglich, die volle Parallelität in der untersten Stufe der Lastbalancierungshierarchie für diese Aufgabe auszunutzen. Dies schlägt sich positiv auf die erzielten Meßergebnisse, die in Kapitel 7 dargestellt werden, nieder.

Kapitel 6

Leistungsbewertung des Lastbalancierungsansatzes

In diesem Kapitel wird die Leistungsfähigkeit der im vorigen Kapitel vorgestellten hierarchischen Lastbalancierungsumgebung PaLaBer untersucht. Da es aufgrund der Komplexität des Gesamtsystems nicht möglich ist, die korrekte Arbeitsweise des Lastbalancierungssystems in einem streng mathematischen Sinne zu beweisen, soll anhand von verschiedenen Konfigurationen aufgezeigt werden, daß sich das System im Rahmen der testbaren Konfigurationen plausibel verhält. Dies ist im Hinblick auf das nächste Kapitel von besonderer Bedeutung, da in dieser Arbeit die Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung im allgemeinen und nicht im Hinblick auf die speziellen Eigenschaften eines Lastbalancierungssystem hin untersucht werden sollen (Übertragbarkeit der erzielten Ergebnisse). Bevor jedoch die durchgeführten Meßreihen ausführlich diskutiert werden, werden die zur Evaluierung des Systems herangezogenen parallelen Anwendungen sowie die verwendeten Meßkonfigurationen vorgestellt.

6.1 Beschreibung der Anwendungen

Zur Evaluierung des Lastbalancierungsansatzes bzw. zur Untersuchung der Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung werden zwei existierende parallele Anwendungen mit sehr unterschiedlichem dynamischem Lastverhalten verwendet. Der Beschreibungsschwerpunkt liegt hier auf dem Lastverhalten der Anwendungen und nicht auf den algorithmischen Konzepten, da nur das Lastverhalten für einen dynamischen Lastbalancierer von Interesse ist.

6.1.1 Parallele Strömungssimulation

Die Lattice-Boltzmann-Hydrodynamik-Anwendung (kurz: Strömungs-Anwendung) ist eine enggekoppelte, parallele SPMD-Anwendung[1] [Pätz93]. Die Anwendung simuliert mit Hilfe des Lattice-Boltzmann-Verfahrens Konvektionsströmungen, d.h. Flüssigkeitsbewegungen in einem Schwerefeld, die durch Temperatur- und die damit verbundenen Dichteunterschiede angetrieben werden (vgl. Abbildung 30). Das Lattice-Botzmann-Verfahren ist ein gitterbasiertes Verfahren, bei dem sich die zu

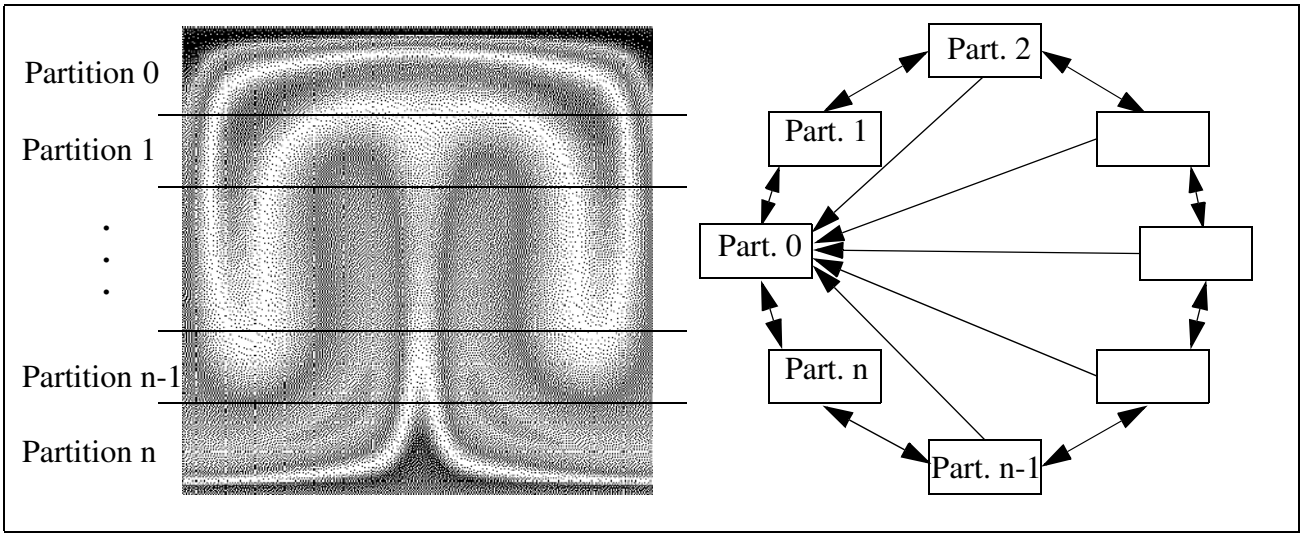


Abbildung 30: Partitionierung und Kommunikationsbeziehungen der Lattice-Anwendung.

simulierenden Teilchen nur entlang der Gitterlinien bewegen können (siehe Abbildung 31) und es in den Gitterpunkten zu Kollisionen kommen kann. Erfüllen die Teilchen und die Kollisionen bestimmte Bedingungen, so entspricht die Teilchenbewegung einer Bewegung, die auch mit Hilfe der Navier-Stoke-Gleichungen beschrieben werden kann.

Das Berechnungsgebiet ist ein zweidimensionales, regelmäßiges Rechteckgitter, das entlang einer Dimension in gleichgroße Partitionen aufgeteilt wird. Es handelt sich hierbei um eine adaptive Version des Verfahrens, da zur Berechnung zwei verschiedene Gitter mit unterschiedlicher Auflösung (Faktor 4) verwendet werden (siehe Abbildung 31(a)). Während die Prozesse, die auf den Partitionen *0* und *n* arbeiten, immer auf dem feinen Gitter iterieren müssen, können die restlichen Prozesse (1,..., n-1) in Abhängigkeit des Berechnungsergebnisses der letzten Iteration autonom zwischen den Gittern wechseln. Grenzen zwei Gitter mit unterschiedlicher Auflösung aneinander, wird mit Hilfe einer linearen Interpolation ein Übergang zwischen den Gittern ermittelt (siehe Abbildung 31(b)). Die Wahl des Berechnungsgitters hat dabei entscheidenden Einfluß auf das Lastverhalten eines Prozesses. Während ein Prozeß, der auf dem feinen Gitter arbeitet, innerhalb einer Iteration sein verfügbares Prozessorquantum nahezu vollständig ausnutzen kann, kann ein Prozeß, der auf dem groben Gitter arbeitet, bei dem betrachteten Zielsystem (vgl. Kapitel 4) maximal 50% seines Quantums nutzen, da sich die Prozesse pro Berechnungsiteration durchschnittlich zehn mal synchronisieren und die Anzahl der Gitterpunkte im groben Gitter viermal geringer ist als im feinen Gitter.

Aufgrund der Partitionierung des Berechnungsgebietes entlang einer Dimension ergibt sich als logische Kommunikationsstruktur ein Ring (vgl. Abbildung 30). Pro Iteration sendet und empfängt jeder Prozeß mindestens vier Nachrichten. Da die Anwendung zur Kommunikation synchrone Kommunikationsroutinen verwendet und sich jeder Prozeß innerhalb einer Iteration somit mehrfach mit seinen unmittelbaren Nachbarprozessen synchronisiert, handelt es sich um eine enggekoppelte Anwendung. Das Kommunikationsaufkommen pro Anwendungsprozeß hängt dabei nur von der Anzahl der Iterationen (Zeitschritte) ab, nicht von der Problemgröße bzw. der Anzahl der Anwendungsprozesse. Die Ausgabe des Berechnungsergebnisses erfolgt über den Prozeß, der die Partition *0* bearbeitet.

In Abbildung 32 sind die Bearbeitungszeiten der Anwendung in Abhängigkeit von verschiedenen Größen des feinen Berechnungsgitters und in Abhängigkeit von unterschiedlichen Prozessorzahlen

1. SPMD - Single Program Multiple Data

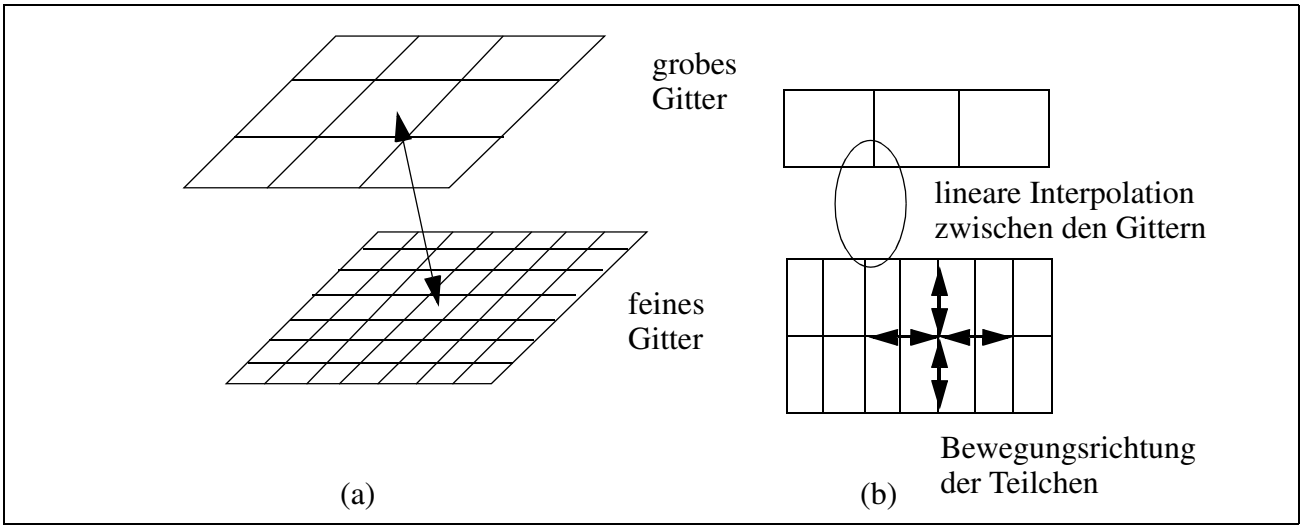


Abbildung 31: Gitteraufbau für die Lattice-Boltzman-Anwendung.

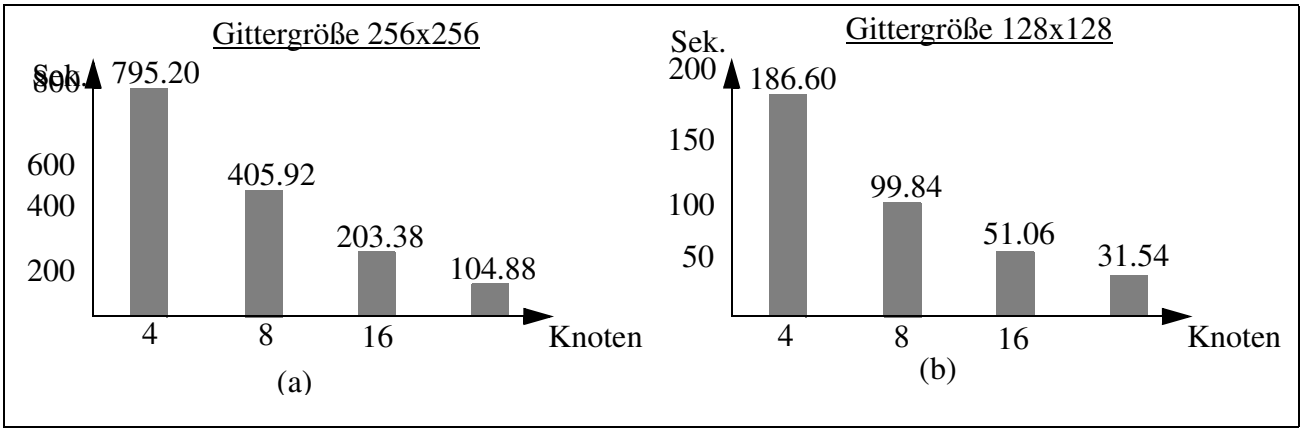


Abbildung 32: Laufzeitverhalten der Lattice-Boltzman-Anwendung.

dargestellt. Der Abbildung ist zu entnehmen, daß die Anwendung für ausreichend große Probleme nahezu linear skaliert.
Die Messungen wurden mit Hilfe des PaLaBer-Systems durchgeführt, wobei jeder Blattkomponente und somit jedem Rechenknoten des Parallelrechners nur ein Anwendungsprozeß zugeordnet wurde. Daraus folgt, daß keine dynamische Lastbalancierung durchgeführt wird. Lediglich die Plazierungskomponente bzw. der Kommunikationsmanager der Lastbalancierungsumgebung werden verwendet. Der nahezu lineare Speedup der parallelen Anwendung ist ein Indiz dafür, daß das Laufzeitsystem und insbesondere der Kommunikationsmanager der PaLaBer-Umgebung gut skalieren, da in den in Abbildung 32 dargestellten Meßläufen

- mit 4 Knoten insgesamt ca. 1400 Nachrichten,
- mit 8 Knoten insgesamt ca. 2800 Nachrichten,
- mit 16 Knoten insgesamt ca. 5600 Nachrichten und
- mit 32 Knoten insgesamt ca. 11200 Nachrichten

zwischen den Anwendungsprozessen ausgetauscht werden. Bei einer Gittergröße von 128x128, 32 Knoten und einer Gesamtlaufzeit von 31.54 Sekunden wird somit im Durchschnitt alle 3 Millisekunden eine Anwendungsnachricht über den Kommunikationsmanager der Lastbalancierungsumgebung geleitet.

6.1.2 Paralleles Raytracing

NoSpoort [Rich92] ist ein paralleler Raytracer, der nach dem Client-/Server-Paradigma parallelisiert ist. Das System verfügt über einen einfachen, anwendungsintegrierten Lastausgleichsmechanismus, da der Client bei der Verteilung der Aufträge die aktuelle Auslastung der Server berücksichtigt. Aufgrund dieser Eigenschaft ist die Anwendung in der Lage, die zur Verfügung stehenden Ressourcen effizient auszunutzen.
Zu Beginn der Berechnung teilt der Client-Prozeß (vgl. Abbildung 33) das zu berechnende Bild in *n+1* möglichst gleichgroße Partitionen (Bildzeilen) auf und weist jedem der *n* Server-Prozesse eine Bildpartition zu. Eine Bildpartition wird vom Client selbst bearbeitet. Sobald ein Server seine Bildpartition vollständig bearbeitet hat, fordert er vom Client-Prozeß einen weiteren Bearbeitungsauftrag an. Für den Fall, daß der Client-Prozeß seinen Bildausschnitt noch nicht vollständig berechnet hat, gibt er seine noch zu berechnenden Bildzeilen an den ersten Server ab, der erneut Last anfordert. Verfügt der Client-Prozeß hingegen über keine eigene Bildpartition mehr, fordert er einen anderen noch aktiven Server auf, die Hälfte seines noch zu generierenden Bildausschnittes an den unterbelasteten Server abzugeben. Die Rechenzeitunterschiede bei den Bildpartitionen ergeben sich dabei aus dem Szenenaufbau (Anzahl der Objekte im Bildausschnitt, Oberflächenbeschaffenheit der Objekte, Lichtverhältnisse, etc.)
Die Kommunikation zwischen den Server-Prozessen läuft dabei immer über den Client-Prozeß, weshalb die Struktur der Server-Prozesse einfach gehalten werden konnte. Nachdem alle Server-Prozesse ihre Berechnung beendet haben, senden sie ihren generierten Bildausschnitt an den Client-Prozeß, der das Berechnungsergebnis in korrekter Reihenfolge in eine Datei schreibt. In Abbildung 34 sind die Bearbeitungszeiten der Anwendung in Abhängigkeit verschiedener Bildgrößen bzw. Prozessorzahlen aufgeführt.
Aufgrund des anwendungsintegrierten Lastausgleichs verfügt diese Anwendung über ein sehr interessantes Lastverhalten, da sich das Lastverhalten eines einzelnen Anwendungsprozesses zur Laufzeit unvorhersehbar ändern kann. Verfügt ein Anwendungsprozeß über einen Berechnungsauftrag, kann er sein Prozessorquantum zu 100% ausnutzen, ein Prozeß ohne Berechnungsauftrag nimmt keinerlei Prozessorzeit in Anspruch. Da ein Server-Prozeß im Laufe der Verarbeitung mehrfach Bearbeitungsaufträge zugeteilt bekommen kann, bzw. mehrfach „leerlaufen“ kann, ist diese Anwendung ein interessanter Testfall für die Fähigkeit eines Lastbalancierers, auf die rasch wechselnden Lastprofile von Anwendungsprozessen reagieren zu können.

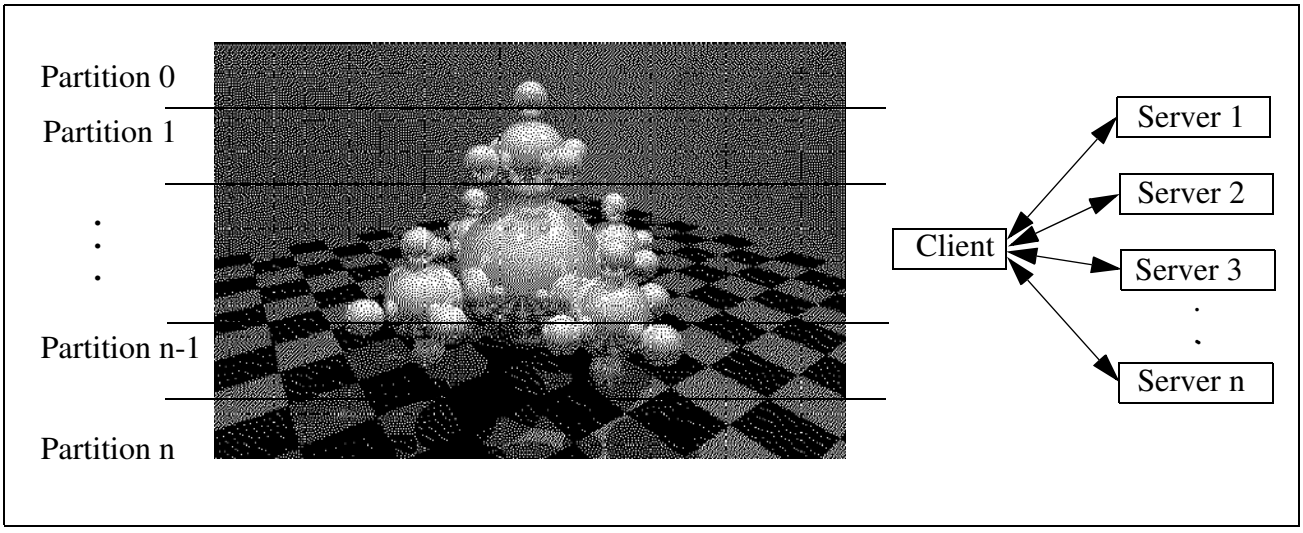


Abbildung 33: Partitionierung und Kommunikationsbeziehungen der Raytracing-Anwendung.

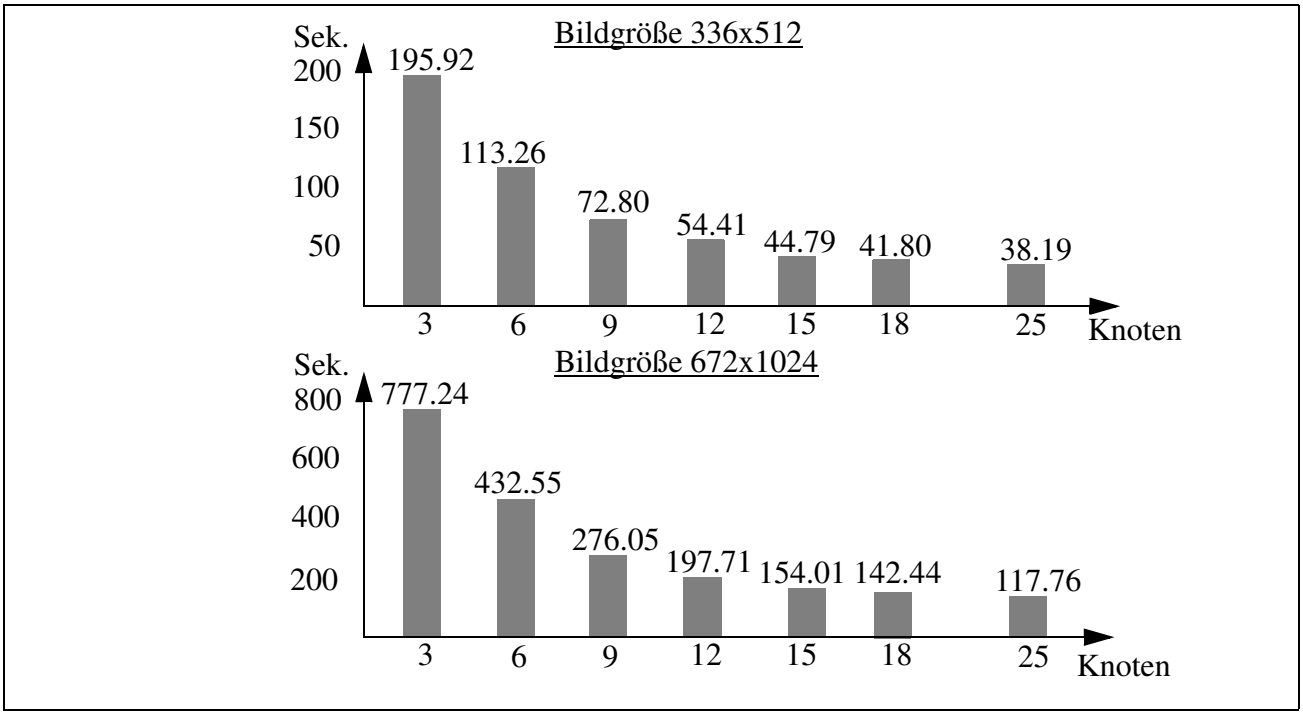


Abbildung 34: Laufzeitverhalten der Raytracing-Anwendung.

6.2 Beschreibung der Meßkonfigurationen

Für die Leistungsmessungen sowie für die anschließenden Untersuchungen in Kapitel 7 werden zwanzig verschiedene Konfigurationen, aufgeteilt in drei Konfigurationsgruppen, verwendet. Die Konfigurationsgruppe 1 (siehe Tabelle 20) dient zur Untersuchung, inwieweit das PaLaBer-System in der Lage ist, positiv auf die Laufzeit einer einzelnen parallelen Anwendung einzuwirken. In der Konfigurationsgruppe 2 (siehe Tabelle 21) wird jeweils eine Gruppe von parallelen Anwendungen gestartet und die Gesamtlaufzeit ermittelt. Mit Hilfe der letzten Konfigurationsgruppe (siehe Tabelle 22) wird untersucht, ob der hierarchische Lastbalancierer in der Lage ist, den Systemdurchsatz zu erhöhen.
Insgesamt unterscheiden sich die zwanzig Konfigurationen somit nicht nur im Hinblick auf die Anzahl der Blattkomponenten (6 bis 66), der inneren Komponenten (1 bis 3) sowie der Anwendungslast, sondern auch im Hinblick auf das Optimierungsziel. Mit der hierarchischen Lastbalancierungsumgebung PaLaBer werden zwei Optimierungsziele verfolgt:

1. Optimierung der Laufzeit einer einzelnen parallelen Anwendung
2. Optimierung des Systemdurchsatzes

Beide Optimierungskritierien stellen unterschiedliche Anforderungen an einen dynamischen Lastbalancierer. Um die Laufzeit einer einzelnen parallelen Anwendung zu optimieren, muß der Lastbalancierer versuchen, mit möglichst wenigen Lastverschiebungen eine gute und stabile Lastverteilung zu erreichen. Speziell bei enggekoppelten, parallelen Anwendungen kann jede Lastverschiebung zu einem temporären Stillstand der gesamten Anwendung führen.
Hingegen muß ein Lastbalancierer zur Steigerung des Systemdurchsatzes versuchen, die zur Verfügung stehenden Ressourcen bestmöglich auszulasten. Der temporäre Stillstand einer parallelen Anwendung ist dabei von untergeordneter Bedeutung, da die Ressourcen währenddessen von anderen aktiven Anwendung ausgelastet werden können.

Konfiguration	Anzahl der Blatt-komponenten	Anzahl der inneren Komponenten	Anwendung	Anzahl der Prozesse
1	6	1	Strömungs-Anwendung	8
2	12	1	Strömungs-Anwendung	16
3	24	1	Strömungs-Anwendung	32
4	6	1	Raytracing-Awendung	8
5	9	1	Raytracing-Anwendung	15
6	18	1	Raytracing-Anwendung	30

Tabelle 20: Konfigurationsgruppe 1- Laufzeitoptimierung.

6.2.1 Beschreibung der Konfigurationsgruppe 1

Die Leistungsfähigkeit des Lastbalancierungsansatzes in bezug auf die Laufzeitoptimierung einer einzelnen Anwendung wird mit Hilfe der Konfigurationsgruppe 1 (siehe Tabelle 20) untersucht. Als Leistungskriterium für diese Untersuchung wird die Gesamtlaufzeit der Anwendungen herangezogen. In den Konfigurationen 1 bis 3 wird versucht, die Laufzeit der Strömungs-Anwendung mit unterschiedlicher Prozeßanzahl zu optimieren. Da bei sämtlichen Messungen die Problemgröße konstant gehalten wird, wird dabei mit zunehmender Prozeßanzahl das Verhältnis von Rechenzeit zu Kommunikationsaufkommen ungünstiger.
Mit Hilfe der Konfigurationen 4 bis 6 wird eine vergleichbare Untersuchung an der Raytracing-Anwendung durchgeführt. Auch hier wird die Problemgröße konstant gehalten, wobei der anwendungsintegrierte Lastausgleichsmechanismus dafür sorgt, daß mit steigender Prozeßanzahl die Arbeitslast immer gleichmäßiger auf die Server-Prozesse verteilt wird und somit die Lastungleichgewichte immer geringer werden. In den Konfigurationen 4 bis 6 werden jeder Blattkomponente mehr Anwendungsprozesse zugeteilt als in den Konfigurationen 1 bis 3, da bei der Raytracing-Anwendung in der Endphase der Berechnung die einzelnen Serverprozesse „leerlaufen“. Damit der Lastbalancierer auch in dieser Verarbeitungsphase noch positiv auf die Lastsituation einwirken kann, wurde die Anzahl der Anwendungsprozesse in den Konfigurationen 4 bis 6 erhöht.

6.2.2 Beschreibung der Konfigurationsgruppe 2

In den Kofigurationen 7 bis 17 werden jeweils eine Gruppe von Anwendungen, nachfolgend als Anwendungspulk beizeichnet, gestartet und die Gesamtlaufzeit des Pulks gemessen. Diese Lastkonfiguration entspricht z.B. dem interaktiven Betrieb von Parallelrechnern bzw. Workstationnetzen. Für einen dynamischen Lastbalancierer ist dieses Lastszenario von besonderem Interesse [DoHB95] [EaLZ88][TiWi84], da in ihm sämtliche Phasen der Verarbeitung (ansteigende Parallelität in der Startphase, Vollast während der eigentlichen Verarbeitung und abnehmende Parallelität in der Endphase) enthalten sind. Insbesondere im Hinblick auf die Untersuchungen im nächsten Kapitel sind diese Konfigurationen interessant, da mit ihrer Hilfe die Auswirkungen der höheren Informationsebenen auf die verschiedenen Verarbeitungsphasen ausgetestet werden können.
Die Konfigurationen in Tabelle 21 wurden in zwei Untergruppen aufgeteilt. Während in den Konfigurationen 7 bis 13 (Anwendungspulk I) sämtliche Anwendungsprozesse unmittelbar hintereinander gestartet werden, ist die Anzahl der Anwendungsprozesse in den Konfigurationen 14 bis 17 (Anwendungspulk II) so groß, daß zu Beginn nur eine Teilmenge der Anwendungen gestartet wird.

Konfiguration	Anzahl der Blatt-komponenten	Anzahl der inneren Komponenten	Anwendung	Anzahl der Prozesse
Anwendungspulk I				
7	24	1	2xStrömungs-Anwendung	16
8	18	1	2xRaytracing-Anwendung	16
9	18	1	1xStrömungs-Anwendung 1xRaytracing-Anwendung	16 8
10	24	1	1xStrömungs-Anwendung 2xRaytracing-Anwendung	16 8
11	36	1	2xStrömungs-Anwendung 2xRaytracing-Anwendung	16 8
12	48	2	3xStrömungs-Anwendung 2xRaytracing-Anwendung	16 8
13	66	3	4xStrömungs-Anwendung 2xRaytracing-Anwendung	16 18
Anwendungspulk II				
14	9	1	2xRaytracing-Anwendung	12
15	12	1	2xStrömungs-Anwendung	16
16	18	1	2xStrömungs-Anwendung 2xRaytracing-Anwendung 1xRaytracing-Anwendung	16 16 10
17	36	1	3xStrömungs-Anwendung 2xRaytracing-Anwendung 2xRaytracing-Anwendung	16 8 16

Tabelle 21: Konfigurationsgruppe 2- Anwendungspulks

Der Start der verbleibenden Anwendungen wird dann durch die Terminierung einer bereits gestarteten Anwendung initiiert.
Um die Skalierbarkeit der inneren Komponenten zu testen, werden in den Konfigurationen 9 bis 11 bis zu 36 Blattkomponenten einer inneren Komponente zugeordnet. Der Lastausgleich über Teilbaumgrenzen hinweg wird mit den Konfigurationen 12 und 13 evaluiert.

6.2.3 Beschreibung der Konfigurationsgruppe 3

Die Leistungsfähigkeit des Lastbalancierungsansatzes in bezug auf die Durchsatzoptimierung wird mit Hilfe der Konfigurationen 18 bis 20 in Tabelle 22 untersucht. Als Leistungskriterium für die Durchsatzoptimierung dient die Anzahl der Barbeitungsaufträge pro Zeiteinheit. Diese Durchsatzdefinition ist auf die Konfiguration 18 in einfacher Weise anwendbar, wenn man eine Iteration auf dem Berechnungsgitter als Bearbeitungsauftrag definiert. Die Gesamtlaufzeit der Anwendung beträgt 10 Minuten.

Konfiguration	Anzahl der Blatt-komponenten	Anzahl der inneren Komponenten	Anwendung	Anzahl der Prozesse
18	24	1	1xStrömungs-Anwendung	32
19	18	1	5xRaytracing-Anwendung	30
20	22	1	1xStrömungs-Anwendung	16
			13xRaytracing-Anwendung	16

Tabelle 22: Konfigurationsgruppe 3- Durchsatz

Für die Konfiguration 19 ist die Definition eines geeigneten Durchsatzmaßes schwieriger, da es bei der Raytracing-Anwendung keinen Awendungsprozeß gibt, der Kenntnis darüber hat, wieviele Bildpunkte bzw. -zeilen bereits berechnet wurden. Als alternatives Maß für den Systemdurchsatz wird deshalb die Zeitspanne gemessen, in der das System sich mit Hilfe der fünf Raytracing-Anwendungen in einem ausgelasteten Zustand befand. Dazu wird der Startzeitpunkt der ersten Anwendung und der Startzeitpunkt der fünften Anwendung erfaßt (siehe Abbildung 35). Durch diese Messung wird der Zeitpunkt ermittelt, an dem das System nach der Abarbeitung der ersten vier Raytracing-Anwendungen wieder bereit ist, einen weitern Auftrag zu bearbeiten.
Für die Durchsatzermittlung der Konfiguration 20 besteht die Problemstellung, zwei Anwendungen mit unterschiedlichen Durchsatzmaßen miteinander zu vergleichen. Dazu wurden die Laufzeiten der einzelnen Anwendungen ermittelt und verglichen. Das daraus gewonnene Gewichtungsverhältnis ist 1:9, d.h. eine Bildberechnung der Raytracing-Anwendung entspricht dem Aufwand von 9 Berechnungsiterationen der Strömungsanwendung auf dem Gitter (vgl. Abbildung 36). Um die Meßgenauigkeit für die Raytracing-Anwendung noch weiter zu erhöhen, werden in der Konfiguration 20 nicht nur die Start- bzw. Endzeiten der Raytracing-Anwendung ermittelt, sondern auch der Zeitpunkt, an dem der Client-Prozess keine weiteren Bearbeitungsaufträge mehr zu verteilen hat. Bei der gewählten Szenenbeschreibung geschieht dies etwa zur Hälfte der Bearbeitungszeit. Die Gesamtlaufzeit der Konfiguration 20 beträgt 260 Sekunden.
Da die Durchsatzmaße, insbesondere das Maß für die Konfiguration 20, relativ grob sind, werden diese Konfigurationen nur dann zur Auswertung der Untersuchungen herangezogen, wenn die erzielten Durchsatzsteigerungen deutlich über den Meßungenauigkeiten liegen. Für die Konfiguration 18 liegt die Meßungenauigkeit bei einer Iteration, weshalb die erzielten Ergebnisse erst bei einem Unterschied von 5 Iterationen gegenüber dem gewählten Referenzlauf Verwendung finden.

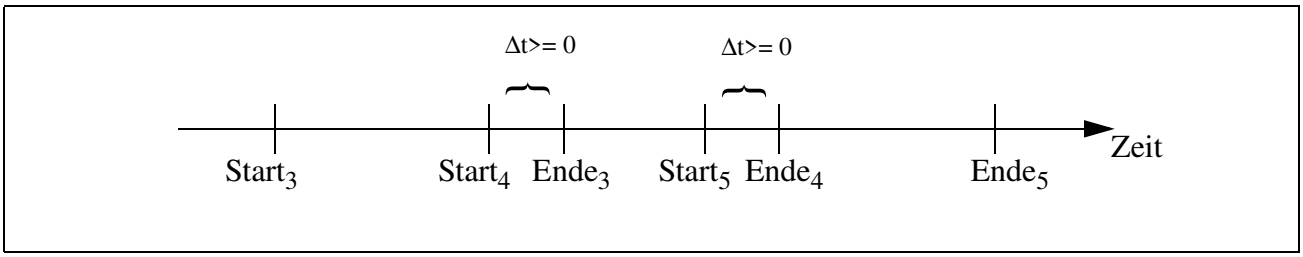


Abbildung 35: Durchsatzbestimmung für die Konfiguration 19.

6.3 Ergebnisse der durchgeführten Untersuchungen

In diesem Abschnitt werden die erzielten Untersuchungsergebnisse in bezug auf die Leistungsfähigkeit der dynamischen Lastbalancierungsumgebung PaLaBer vorgestellt. Dazu werden die mit Hilfe des dynamischen Lastbalancierungssystem PaLaBer erzielten Ergebnisse mit denen eines Referenzlaufes ohne dynamische Lastbalancierung verglichen. Für den Referenzlauf wurde nur

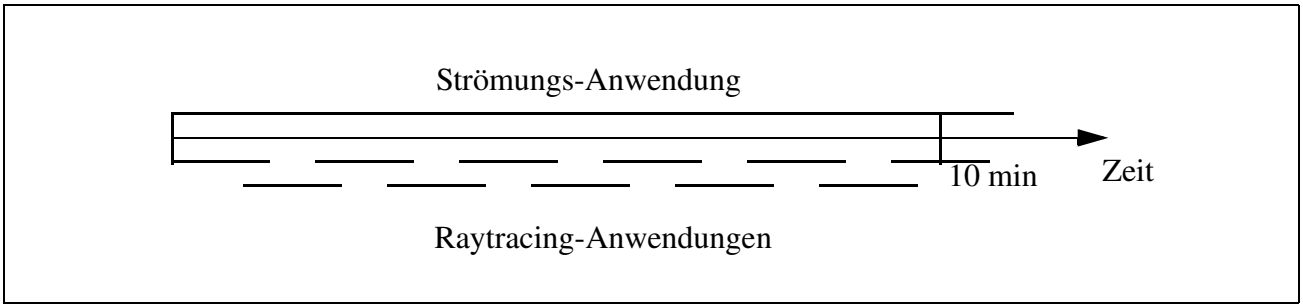


Abbildung 36: Durchsatzbestimmung für die Konfiguration 20.

der Clustering-Algorithmus der Plazierungskomponente und die Funktionalität des Kommunikationsmanagers des PaLaBer-Systems verwendet.
Zuvor wird jedoch das graphische Auswertungswerkzeug *PaStatTool* vorgestellt, das für die Auswertung einzelner Anwendungsläufe herangezogen wird.

6.3.1 Beschreibung des graphischen Auswertungswerkzeuges PaStatTool

Jede Blattkomponente der hierarchischen Lastbalancierungsumgebung PaLaBer erfaßt in regelmäßigen Zeitintervallen die knotenbezogenen Lastwerte. Diese Lastwerte werden, im Hinblick auf eine nachfolgende Analyse des Anwendungslaufes, von jeder Blattkomponente in einer eigenen Log-Datei erfaßt. Mit Hilfe des Auswertungswerkzeuges PaStatTool können diese Meßwerte in einfacher Weise graphisch veranschaulicht werden (siehe Abbildung 37). Die Lastsituation jeder Blattkomponente wird dabei in drei Ausgabefenstern dargestellt. Im obersten Fenster ist die Anzahl der Anwendungsprozesse abgebildet. Dabei wird zwischen bereits gestarteten und noch zwischengespeicherten Anwendungspozessen unterschieden. Dadurch ist es in einfacher Weise möglich zu erkennen, ob ein Anwendungsprozeß zum Zweck des Lastausgleichs zwischengespeichert oder bereits gestartet wurde. Im mittleren Fenster wird die jeweilige Prozessorauslastung in Prozent angegeben. Im untersten Fenster wird in Form eines einfachen Strichcodes angegeben, weshalb die Lastdaten erfaßt und in die Log-Datei abgespeichert wurden. Als mögliche Ereignisse kommen dabei in Frage: Prozeßstart, Prozeßmigration, Prozeßterminierung sowie die periodische Erfassung der knotenbezogenen Lastwerte. Im weiteren Verlauf wird jedoch auf die Darstellung des Strichcodes verzichtet.

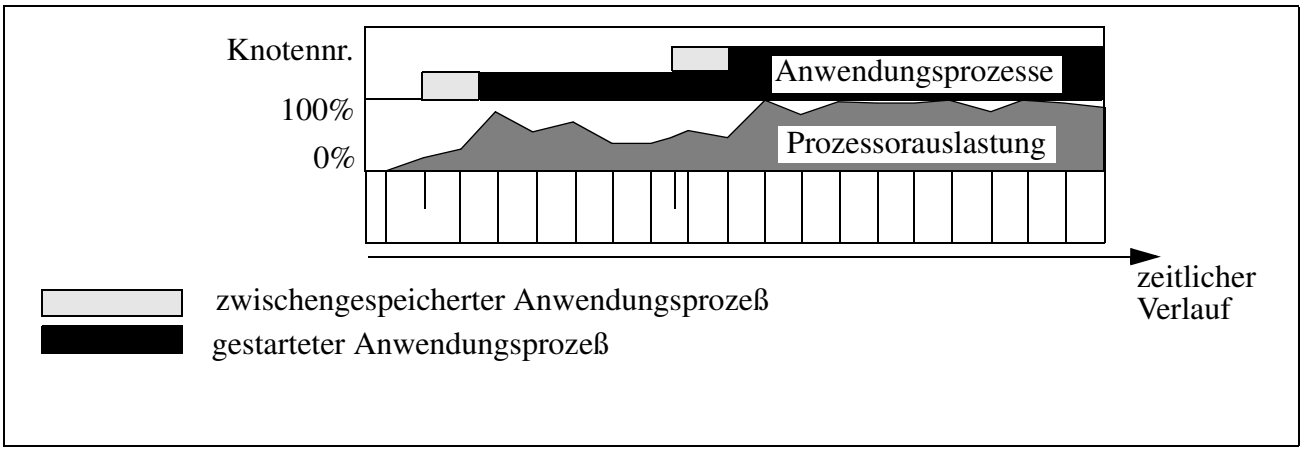


Abbildung 37: Beschreibung des Auswertungswerkzeuges PaStatTool.

Konfiguration	Referenzlauf	Dynamische Lastbalancierung unter Verwendung von statischer Plazierungsinformation		Laufzeit-differenz in %
	Laufzeit in Sek.	Laufzeit in Sek.	Migrationen	
1	647.95	450.27	11	31
2	328.76	241.66	11	26
3	181.56	171.37	24	6
4	413.45	321.70	2	22
5	261.88	225.77	5	14
6	176.39	156.59	18	11
Prozentuale Laufzeitgewinne liegen im Bereich von **6%** bis **31%**				

Tabelle 23: Laufzeitoptimierung einer einzelnen Anwendung - Ebene 0.

6.3.2 Laufzeitoptimierung

Mit den Konfigurationen 1 bis 6 wird die Fähigkeit des PaLaBer-Systems zur Optimierung der Laufzeit einer einzelnen Anwendung untersucht.
In den Konfigurationen 1 bis 3 wird die enggekoppelte Lattice-Anwendung mit unterschiedlichen Prozeßanzahlen ausgetestet.
Das Verhältnis von Rechenzeit zu Kommunikationsaufkommen wird dabei mit zunehemender Prozessanzahl zunehmend ungünstiger (vgl. Tabelle 24). Während sich die Anwendungsprozesse in Konfiguration 1 durchschnittlich alle 1,8 Sekunden synchronisieren, verringert sich die Zeitspanne zwischen zwei Synchronisationspunkten bei Konfiguration 3 auf 0,5 Sekunden. Vergleicht man die Synchronisationszeiten mit der Zeitspanne, die das Betriebssystem benötigt, um im Falle einer Prozeßmigration auf dem Zielknoten einen neuen Prozeß anzulegen (Freezing-Time), stellt man fest, daß bei der Konfiguration 3 jede Prozeßmigration zu einem temporären Stillstand von einzelnen, evtl. sogar von allen Prozessen der parallelen Anwendung führt. Hingegen ist bei der Konfiguration 1 die Differenz zwischen Synchronisationszeiten und der Freezing-Time so groß, daß eine Prozeßmigration nicht in jedem Fall zu einem Stillstand einzelner Prozesse oder sogar einer gesamten Anwendung führt. Ausgehend von diesen Tatbeständen ist es nicht überraschend, daß die prozentualen Laufzeitgewinne bei der Konfiguration 1 mit 31% am größten waren und mit zunehmender Prozeßanzahl auf 6% bei Konfiguration 3 sinken.
Noch deutlicher wird der Sachverhalt, wenn man die Strömungs-Anwendung in ihre verschiedenen Verarbeitungsphasen aufteilt und mit der Dauer der Freezing-Time vergleicht. Die gesamte Lastverhalten der Stömungsanwendung läßt sich beschreiben mit *n * [Gitteriteration],* wobei sich eine Gitteriteration darstellen läßt als:

$$n_{iter} * [\, CPU_{grobesGitter} + CPU_{zusatzFeinesGitter} + CPU_{Leerlauf} + Komm]$$

mit

n_{iter}	= Anzahl der Synchronisationspunkte innerhalb einer Gitteriteraton
$CPU_{grobesGitter}$	= Berechnungsaufwand auf dem groben Gitter
$CPU_{zusatzFeinesGitter}$	= zusätzlicher Berechnungsaufwand auf dem feinen Gitter
$CPU_{Leerlauf}$	= Leerlaufzeit eines Prozesses, der auf dem groben Gitter arbeitet
Komm	= Kommunikationsaufwand innerhalb einer Synchronisationsphase

Konf.	Anwendung (Prozesse)	durchschnittliche Synch.zeiten	Freezing-Time	Differenz
1	Strömungs-Anwendung (8)	1.8 Sek.	0.7-0.8 Sek.	1.0-1.1 Sek.
2	Strömungs-Anwendung (16)	0.9 Sek.		0.1-0.2 Sek.
3	Strömungs-Anwendung (32)	0.5 Sek.		(-0.2) - (-0.3) Sek.

Tabelle 24: Verhältnis von Synchronisationszeiten und Freezing-Time.

Da die Berechnungsaufwände zwischen dem groben Gitter und dem feinen Gitter sich um den Faktor 4 unterscheiden, ergibt sich zwischen den Größen $CPU_{grobesGitter}$ und $CPU_{zusatzFeinesGitter}$ ein Verhältnis von 1:3, desweiteren ergibt sich ebenfalls ein Verhältnis von 1:3 zwischen den Größen $CPU_{grobesGitter}$ und $CPU_{Leerlauf}$, da nur ein Prozess, der auf dem groben Gitter arbeitet, eine Leerlaufphase aufweist und diese der Zeit $CPU_{zusatzFeinesGitter}$ entsprechen muß. Unter der Annahme, daß die Zeitdauer für die Kommunikation innerhalb einer Synchronisationsphase vernachlässigt werden kann, kommt man zu den in Tabelle 25 dargestellten Zeitdauern. Da das hierarchische Lastbalancierungssystem PaLaBer bei der Auswahl eines Migrationskandidaten immer sicherzustellen versucht, daß der Quellknoten der Migration auch nach erfolgter Migration noch ausgelastet ist, wird bei der Strömungs-Anwendung i.d.R. immer ein Anwendungsprozeß migriert, der auf dem groben Gitter arbeitet. Da bei der Konfiguration 1 die Leerlaufphase innerhalb einer Synchronisationsphase durchschnittlich 1.2 Sekunden dauert, führt in diesem Fall eine Migration nicht zwangsläufig zu einem temporären Stillstand der Anwendung bzw. einzelner Anwendungsprozesse. Hingegen sind bei den Konfigurationen 2 und 3 die Leerlaufphasen kürzer als die Migrationszeiten, weshalb bei diesen Konfigurationen ein migrationsbedingter temporärer Stillstand der Anwendung eintritt.
Ein vergleichbares Ergebnis ist in den Konfigurationen 4 bis 6 zu erkennen. In diesen Konfigurationen wird der parallele Raytracer mit unterschiedlichen Prozeßanzahlen ausgetestet. Bei dieser Anwendung spielt das Kommunikationsaufkommen allerdings nur eine untergeordnete Rolle. Mit abnehmendem Prozeßgranulat ist jedoch der anwendungsintegrierte Lastausgleichsmechanismus immer besser in der Lage, die Arbeitslast gleichmäßig auf die Server-Prozesse zu verteilen. In Tabelle 26 sind für die Konfigurationen 4 bis 6 die Gesamtlaufzeiten, die Zeitdauer mit maximaler Prozessorauslasung durch einen Anwendungsprozesse sowie die Zeitdauer mit minimaler Prozessorauslastung durch einen Anwendungsprozesses aufgeführt. Dieser Tabelle ist zu entnehmen, daß mit steigender Prozeßanzahl bei konstant gehaltener Problemgröße die Prozessorleerlaufzeit nicht nur in absoluten Zahlen abnimmt, sondern auch das Verhältnis von Gesamtlaufzeit und Leerlaufzeit ständig besser wird. Dementsprechend nehmen die erzielten Laufzeitgewinne von 22% bei Konfiguration 4 auf 11% bei Konfiguration 6 ab.

Insgesamt wird in allen sechs Konfigurationen ein zum Teil deutlicher Laufzeitgewinn (6%-31%) erzielt. Somit ist die Prozeßmigration ungeachtet ihrer Kosten [DoHB95][EaLZ88] durchaus ein geeigneter Lasttransfermechanismus für parallele Anwendungen. Das Prozeßgranulat hat hierbei jedoch einen erheblichen Einfluß auf den erzielbaren Leistungsgewinn mittels der Migration eines bereits laufenden Prozesses.

Anwendung (Prozesse)	$CPU_{grobesGitter}$	$CPU_{zusatzFeinesGitter}$ $CPU_{Leerlauf}$
Strömungs-Anwendung (8)	0.6 Sek.	1.2 Sek.
Strömungs-Anwendung (16)	0.3 Sek.	0.6 Sek.
Strömungs-Anwendung (32)	0.17 Sek.	0.33 Sek.

Tabelle 25: Dauer der Verarbeitungsphasen.

Anwendung (Prozesse)	Gesamtlaufzeit	max. CPU-Auslastung	min. CPU-Auslastung
Raytracing-Anwendung (8)	274 Sek.	254 Sek.	78 Sek.
Raytracing-Anwendung (15)	157 Sek.	127 Sek.	60 Sek.
Raytracing-Anwendung (30)	126 Sek.	100 Sek.	50 Sek.

Tabelle 26: Verhältnis von Leerlaufzeiten zu Gesamtlaufzeiten.

6.3.3 Anwendungspulk I

Zur Untersuchung der Leistungsfähigkeit in bezug auf die Fähigkeit des Lastbalancierers, die Laufzeit einer Gruppen von Anwendungen zu verkürzen, werden drei unterschiedliche Lastkombinationen (siehe Tabelle 21 und Tabelle 27) verwendet:

1. Menge von enggekoppelten, parallelen Anwendungen (Konfiguration 7)
2. Menge von losegekoppelten, parallelen Anwendungen (Konfiguration 8)
3. Kombinationen aus lose- und enggekoppelten Anwendungen (Konfigurationen 9 bis 13)

Anhand der Laufzeiten der Konfigurationen 9, 10 und 11 läßt sich die Skalierbarkeit der inneren Komponenten gut erkennen. Speziell der Laufzeitgewinn von Konfiguration 11 zeigt, daß eine innere Komponente problemlos 36 Blattkomponenten auch bei sehr heterogener Arbeitslast verwalten kann. Die Skalierbarkeit der inneren Komponenten des Lastbalancierungsbaumes resultiert in erster Linie aus der Tatsache, daß die rechenzeitintensiven Protokollteile völlig verteilt innerhalb der Blattkomponentenebene realisiert sind.
Um den Lastaustausch zwischen verschiedenen Teilbäumen der Lastbalancierungshierarchie zu testen, werden in den Konfigurationen 12 und 13 zwei bzw. drei innere Komponenten verwendet. Obwohl der Lastaustausch zwischen den Teilbäumen mehr Verwaltungsaufwand erfordert als der Lastaustausch innerhalb eines Teilbaumes, werden auch in diesen Konfigurationen signifikante Laufzeitverbesserungen erzielt. Dies beruht einerseits auf der Tatsache, daß speziell die zeitintensiven Protokolle völlig verteilt in den Blattkomponenten realisiert sind. Andererseits versucht jede innere Komponente zuerst die Lastungleichgewichte durch Lastverschiebungen innerhalb ihres Teilbaumes auszugleichen. Erst wenn dies nicht möglich ist, wird eine Lastverschiebung über Teilbaumgrenzen hinweg initiiert. So werden in der Konfiguration 12 durchschnittlich nur 10% aller Prozeßmigrationen über Teilbaumgrenzen hinweg durchgeführt. In der Konfiguration 13 erhöht sich dieser Anteil dann auf 30%.

Konfiguration	Referenzlauf	Dynamische Lastbalancierung unter Verwendung von statischer Plazierungsinformation		Laufzeit-differenz in %
	Laufzeit in Sek.	Laufzeit in Sek.	Migrationen	
7	411.21	314.93	44	23
8	260.49	246.22	15	5
9	484.39	330.55	34	32
10	454.66	336.84	30	26
11	472.29	344.75	61	27
12	463.78	349.30	88	25
13	413.98	336.09	114	19
Prozentuale Laufzeitgewinne liegen im Bereich von **5%** bis **32%**				

Tabelle 27: Gesamtlaufzeiten - Anwendungspulk I

Anhand der Konfiguration 9, bei der sowohl eine Strömungs-Anwendung mit 16 Prozessen als auch eine Raytracing-Anwendung mit 8 Prozessen gestartet wird, soll aufgezeigt werden, wie die hierarchische Lastbalancierungsumgebung bei einer heterogenen Arbeitslast Laufzeitgewinne erzielt. In der Abbildung 38 ist dazu das Laufzeitprofil der Konfiguration mit Hilfe des Auswertungswerkzeuges PaStatTool dargestellt, wobei bei diesem Laufzeitprofil keine dynamische Lastbalancierung stattfand. Da die Laufzeitprofile der Knoten 3 bis einschließlich 13 vergleichbar sind, wurden nicht alle Knoten abgebildet. Wie dieser Abbildung zu entnehmen ist, treten bei dieser Konfiguration speziell in der Endphase sehr deutliche Lastungleichgewichte auf, die durch die Anwendungsprozesse auf dem Knoten 1 verursacht werden. Vergleicht man dieses Lastprofil mit dem Profil in Abbildung 39 (mit dynamischer Lastbalancierung), stellt man fest, daß nicht nur die Leerlaufzeiten am Ende der Bearbeitung deutlich geringer sind als im Konfigurationslauf ohne dynamische Lastbalancierung, sondern daß auch die Ressourcenauslastung während des Laufes deutlich höher ist. Insgesamt konnte mit Hilfe der dynamischen Lastbalancierungsumgebung PaLaBer die Laufzeit der Konfiguration 9 um 32% verringert werden.

6.3.4 Anwendungspulk II

Die Meßergebnisse des Anwendungspulkes II (siehe Tabelle 28) werden hier nur zur Vollständigkeit angegeben, da diese Konfigurationen erst im Zusammenhang mit den höheren Informationensebenen von Bedeutung sind. Denn wie in Kapitel 7 noch ausführlich aufgezeigt wird, kann ein dynamischer Lastbalancierer die Informationen der höheren Ebenen speziell in den Start- bzw. Endphasen eines Anwendungslaufes verwenden. An dieser Stelle soll nur darauf hingewiesen werden, daß auch ein Lastbalancierer der Ebene 0 bei den Konfigurationen 14 bis 17 in der Lage ist, zum Teil sogar deutliche Laufzeitgewinne von bis zu 26% zu erzielen.

6.3.5 Durchsatzoptimierung

In Tabelle 29 sind die Meßergebnisse in Bezug auf die Fähigkeit des PaLaBer-Systems, den Systemdurchsatz zu erhöhen, angegeben. Für die Konfiguration 18 wird der Durchsatz dabei durch die Anzahl der Iterationen auf dem Berechnungsgitter innerhalb von 10 Minuten angegeben.

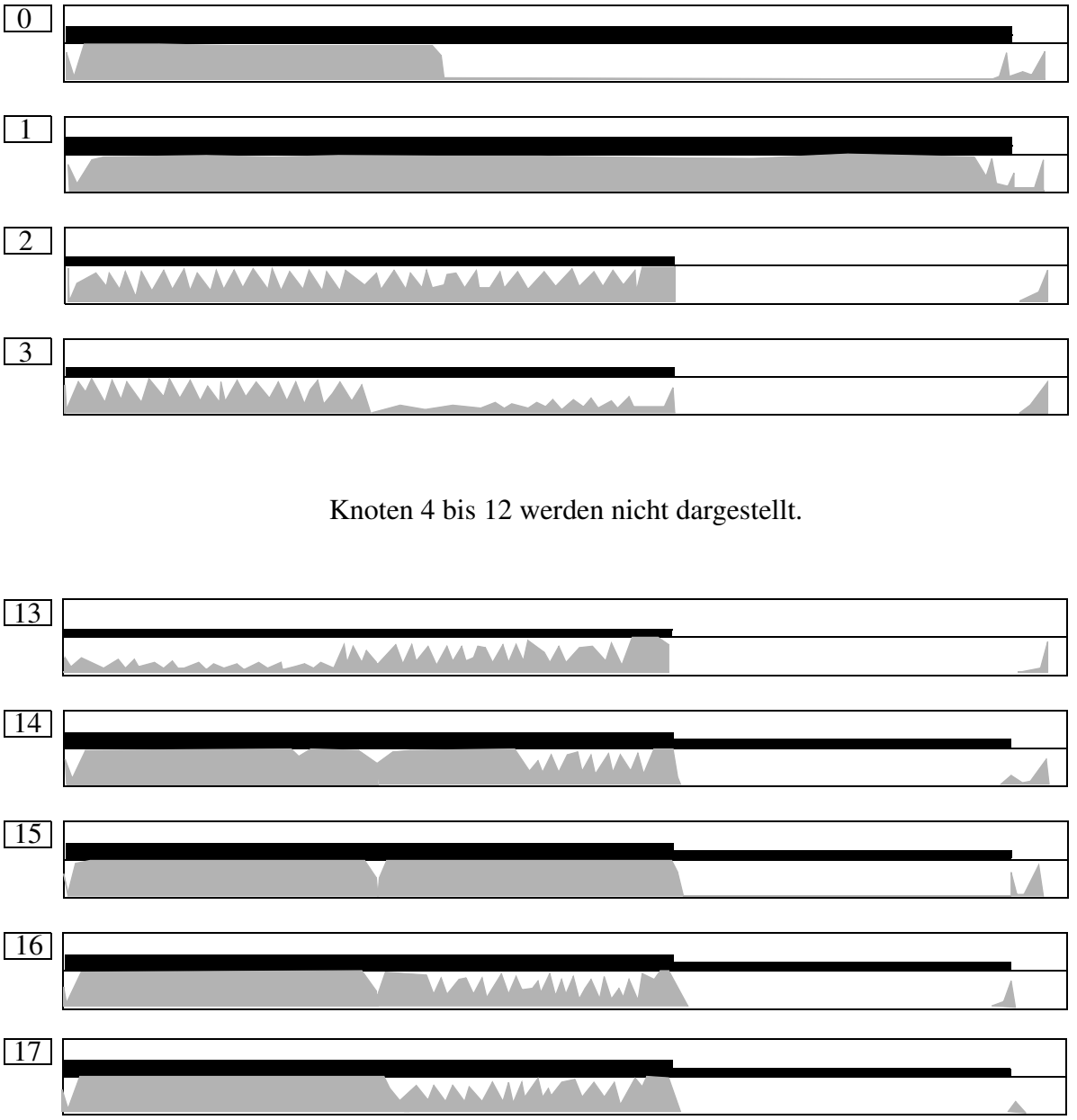


Abbildung 38: Konfiguration 9 - ohne dynamische Lastbalancierung.

Für die Konfiguration 19 wird anstelle der klassischen Durchsatzdefinition die Zeitdauer ermittelt, in dem sich das System in einem ausgelasteten Zustand befand. Dazu wird der Startzeitpunkt der fünften Raytracing-Anwendung ermittelt. Um sicherzustellen, daß das vorzeitige Starten tatsächlich zu einer besseren Systemauslastung und somit zu einem besseren Durchsatz führt, wird zusätzlich noch die Gesamtlaufzeit dieser Konfiguration angegeben.
Zur Ermittlung der abgearbeiteten Aufträge in Konfiguration 20 wurde das bereits oben angeführte Gewichtungsverhältnis von 1:9 verwendet.
Wie der Tabelle zu entnehmen ist, kann das PaLaBer-System Durchsatzsteigerungen von bis zu 21% erzielen. Die Begründungen für die Leistungssteigung bei den Konfigurationen 18 und 19 sind dabei vergleichbar mit den Begründungen für die Konfigurationen 1 bis 6, da es sich hierbei um

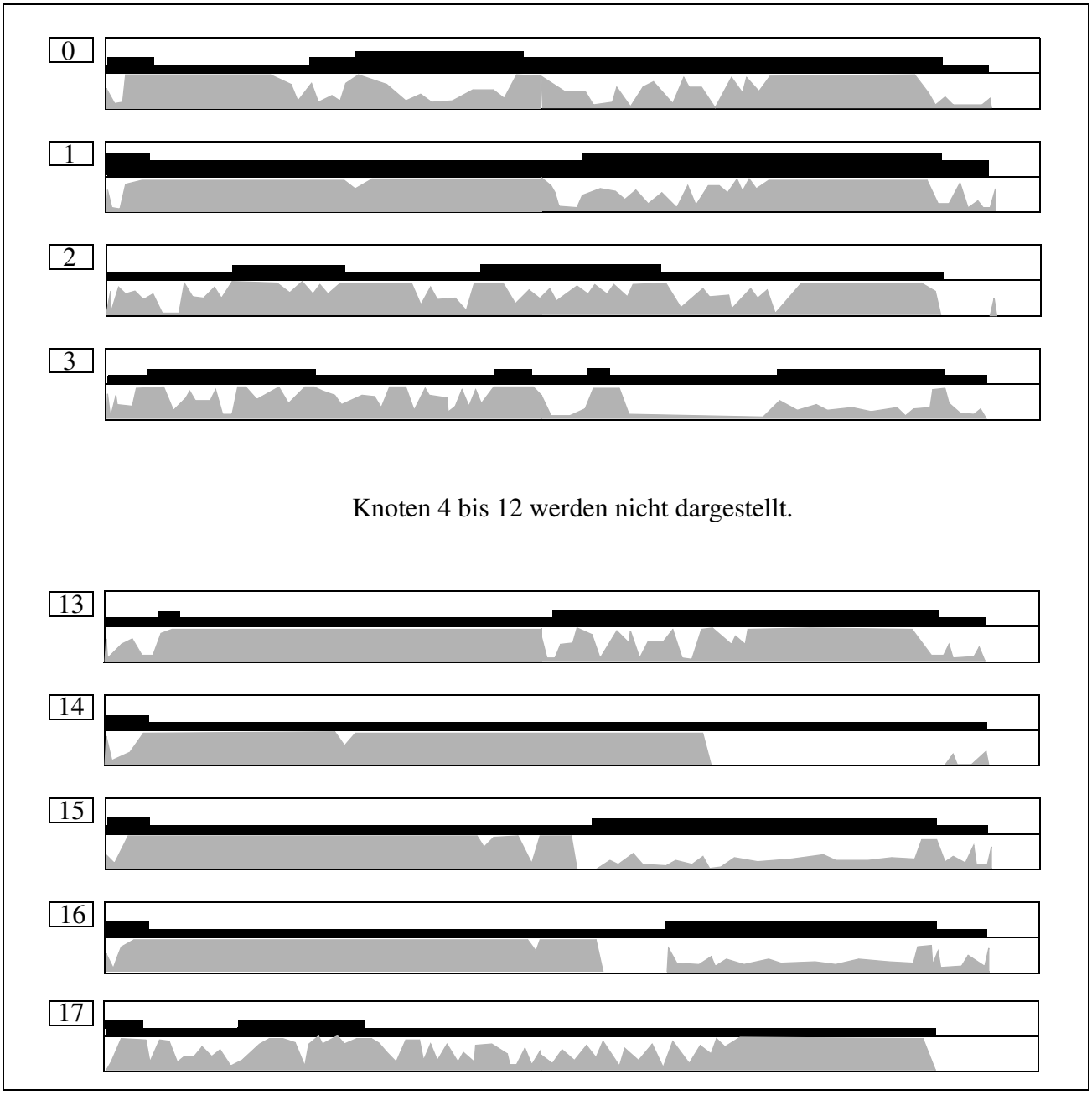


Abbildung 39: Konfiguration 9 - mit dynamischer Lastbalancierung.

Konfigurationen mit vergleichbaren Laufzeitprofilen handelt, die lediglich bezüglich eines unterschiedlichen Optimierungskriterium ausgewertet wurden.

Deutlich geringer als im Vergleich zu den Konfigurationen 18 und 19 fallen hingegen die Leistungssteigerungen für die Konfiguration 20 aus. Dies liegt jedoch an den speziellen Eigenschaften dieser Konfiguration. Die Konfiguration 20 besteht aus einer parallelen Strömungs-Anwendung, die während des gesamten Anwendungslaufes (260 Sekunden) aktiv ist und 13 Raytracing-Anwendungen, die sukzessiv gestartet werden, d.h. die Terminierung einer Raytracing-Anwendung initiiert den Start der nächsten. Da die Laufzeit einer einzelnen Raytracing-Anwendung bei dieser Konfiguration relativ kurz ist (ca. 30 Sekunden), kann bei der Konfiguration 20 die Plazierungskomponente des PaLaBer-Systems bereits deutlich auf evtl. auftretende

Lastungleichgewichte reagieren, weshalb der dynamische Lastbalancierer nur eine Durchsatzsteigerung von 2% erzielen kann.

Insgesamt können mit Hilfe des dynamischen Lastbalancierungssystem PaLaBer Leistungssteigerungen zwischen 2% und 32% erzielt werden. Insbesondere konnte in allen 20 Konfigurationen mit Hilfe des PaLaBer-Systems eine Leistungssteigerung (Laufzeitverkürzung, Durchsatzsteigerung) erzielt werden, obwohl es sich hierbei nur um einen Lastbalancierer der Ebene 0 handelt. In den nachfolgenden Abschnitten wird nun auf verschiedene Einzelaspekte des PaLaBer-Systems ausführlicher eingegangen.

Konfiguration	**Referenzlauf**	**Dynamische Lastbalancierung unter Verwendung von statischer Plazierungsinformation**		**Differenz in %**
	Laufzeit in Sek.	**Laufzeit in Sek.**	**Migrationen**	
14	473.60	459.08	7	3
15	657.54	486.99	26	26
16	782.87	731.79	97	7
17	642.77	579.71	117	10
Prozentuale Laufzeitgewinne liegen im Bereich von **3%** bis **26%**				

Tabelle 28: Laufzeitgewinne - Anwendungspulk II

Konfiguration	**Referenzlauf**	**Dynamische Lastbalancierung unter Verwendung von statischer Plazierungsinformation**		**Differenz in %**
18	291 Aufträge	351 Aufträge	49 Migrationen	21
19	(720,901) Sek.	(633,782) Sek.	113 Migrationen	12
20	126 Aufträge	129 Aufträge	121 Migrationen	2
Prozentuale Durchsatzsteigerungen liegen im Bereich von **2%** bis **21 %**				

Tabelle 29: Durchsatzsteigerungen

6.3.6 Auswirkungen der initialen Plazierung auf das Ergebnis der dynamischen Lastbalancierung

Um die Leistungsfähigkeit der Plazierungskomponente bzw. deren Auswirkung auf das Ergebnis der dynamischen Lastbalancierung zu ermitteln, werden die in Tabelle 23 beschriebenen Lastbalancierungsläufe mit deaktivierter Plazierungskomponente wiederholt. Deaktivierte Plazierungskomponente bedeutet, daß die Anwendungsprozesse unter Berücksichtigung der Lastsituation zufällig auf die Blattkomponenten verteilt werden. Bei einer deaktivierten Plazierungskomponente verwendet der Lastbalancierer jedoch immer noch eine maximale Anzahl von Anwendungsprozessen pro Knoten um sicherzustellen, daß zumindest jeder Knoten einen Anwendungsprozeß zugewiesen bekommt. Die Ergebnisse der Vergleichsmessung sind in Tabelle 30 wiedergegeben.

Konfiguration	Dynamische Lastbalancierung mit statischer Plazierungsinformation		Dynamische Lastbalancierung ohne statische Plazierungsinformation		Diff. in %
		Migrationen		Migrationen	
Laufzeitoptimierung einer einzelnen Anwendung					
1	450.27 Sek.	11	496.51 Sek.	16	10
2	241.66 Sek.	11	280.95Sek.	20	14
3	171.37 Sek.	24	193.47 Sek.	30	13
4	321.70 Sek.	2	364.45 Sek.	4	13
5	225.77 Sek.	5	229.82 Sek.	5	2
6	156.59 Sek.	18	155.02 Sek.	16	-1
Anwendungspulk I					
7	314.93 Sek.	44	306.45 Sek.	42	-3
8	246.22 Sek.	15	247.22 Sek.	19	0
9	330.55 Sek.	34	328.98 Sek.	37	0
10	336.84 Sek.	30	346.69 Sek.	38	3
11	344.75 Sek.	61	341.53 Sek.	60	-1
12	349.30 Sek.	88	348.08 Sek.	85	0
13	336.09 Sek.	114	334.80 Sek.	139	0
Anwendungspulk II					
14	459.08 Sek.	7	470.46 Sek.	5	2
15	486.99 Sek.	26	562.91 Sek.	32	16
16	731.79 Sek.	97	735.56 Sek.	101	1
17	579.71 Sek.	117	581.38 Sek.	115	0
Durchsatz					
18	351 Auftr.	49	360 Auftr.	47	-2
19	(633,782) Sek.	113	(646,815) Sek.	102	2
20	126 Auftr.	121	122 Auftr.	107	3

Tabelle 30: Auswirkungen der initialen Anwendungsplazierung.

Während sich die Qualität der initialen Plazierung deutlich auf die Laufzeiten einer einzelnen Anwendung (Konfiguration 1 bis 6) auswirkt, sind die Auswirkungen auf die Konfigurationsgruppe Anwendungspulk I uneinheitlich. Bei den Konfigurationen des Anwendungspulks II ergibt sich ein vergleichbares Bild. In den Konfigurationen 14 und 15, in denen jeweils nur eine parallele Anwendung aktiv ist, können zum Teil deutliche Laufzeitverbesserungen erzielt werden, die mit denen der Konfigurationen 1 bis 6 vergleichbar sind. In den Konfigurationen 16 und 17 entsprechen die Ergebnisse denen der Konfigurationen 9 und 11.
Für die Konfigurationsgruppe 3 hat sich die Verwendung der statischen Plazierungskomponente nur im Fall der Strömungs-Anwendung nicht positiv ausgewirkt, wobei die Durchsatzverschlechterung von rund 2% noch im Bereich der Meßungenauigkeit liegt. Für die Konfigurationen 19 und 20

hingegen konnten durch die Verwendung der Plazierungskomponenten leichte Durchsatzsteigerungen erzielt werden.
Besonders deutlich wird die Auswirkung einer guten initialen Plazierung in der Konfiguration 2. Im Vergleich zu den Läufen mit statischer Plazierungsinformtion benötigt der Lauf ohne Plazierungsinformation 9 Migrationen mehr. Insbesondere benötigt diese Konfiguration in der ersten Hälfte der Berechnung mehr Migrationen, um einen stabilen Lastverteilungszustand zu erreichen.
Insgesamt lohnt sich die Verwendung der initialen Plazierungskomponente des PaLaBer-Systems, da die Laufzeitgewinne deutliche größer sind als die Laufzeitverluste.

6.3.7 Dynamische Lastbalancierung unter Berücksichtigung der Kommunikationsbeziehungen

Zur Ermittlung, inwieweit sich die Berücksichtigung der Kommunikationsbeziehungen in der dynamischen Lastbalancierungsstrategie auf die Laufzeiten auswirkt, werden sämtliche 20 Konfigurationen mit und ohne Berücksichtigung der Kommunikationsbeziehungen in den Blattkomponenten und den inneren Komponenten getestet.
Die dynamische Lastbalancierungshierarchie PaLaBer verwendet die Information bzgl. der Kommunikationsbeziehungen sowohl in den inneren Komponenten, bei der Auswahl von Quell- und Zielknoten, als auch in den Blattkomponenten zur Auswahl geeigneter Migrationskandidaten (vgl. Abschnitt 5.6 und 5.7).
Das Ergebnis der Messungen ist vergleichbar mit den Ergebnissen im vorigen Abschnitt. Für die Laufzeitoptimierung einer einzelnen Anwendung erweist sich die Verwendung in allen 6 Konfigurationen als lohnenswert. Die Laufzeitunterschiede betragen zwischen 1% und 7%.
Für den Anwendungspulk I und II ist das Ergebnis wieder uneinheitlich. Die Laufzeitunterschiede betragen hier zwischen -1% und 9%.
Für die Durchsatzkonfigurationen 18 bis 20 ist das Ergebnis dieser Meßreihe wieder uneinheitlich. Während für die Konfigurationen 18 und 20 Durchsatzsteigerungen von bis zu 5% erzielt wurden, wurde bei Konfiguration 19 eine Verschlechterung von 3% erzielt.
Da in insgesamt 15 der 20 Konfigurationen Laufzeitgewinne von bis zu 9% erzielt wurden, in 3 Konfigurationen die Laufzeitverschlechterung ≤ 0.5% waren und nur in einer Konfiguration eine Laufzeitverschlechterung von 3% erzielt wurde, wird in allen weiteren Messungen die Kommunikationsbeziehung zwischen den Anwendungsprozessen berücksichtigt. Dieses Ergebnis stimmt auch mit den Ergebnissen verschiedener anderer Untersuchungen [FeYN88][GrNR90][Heis94][Milo94] überein, die besagen, daß sich die Berücksichtigung der Kommunikationsbeziehungen in vielen Fällen positiv auf die Laufzeitergebnisse auswirkt.

6.3.8 Ermittlung der systembedingten Zusatzlast

Zur Ermittlung der systembedingten Zusatzlast werden die in Abbildung 32 a) auf Seite 95 dargestellten Läufe mit der Originalversion der Strömungs-Anwendung wiederholt. Diese Anwendungsversion basiert direkt auf der NX-Kommunikationsbibliothek von Intel und verwendet keinerlei Funktionalität des PaLaBer-Systems. Für diese Untersuchung wird die Strömungs-Anwendung verwendet, da sie im Vergleich zum parallelen Raytracer intensiven Gebrauch vom Laufzeitsystem der Lastbalancierungsumgebung macht.
In allen Messungen wird jeweils ein Anwendungsprozeß auf einen Rechenknoten des Zielsystems gelegt. In allen vier Konfigurationen liegt der Systemoverhead des PaLaBer-Systems bei nur maximal 3.5%. D.h. die Gesamtlaufzeit ist beim Lauf unter dem PaLaBer-System um maximal 3.5% länger als bei der Originalversion, wobei keinerlei Gebrauch von der Lastbalancierungsfunktionalität

gemacht wird. Erfaßt werden bei diesen Vergleichsmessungen sowohl Zusatzlasten, die unabhängig von der Größe der Lastbalancierungshierarchie sind wie z.B.

- Start der Anwendung

als auch Zusatzlasten, die größenabhängig sind, wie z.B.

- die Lasterfassung und -aufbereitung,
- die Erfassung und Abspeicherung von Lastdaten zur späteren graphischen Darstellung und
- der Kommunikationsmanager.

Da bei den in Abbildung 32 dargestellten Konfigurationen keine dynamische Lastbalancierung stattfindet, verursacht das PaLaBer-System bei diesen Messungen nur Zusatzlast und trägt somit nicht zur Verbesserung der Laufzeit bei. D.h. in den ermittelten 3.5% Systemoverheads sind sämtliche relevanten Zusatzlasten erfaßt.
Wie bereits im vorigen Kapitel angesprochen, könnte der Systemoverhead noch weiter reduziert werden, wenn die Lastbalancierungsumgebung in das Betriebssystem integriert würde. Dies würde sich vor allem auf die Leistungsfähigkeit des Kommunikationsmanagers positiv auswirken, da die Anzahl der Verwaltungsnachrichten dadurch verringert werden könnte.

6.3.9 Auswirkungen der dynamischen Lastbalancierung auf das Lastprofil einer parallelen Anwendung

Im letzten Abschnitt dieses Kapitels wird die Auswirkung der dynamischen Lastbalancierung auf das Lastprofil einer parallelen Anwendung beschrieben. D.h. es wird versucht, die Frage zu beantworten, wie das dynamische Lastbalancierungssystem PaLaBer die Laufzeitgewinne erzielt. Dazu wurden die Lasterfassung und Lastaufbereitung in den Blattkomponenten so modifiziert, daß jede Blattkomponente die Anzahl der Meßintervalle mit vergleichbarer Prozessorauslastung (5% Intervall) pro Anwendungslauf erfaßt und abspeichert. Die ermittelten Werte dieser Messungen sind für die Konfigurationen 1 und 4 in Abbildung 40 und Abbildung 41 dargestellt.
In Abbildung 40 a) ist das Laufzeitprofil der Raytracing-Anwendung auf sechs Knoten mit jeweils einem Anwendungsprozeß pro Knoten dargestellt (vgl. Abbildung 34 auf Seite 97). Abbildung 40 b) zeigt das Laufzeitprofil der Konfiguration 4 aus Tabelle 20. Es ist deutlich erkennbar, daß mit Hilfe der dynamischen Lastbalancierung des PaLaBer-Systems die Anzahl der Meßintervalle mit einer Prozessorauslastung zwischen 0% und 5% signifikant geringer ist (142 Meßintervalle). Teilt man die 142 Meßintervalle durch die Anzahl der verwendeten Knoten und multipliziert man die Anzahl der Meßintervalle mit der Länge des Meßintervalls (2,7 Sekunden), ergibt sich eine durchschnittliche Laufzeitersparnis von 63,7 Sekunden. Da der tatsächlich erzielte Laufzeitgewinn mittels der dynamischen Lastbalancierung ca. 67 Sekunden beträgt, kann festgestellt werden, daß sich die Verringerung der Meßintervalle mit geringer Prozessorauslastung unmittelbar positiv auf das Laufzeitergebnis auswirkt und nicht durch die Systemlast der PaLaBer-Umgebung verursacht wird.
Daß in beiden Profilen in Abbildung 40 die Anzahl der Intervalle mit einer Prozessorauslastung von 100% gleich ist, resultiert aus den Eigenschaften der Raytracing-Anwendung. Bei der initialen Zuteilung der Bearbeitungsaufträge an die Server durch den Client-Prozeß sind alle Server in der Lage, ihre Prozessorressource zu 100% auszulasten. Aus diesem Grund ändert sich die Anzahl der Intervalle mit einer Auslastung von 100% nicht wesentlich.
In Abbildung 41 ist das Laufzeitprofil der Konfiguration 1 aus Tabelle 20 dargestellt. Auch dieser Vergleich der Laufzeitprofile zeigt deutlich, daß mit Hilfe der dynamischen Lastbalancierung die Anzahl der Meßintervalle mit einer Prozessorauslastung kleiner 50% signifikant, d.h. von 41% auf

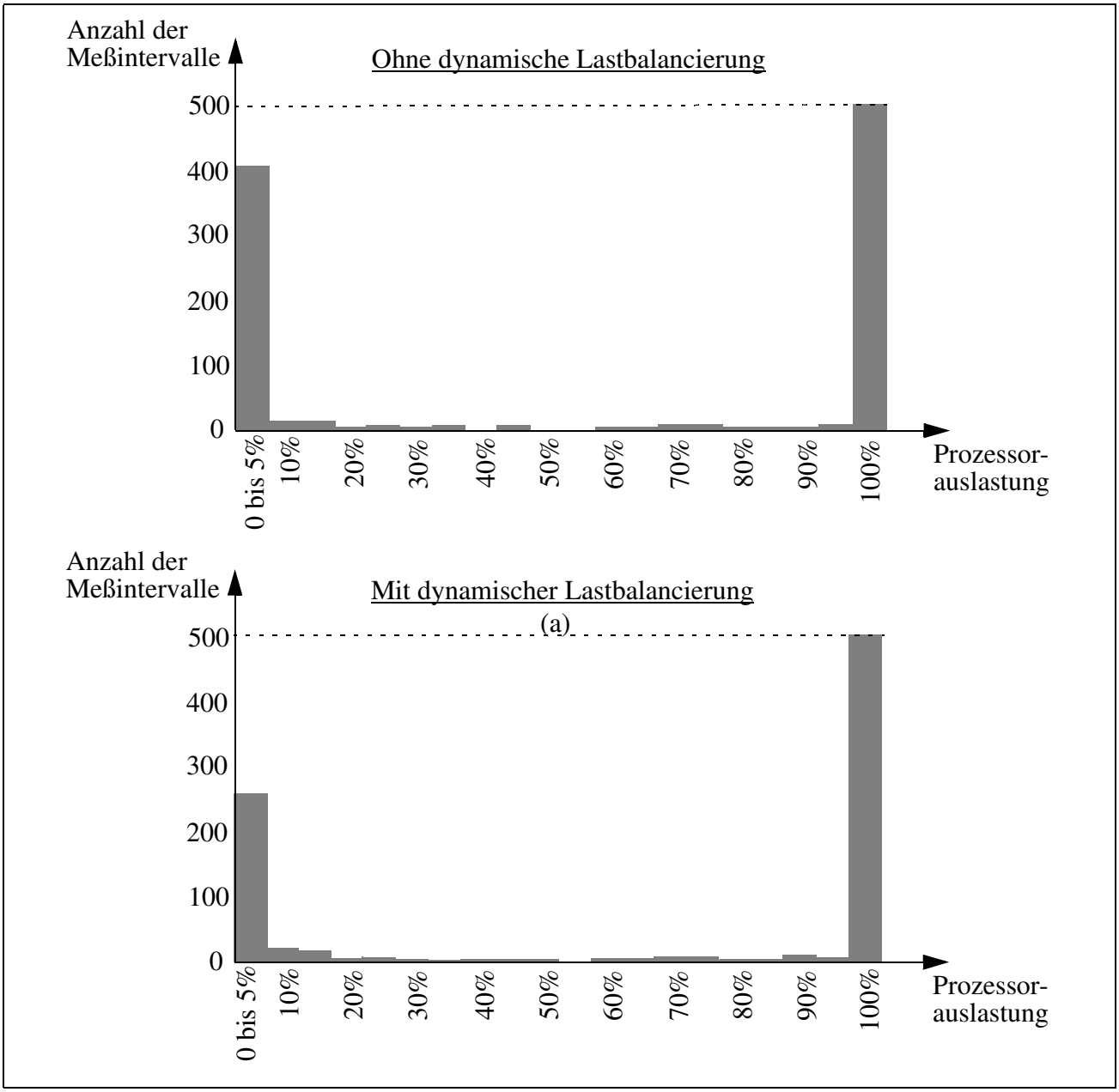


Abbildung 40: Lastprofil der Raytracing-Anwendung

17%, verringert werden kann. In diesem Fall kann jedoch nicht nur die Anzahl der Meßintervalle mit geringer Prozessorauslastung verringert werden, sondern auch die Anzahl der Meßintervalle mit hoher Prozessorauslastung deutlich erhöht werden, d.h. von 59% auf 83%. Insgesamt erzielt das PaLaBer-System seine Laufzeitgewinne durch eine bessere Auslastung der zur Verfügung stehenden Ressourcen.

Durch die gute Auslastung der Systemressourcen, die bereits mit Hilfe eines Lastbalancierer der Ebene 0 erzielt werden kann, ist die Möglichkeit von Leistungsgewinnen durch die Verwendung von höheren Informationsebenen natürlich eingeschränkt. So stehen einem Lastbalancierer der höheren Informationsebenen in der Konfiguration 1 theoretisch nur noch ungefähr 17% der Meßintervalle zur Optimierung der Laufzeitergebnisse zur Verfügung.

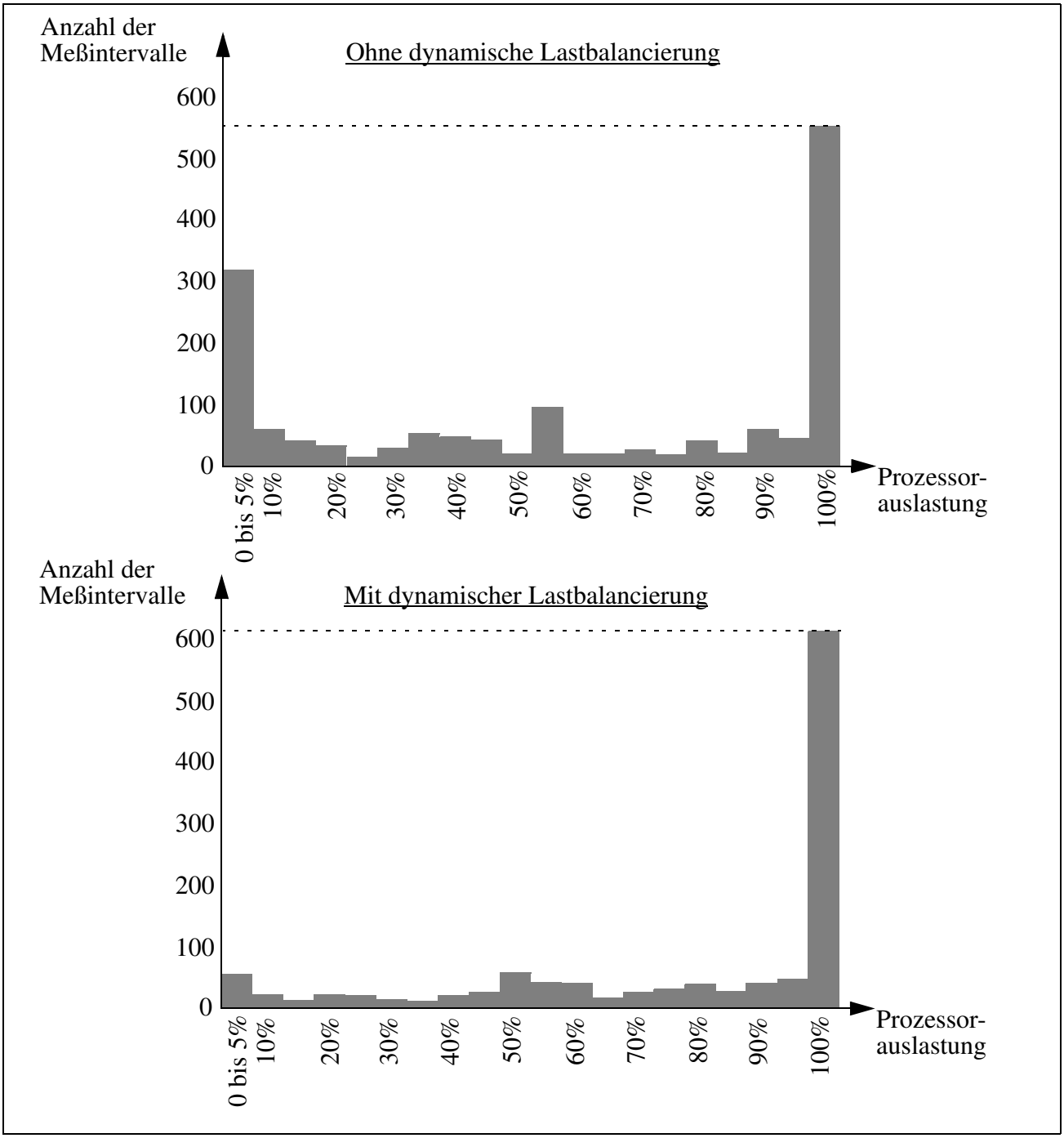


Abbildung 41: Lastprofil der Strömungs-Anwendung.

Kapitel 7

Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung

In diesem Kapitel werden die Ergebnisse der Untersuchung präsentiert, wie sich die verschiedenen Informationsebenen auf die Effizienz der dynamischen Lastbalancierung auswirken. Dazu werden alle 2^7=128 möglichen Kombinationen[1], die mit Hilfe der Informationsebenen erzielbar sind, mit 19 verschiedenen Meßkonfigurationen (vgl. Tabelle 20) kombiniert. Auf die Meßkonfiguration 13 mit 66 Blattkomponenten wird bei dieser Auswertung verzichtet, da die Kooperation verschiedener Lastbalancierungsteilbäume bereits mit der Konfiguration 12 getestet werden kann[2]. Um für die Auswertung eine gesicherte Zahlenbasis zu bekommen, wird jeder der 128 * 19 = 2048 Konfigurationen mehrfach ausgeführt, so daß insgesamt ca. 13.000 Einzelmessungen mit Laufzeiten von bis zu 13 Minuten durchgeführt werden. Dadurch konnte sichergestellt werden, daß die Meßergibnisse innerhalb der Konfigurationen Schwankungen von maximal 3% aufweisen. Alle im nachfolgenden beschriebenen Meßergebnisse werden dabei immer mit einem Referenzlauf der Ebene 0 ohne statische Plazierungsinformation (siehe dazu Tabelle 30 auf Seite 109) verglichen.

7.1 Plausibilitätsprüfung der Ebenenumsetzung

Wie bereits im vorigen Kapitel begründet, läßt sich die korrekte Arbeitsweise des hierarchischen Lastbalancierungssystems PaLaBer und insbesondere die technische Umsetzung der Informationsebenen im System nicht in einem streng mathematischen Sinne nachweisen. Aus diesem Grund beginnt dieses Kapitel mit einer Plausibilitätsprüfung sowohl der technischen Umsetzung der Informationsebenen als auch deren Verwendung durch den Lastbalancierer. Dazu wird ausgehend von den umfangreichen Meßergebnissen versucht, anhand von Analysen und Einzellaufbeschreibungen die korrekte Arbeitsweise des Systems aufzuzeigen. Auf eine Plausibilitätsprüfung der Umsetzung der Ebene 0 wird hier verzichtet, da diese Überprüfung bereits Bestandteil des vorigen Kapitels war.

---

1. Die Informationsebenen 1, 2 und 3 werden im Rahmen dieser Untersuchung exemplarisch mit 7 verschiedenen Informationen (siehe Tabelle 35) dargestellt. Dadurch ergeben sich die 128 verschiedenen Informationskombinationen.
2. Außerdem war die Verwendung dieser Konfiguration im Hinblick auf andere Benutzer des Zielsystems Intel Paragon nicht vertretbar.

7.1.1 Informationsebene 1

In diesem Unterabschnitt werden die Auswirkungen der Informationsebene 1 auf die Effizienz der dynamischen Lastbalancierung untersucht. Zur Erinnerung sei darauf hingewiesen, daß für die Informationsebene 1 die folgenden drei Informationen verwendet werden:

1. Statische Plazierungsinformation
2. Statische Angaben über die Kommunikationspartner eines Anwendungsprozesses
3. Statistische Angaben des Prozessor-/Hauptspeicherbedarfes eines Anwendungsprozesses

Im nachfolgenden wird auf jede dieser drei Informationen getrennt eingegangen, und es werden ihre Auswirkungen auf das Laufzeitverhalten der im vorigen Abschnitt genannten 19 Konfigurationen beschrieben.

Statische Plazierungsinformation

Statische Plazierungsinformation versetzt einen dynamischen Lastbalancierer in die Lage, eine initiale Aufteilung der Anwendungsprozesse auf die verfügbaren Systemressourcen zu erzielen, die von Anfang an eine möglichst gleichmäßige Auslastung der Systemressourcen bewirkt.
Eine gute initiale Plazierung lohnt sich dann, wenn sich die Plazierung aufgrund des Kommunikationsverhaltens und des Prozessorbedarfs der Anwendung auf die Laufzeit auswirkt und die parallele Anwendung keine oder nur unzureichende Möglichkeiten hat, auf eine schlechte initiale Plazierung zu reagieren. Dementsprechend ist zu erwarten, daß sich die statische Plazierungsinformation besonders bei der enggekoppelten, parallelen Lattice-Boltzman-Hydrodynamik-Anwendung positiv auswirken wird. Wie der Tabelle 30 auf Seite 109 zu entnehmen ist, können in den Konfigurationen 1-3 und 15 sehr deutliche Laufzeitgewinne von bis zu 16% erzielt werden. In diesen vier Konfigurationen wird nur die Lattice-Boltzman-Hydrodynamik-Anwendung verwendet. Da diese Anwendung keinen anwendungsintegrierten Lastausgleichsmechanismus verwendet, hat sie keine Möglichkeit, auf eine ungünstige initiale Plazierung zu reagieren, weshalb sich die statische Plazierungsinformation besonders bei dieser Anwendung auswirkt.
Im Unterschied dazu verfügt der parallele Raytracer über einen anwendungsintegrierten Lastausgleichsmechanismus und kann entsprechend auf eine schlechte initiale Plazierung reagieren. Allerdings arbeitet dieser Ausgleichsmechanismus, wie bereits in Kapitel 6 beschrieben, bei einem groben Prozeßgranulat nur unzureichend. Folgerichtig kann mit Hilfe der statischen Plazierungsinformation in der Konfiguration 4 ein deutlicher und in Konfiguration 5 noch ein geringer Laufzeitgewinn erzielt werden, während bei der Konfiguration 6 kein Laufzeitgewinn erzielbar ist.

Statische Kommunikationspartner

Wie bereits in Kapitel 4 beschrieben, ist die Kommunikation zwischen zwei Prozessen innerhalb eines Knotens des Zielsystems wesentlich teurer als die Kommunikation über Knotengrenzen hinweg. Der dynamische Lastbalancierer berücksichtigt diese Tatsache dadurch, daß Prozesse, die miteinander kommunizieren, nicht auf den selben Knoten plaziert bzw. migriert werden.
Da es sich beim parallelen Raytracer um eine Anwendung mit nur sehr geringer Kommunikation handelt, ist zu erwarten, daß sich die Information über Kommunikationspartner nicht auf die Laufzeit auswirkt. Dies wird durch die Messungen bestätigt; in keiner der Konfigurationen kann mit Hilfe dieser Information ein Laufzeitgewinn erzielt werden.
Im Vergleich dazu können in den Konfigurationen 1, 2 und 15 Laufzeitgewinne zwischen 2% und 7% erzielt werden. Im ersten Moment scheint es überraschend, daß die Verwendung dieser Information nicht zu Laufzeitgewinnen bei der Konfiguration 3 führt, obwohl diese Konfiguration sehr

kommunikationsintensiv ist. Da diese Konfiguration jedoch aus 32 Prozessen besteht und jeder Prozeß im wesentlichen nur mit seinen beiden Nachbarprozessen kommuniziert, ist die Wahrscheinlichkeit, Kommunikationspartner durch die Prozeßmigration auf denselben Knoten zu legen, wesentlich geringer als bei den Konfigurationen 1, 2 oder 15. So beträgt die Wahrscheinlichkeit bei der Konfiguration 1 einen Kommunikationspartner auf dem Zielknoten anzutreffen 1/3, bei den Konfigurationen 2 und 15 jeweils 2/6 und bei der Konfiguration 3 1/12.

Statistische Angaben des Prozessor- und Hauptspeicherbedarfes eines Anwendungsprozesses

Die statistischen Angaben über den Ressourcenbedarf eines Anwendungsprozesses kann der dynamische Lastbalancierer in dreifacher Hinsicht verwenden:

1. Beim Start eines Anwendungsprozesses zur Minimierung der Seitenwechselrate.
2. Zur Vermeidung unproduktiver Migrationen eines Anwendungsprozesses am Ende seiner Laufzeit.
3. Zum rechtzeitigen Start weiterer Anwendungen.

Während die Punkte 1) und 2), bei Anwendungen mit einer längeren Laufzeit nur zu geringen Laufzeitverbesserungen führen können (bei den Konfigurationen 1 bis 12 liegt der maximale Laufzeitgewinn bei 2%), kann der Punkt 3) zu erheblichen Laufzeitgewinnen führen.
Da ein Lastbalancierer der Ebene 0 nicht über die Information verfügt, wie lange eine Anwendung aktiv ist, kann er nicht zwischen einem lokalen bzw. globalen Synchronisationspunkt und dem Ende der Anwendungslaufzeit unterscheiden. Ausgehend von der statistischen Ressourcenabschätzung kann ein Lastbalancierer nun, unter der Annahme, daß die Information hinreichend genau ist, eine Anwendung vorzeitig starten und dadurch die Systemressourcen deutlich besser auslasten.
Dementsprechend werden in den Konfigurationen 14, 16 und 17 Laufzeitgewinne von bis zu 8% erzielt. Lediglich bei der Konfiguration 15 kann kein Laufzeitgewinn erzielt werden, da bei dieser Konfiguration alle Prozesse bis zum Ende der gesamten Berechnung aktiv sind und es somit nicht möglich bzw. sinnvoll ist, eine andere Anwendung vorzeitig zu starten. Im Abschnitt 7.4 wird auf das vorzeitige Starten einer Anwendung anhand der Konfiguration 14 noch detaillierter eingegangen.

Insgesamt kann daraus die Schlußfolgerung gezogen werden, daß sich die Verwendung von Informationen der Ebene 1 in plausiblen Meßergebnissen widerspiegelt und somit auf die korrekte Umsetzung der Konzepte geschlossen werden kann.

7.1.2 Informationsebene 2

Diese Ebene umfaßt Informationen über den aktuellen Bearbeitungszustand einer parallelen Anwendung. Sie wird vom Lastbalancierer dazu verwendet zu entscheiden, ob sich ein Anwendungsprozeß gegenwärtig in einer Bearbeitungsphase befindet, in der er ein geeigneter oder ein ungeeigneter Migrationskandidat ist.
Da lediglich die Strömungs-Anwendung über entsprechende Bearbeitungsphasen verfügt, wird für die Plausibilitätsprüfung der Ebene 2 hauptsächlich diese Anwendung verwendet. Angesichts der Tatsache, daß in den Konfigurationen 1, 2 und 3 die Problemgröße konstant gehalten wird und lediglich die Anzahl der Anwendungsprozesse variiert, unterscheiden sich diese Konfigurationen erheblich in ihrer Phasenausprägung. In Abschnitt 6.3.2 wurden die Phasen der Strömungs-Anwendung mit $CPU_{grobesGitter}$, $CPU_{zusatzFeinesGitter}$, $CPU_{Leerlauf}$ und Komm bezeichnet. Konfiguration 1 verfügt über sehr ausgeprägte Phasen, während bei der Konfiguration 2 und 3 die Phasenausprägung erheblich nachläßt. Folgerichtig nehmen die Laufzeitgewinne von 5.3% bei Konfiguration 1 bis auf

2.5% bei Konfiguration 3 ab.
Somit spiegelt sich die Verwendung der Informationen der Ebene 2 in plausiblen Meßergebnissen wider, weshalb auch bei der Ebene 2 auf eine korrekte Umsetzung geschlossen werden kann.

7.1.3 Informationsebene 3

Bei den Informationen der Ebene 3 handelt es sich um Prognosen über das Laufzeitverhalten, die von der Anwendung an den dynamischen Lastbalancierer weitergereicht werden. Für die Umsetzung dieser Ebene verwendet das PaLaBer-System drei Informationen:

1. Vorzeitige Ankündigung der Terminierung
2. Abschätzung der Restlaufzeit
3. Durchschnittlicher Prozessorbedarf innerhalb der nächsten Bearbeitungsphase

Vorzeitige Ankündigung der Terminierung

Die vorzeitige Ankündigung ist die einfachste Anwendungsprognose, die für die Informationsebene 3 verwendet wird. Sie ist jedoch nicht nur die einfachste Information, sondern zugleich auch die Information mit der höchsten Zuverlässigkeit, da diese Information unmittelbar vor der Beendigung der Berechnungen an den Lastbalancierer weitergeleitet wird. Ein dynamischer Lastbalancierer kann diese Information, vergleichbar mit der statistischen Ressourcenbedarfsschätzung der Ebene 1, in doppelter Hinsicht verwenden:

1. Zur Vermeidung einer unproduktiven Migration eines Anwendungsprozesses am Ende seiner Laufzeit
2. Zum vorzeitigen Starten einer anderen Anwendung

Da diese Information wesentlich zuverlässiger ist als die statistische Abschätzung der Gesamtlaufzeit eines Anwendungsprozesses, sind die erzielbaren Laufzeitersparnisse deutlich höher als bei der statistischen Ressourcenbedarfsschätzung der Ebene 1. So kann bei der Konfiguration 15 die Laufzeit um über 12% durch das vorzeitige Starten der zweiten Anwendung verringert werden.
Die Auswertung einer Vielzahl von Einzelläufen hat auch gezeigt, daß mit Hilfe dieser Information unproduktive Migrationen am Ende der Berechnung vermieden werden können. Allerdings wirkt sich dieser Sachverhalt, speziell bei lang laufenden Anwendungen, nicht signifikant auf die Gesamtlaufzeit aus. In Abschnitt 7.4 wird ein ausführliches Beispiel für die Verwendung dieser Information besprochen.

Abschätzung der Restlaufzeit

Die Information über die Abschätzung der Restlaufzeit eines Anwendungsprozesses kann ein dynamischer Lastbalancierer in gleicher Weise einsetzen wie die Information bzgl. der vorzeitigen Ankündigung der Terminierung bzw. der statistischen Ressourcenabschätzung.

1. Zur Vermeidung einer unproduktiven Migration eines Anwendungsprozesses am Ende seiner Laufzeit
2. Zum vorzeitigen Starten einer anderen Anwendung

Da diese Information jedoch größere Unsicherheiten aufweist als die vorzeitige Ankündigung der Terminierung sind die erzielbaren Laufzeitgewinne geringer. Bei der Konfiguration 15 liegen z.B. die Laufzeitgewinne bei 5%.
Anhand des Laufzeitverhaltens der Konfiguration 15 läßt sich sehr anschaulich beschreiben, warum

diese Information mit einer erheblichen Unsicherheit belastet ist. Wie im vorigen Kapitel beschrieben, verteilt der Client-Prozess der Raytracing-Anwendung zu Beginn der Berechnungen sämtliche Bildzeilen auf die verfügbaren Server-Prozesse. Sobald ein Server-Prozess seine Bildzeilen berechnet hat, fordert er vom Client neue Bildzeilen an. Der Client versucht daraufhin, von einem anderen Server die Hälfte der noch nicht berechneten Bildzeilen abzuziehen. Da kein Server weiß, ob er einen weiteren Berechnungsauftrag vom Client zugewiesen bekommt, bzw. kein Server weiß, ob er die Hälfte seiner noch nicht berechneten Bildzeilen an einen unterbelasteten Server abgeben muß, kann ein Server seine Restlaufzeit nur sehr ungenau bestimmen.

Durchschnittlicher Prozessorbedarf innerhalb der nächsten Bearbeitungsphase

Mit Hilfe dieser Information kann ein Anwendungsprozeß dem Lastbalancierer einen bevorstehenden Lastwechsel mitteilen. Der dynamische Lastbalancierer kann diese Information dazu nutzen, um den voraussichtlichen zukünftigen Lastzustand eines Knotens zu bestimmen und gegebenenfalls präventive Maßnahmen in Form von Lastverschiebungen zu ergreifen. Damit sich die Verwendung dieser Information signifikant auf das Laufzeitergebnis auswirken kann, muß das Laufzeitverhalten einer parallelen Anwendung nicht nur gewissen Schwankungen unterworfen sein, sondern die Anwendung muß diese Veränderung auch im voraus abschätzen können.
Während das Laufzeitverhalten des parallelen Raytracers sehr starken Schwankungen unterworfen ist, die die Anwendung aufgrund des integrierten Lastausgleichsmechanismus jedoch nicht im voraus kennt (siehe obige Erläuterung), kann die Lattice-Boltzman-Hydrodynamik-Anwendung ihre Laufzeitschwankungen, d.h. die Umschaltung auf ein anderes Berechnungsgitter, sehr genau vorhersagen.
Dieser Sachverhalt läßt sich sehr gut an den durchgeführten Messungen der Konfigurationen 1, 2 und 3 erkennen. Insbesondere spiegelt sich in den Messungen die Tatsache wider, daß mit abnehmendem Prozeßgranulat die absoluten Laufzeitunterschiede zwischen den beiden Berechnungsgittern der Lattice-Boltzman-Hydrodynamik-Anwendung abnehmen. So weist die Konfiguration 1 einen Laufzeitgewinn von 6.89% auf, Konfiguration 2 von 3.74% und Konfiguration 3 schließlich einen Laufzeitgewinn von nur noch 1.86%.

Insgesamt läßt sich auch für die Informationsebene 3 feststellen, daß sich auch die technische Umsetzung der Ebene 3 in plausiblen Meßergebnissen widerspiegelt.
In den geschilderten Plausibilitätsüberlegungen wurde in erster Linie mit den Konfigurationen 1 bis 6 argumentiert. Dies ist insofern nicht überraschend, als in den Auswertungen jeweils die Auswirkung einer einzelnen Information auf das Ergebnis der dynamischen Lastbalancierung analysiert wurde. Da sich das komplexe Lastverhalten der restlichen Konfigurationen, die aus Kombinationen der beiden zur Verfügung stehenden parallelen Anwendungen bestehen, jedoch nur unzureichend mit nur einer weiteren Information beschreiben läßt, waren die Auswirkungen auf die restlichen 12 Konfigurationen nur marginal.

7.2 Untersuchung des Einflusses der Informationsebenen

In diesem Abschnitt findet eine Auswertung der Meßergebnisse in bezug auf die Informationsebenen bzw. deren Kombination statt. Dabei wird nicht weiter differenziert, aufgrund welcher konkreter Information der jeweiligen Ebene ein Laufzeitergebnis erzielt wird. Eine Auswertung der Ergebnisse bezüglich der konkreten Informationskombinationen ist Bestandteil des nächsten Abschnittes.
Die Auswertung der Meßergebnisse in bezug auf die Informationsebenen erfolgt mit Hilfe der Tabellen 31 und 32. Zur Erinnerung sei gesagt, daß in den Konfigurationen 1 bis 17 und 19 die ermittelten Gesamtlaufzeiten dargestellt wird und somit ein prozentualer Wert kleiner als 100 als Leistungssteigerung zu werten ist, während in den Konfigurationen 18 und 20 die Anzahl der bearbeiteten Aufträge dargestellt. Aus diesem Grund bedeutet bei diesen beiden Konfigurationen ein prozentuales Ergebnis größer als 100 einen Leistungsgewinn.
In Tabelle 31 ist für jede der 19 Konfigurationen und für jede der 7 möglichen Ebenenkombinationen das jeweils beste Laufzeitergebnis eingetragen. Kann für eine Konfiguration kein Laufzeitgewinn ermittelt werden, wird dies durch den Eintrag „-“ gekennzeichnet.
Da die Verwendung der höheren Informationsebenen nicht nur zu Laufzeitgewinnen führt, sondern unter Umständen auch zu einer Verlängerung der Laufzeit im Vergleich zum oben angeführten Referenzlauf der Ebene 0 ohne statische Plazierungsinformation führen kann, werden in Tabelle 32 die Mittelwerte über alle Messungen der jeweiligen Informationskombination eingetragen.
Laufzeitverschlechterungen können bei Verwendung der höheren Informationsebenen im wesentlichen aus zwei Gründen auftreten:

1. Ist bereits der Lastbalancierer der Ebene 0 in der Lage, eine optimale Lastverteilung mit einer minimalen Anzahl von Lastverschiebungen zu erreichen, verursacht die Erfassung und Verwendung der Informationen der höheren Ebenen nur unproduktive Zusatzlast.
2. Die Informationen der höheren Ebenen, speziell der Ebenen 1 und 3, unterliegen einer gewissen Unsicherheit und Ungenauigkeit. Dies kann unter Umständen zu Fehlentscheidungen führen, die sich ihrerseits negativ auf das Laufzeitergebnis auswirken können.

Stellt man die Meßergebnisse in der Tabelle 31 und Tabelle 32 spaltenweise gegenüber, stellt man fest, daß die Ergebnisse der beiden Tabellen tendenziell übereinstimmen. So ist beiden Tabellen zu entnehmen, daß mit Hilfe der Informationsebene 1 im Mittel teilweise deutliche Laufzeitgewinne zu erzielen sind. Dies liegt in erster Linie daran, daß es sich bei den Informationen der Ebene 1 um Informationen handelt, die ein Lastbalancierer der Ebene 0 nicht besitzt und auch nicht bzw. nur sehr begrenzt abschätzen kann. So verfügt ein Lastbalancierer der Ebene 0 z.B. über keine Information bzgl. der Gesamtlaufzeit einer parallelen Anwendung und kann diese Information auch nicht durch die Beobachtung der Anwendung abschätzen. Ebenso kennt ein Lastbalancierer unmittelbar vor dem Start einer parallelen Anwendung nicht den Hauptspeicherbedarf der Anwendungsprozesse usw. Für einen Lastbalancierer der Ebene 0 handelt es sich bei den Informationen der Ebene 1 nicht nur um neue Informationen, sondern vor allem auch um Informationen, die er sowohl beim Start der Anwendung als auch bei den dynamischen Lasttransferentscheidungen verwenden kann. In diesem Zusammenhang ist es wichtig, darauf hinzuweisen, daß diese Informationsebene nicht nur bei der Optimierung der Laufzeit einer einzelnen Anwendung zu positiven Laufzeitergebnissen führen kann, sondern auch bei sehr heterogenen Anwendungslasten (vgl. Konfigurationen 14 bis 17 und Konfiguration 19 und 20).
Dagegen führt die Verwendung der Informationsebene 2 nur dann zu signifikanten Laufzeitverbesserungen (siehe Konfigurationen 1, 2, 3 und 7), wenn die parallele Anwendung über entsprechend ausgeprägte Bearbeitungsphasen verfügt. Daß es bei der Konfiguration 18 zu keiner Durchsatzsteigerung kommt, liegt daran, daß bei dieser Konfiguration bereits der Lastbalancierer der

Ebene 0 eine sehr gute Lastverteilung erreicht. Außerdem handelt es sich bei der Information der Ebene 2 nicht notwendigerweise um neue Informationen für einen dynamischen Lastbalancierer der Ebene 0, denn die prinzipielle Arbeitsweise eines dynamischen Lastbalancierers besteht darin, aus den Daten der Vergangenheit zumindest für einen kurzen Zeithorizont in die Zukunft zu extrapolieren.
Im Zusammenhang mit der Informationsebene 2 bedeutet dies, daß ein Lastbalancierer der Ebene 0 aus dem Lastverhalten der Anwendung im letzten Meßintervall auf das Lastverhalten der Anwendung im aktuellen bzw. zukünftigen Meßintervalls schließt..
Beim Vergleich der Meßergebnisse der Ebene 3 in Tabelle 31 und Tabelle 32 fällt sofort auf, daß es hier große Abweichungen gibt. So kann mit Hilfe der Ebene 3 z.B. für die Konfiguration 14 eine minimale Laufzeit von 87.86% im Vergleich zum Referenzlauf erzielt werden. Andererseits wird bei den durchschnittlichen Laufzeiten der Ebene 3 für die Konfiguration 4 eine Laufzeitverschlechterung von über 9% erzielt. D.h. bei der Informationsebene 3 ist die Zuverlässigkeit und Sicherheit der Anwendungsprognosen ein sehr kritischer Faktor. Speziell beim parallelen Raytracer (Konfiguration 4) kann ein Server-Prozeß nur sehr ungenaue Angaben über sein Lastverhalten machen, da sich das Lastverhalten durch den integrierten Lastausgleichsmechanismus unvorhersehbar ändern kann. In Abschnitt 7.4.2 wird ausführlich aufgezeigt, wie sich die Zuverlässigkeit bzw. Unzuverlässigkeit der Informationsebene 3 auf die Ergenisse der dynamischen Lastbalancierung auswirken kann.
Wesentlich interessanter als die Auswertung einer einzelnen Ebene ist die Auswertung von Ebenenkombinationen und den dabei erzielbaren Synergie-Effekten. Im Zusammenhang damit ist besonders die Kombination der Informationsebenen 1 und 3 interessant. Diese Kombination erzielt in 17 der 19 verwendeten Konfigurationen bessere Ergebnisse als die Ebenen 1 und 3 getrennt. Ein typisches Beispiel dafür ist die Konfiguration 10 in Tabelle 31. Mit Hilfe der Informationsebene 1 wird bei dieser Konfiguration eine Laufzeit von 96.99% des Referenzlaufes erzielt, die Laufzeit der Ebene 3 beträgt für diese Konfiguration 96.98%. Verwendet man hingegen eine Ebenenkombination aus 1 und 3, kann die Laufzeit auf 95.37% reduziert werden. Und selbst bei Konfigurationen, bei denen eine oder sogar beide Informationsebenen nicht in der Lage sind, eine Laufzeitverbesserung zu erzielen (z.B. Informationsebene 3 bei den Konfigurationen 4 und 9), kann mit Hilfe der Ebenenkombination ein Laufzeitgewinn erreicht werden.
Daß es speziell bei der Kombination der Ebene 1 und 3 zu Synergie-Effekten kommt, ist nicht überraschend, da beide Ebenen Informationen bereitstellen, über die ein Lastbalancierer der Ebene 0 nicht verfügt und die er auch nicht bzw. nur sehr ungenau abschätzen kann. Außerdem ergänzen sich z.B. die statistische Laufzeitinformation der Ebene 1 und die vorzeitige Ankündigung der Terminierung der Ebene 3. Ausgehend von der statistischen Angabe über die Gesamtlaufzeit einer Anwendung besitzt ein Lastbalancierer bereits wertvolle Information, die er bei den Lasttransferentscheidungen verwenden kann. Da diese Information jedoch gewissen Unsicherheiten unterliegt, kann mit Hilfe der Ebene 3 (vorzeitige Ankündigung der Terminierung, Abschätzung der Restlaufzeit) die Information über die Gesamtlaufzeit einer Anwendung schrittweise verbessert werden.

Konfig.	Ebene 1	Ebene 2	Ebene 3	Ebene 1,2	Ebene 1,3	Ebene 2,3	Ebene 1,2,3
Laufzeitoptimierung einer einzelnen Anwendung							
1	90.67	94.74	93.0	95.97	89.4	91.64	90.11
2	86.02	97.56	93.23	90.65	84.42	96.27	86.65
3	85.48	97.58	98.04	90.46	84.25	96.28	83.33
4	87.96	-	-	87.63	87.22	98.67	87.42
5	98.68	-	98.63	98.16	96.24	97.93	96.37
6	-	-	97.69	96.92	96.59	-	97.5
Anwendungspulk I							
7	96.3	98.01	90.45	96.74	90.01	93.19	91.41
8	95.96	-	97.37	-	93.50	96.12	95.52
9	-	-	-	-	99.47	-	99.08
10	96.99	-	96.98	97.80	95.37	98.57	95.17
11	-	-	-	98.68	99.27	-	99.02
12	99.08	-	98.0	-	97.19	96.06	96.46
Anwendungspulk II							
14	89.74	-	87.86	91.23	85.95	89.18	85.68
15	86.51	-	97.15	92.0	85.77	-	88.48
16	96.12	99.47	95.28	94.24	97.04	95.54	95.16
17	95.16	97.91	94.49	93.03	92.46	93.67	92.86
Durchsatz							
18	-	-	100,28	-	100.56	-	105.27
19	(89.7,89.6)	(97.8,96.4)	(99.5,98,7)	(96.9,97.9)	(72.2,75.8)	(99.9,98.7)	(71.5,76.2)
20	109.84	115.57	109.02	123.77	103.28	110.66	104.92

Tabelle 31: Bestes prozentuales Ergebnis der Ebenen bzw. Ebenenkombinationen[a].

a. Zur Erinnerung sei gesagt, daß für die Konfiguration 19 sowohl der Startzeitpunkt der 4 Anwendung als auch Endzeitpunkt der 5 Anwendung angegeben wird.

Konfig.	Ebene 1	Ebene 2	Ebene 3	Ebene 1,2	Ebene 1,3	Ebene 2,3	Ebene 1,2,3
Laufzeitoptimierung einer einzelnen Anwendung							
1	98.68	94.73	98.14	100.93	98.49	95.89	97.44
2	94.56	97.58	95.26	97.42	93.16	100.54	95.77
3	94.31	97.58	99.51	94.79	92.51	99.26	92.11
4	96.58	108.77	109.86	98.49	97.37	106.17	96.96
5	101.99	103.53	101.65	100.45	101.45	101.55	101.14
6	104.68	101.05	102.08	105.89	104.22	102.79	104.07
Anwendungspulk I							
7	100.57	98.01	94.94	100.66	97.76	97.05	98.33
8	101.40	100.12	101.11	104.29	102.91	99.39	102.57
9	102.61	101.8	101.59	102.08	102.09	101.47	102.35
10	99.33	101.02	100.84	99.79	99.39	100.15	99.89
11	103.24	102.00	103.23	103.15	102.92	104.11	103.73
12	101.94	99.71	100.96	101.72	100.85	99.78	101.03
Anwendungspulk II							
14	96.82	100.02	93.88	96.40	94.06	95.78	94.37
15	95.53	102.43	102.29	97.07	100.45	108.02	101.08
16	99.12	99.47	98.99	98.91	100.30	98.73	99.51
17	98.57	97.91	99.51	96.46	98.53	96.22	98.35
Durchsatz							
18	83.05	98.89	99.45	97.23	96.67	97.5	92.70
19	(90.7,91.9)	(97.8,96.4)	(100.9,99.5)	(104.1,104.4)	(82.0,84.2)	(100.8,99.1)	(81.4,84.6)
20	105.74	115.56	106.56	114.75	101.64	106.56	104.1

Tabelle 32: Durchschnittliche prozentuale Laufzeitergebnise der Ebenen bzw. Ebenenkombinationen.

7.3 Konfigurationsbezogene Auswertungen

Nachdem im vorigen Abschnitt eine Auswertung der erzielten Ergebnisse im Hinblick auf die Informationsebenen erfolgte, wird in diesem Abschnitt eine Auswertung in bezug auf konkrete Informationskombinationen erfolgen. Dazu wird untersucht, welche Kombination der in Abschnitt 5.10 vorgestellten Informationen bei den 19 verwendeten Konfigurationen den maximalen Laufzeit- bzw. Durchsatzgewinn erzielt, um dann im nächsten Abschnitt für mehrere Konfigurationen eine detaillierte Einzellaufbeschreibung durchzuführen. Bevor auf die Informationskombinationen eingegangen wird, die bei der jeweiligen Konfiguration das beste Ergebnis erzielen, wird noch auf einige generelle Ergebnisse eingegangen.
Bei der Auswertung der 2048 Meßkonfigurationen zeigt es sich, daß keine Informationskombination in der Lage ist, in allen 19 Konfigurationen, d.h. bei einer sehr heterogenen Arbeitslast, das Laufzeitergebnis des Referenzlaufes zumindest einzustellen. Trotzdem muß darauf hingewiesen werden, daß es Informationskombinationen gibt (vgl. Konfiguration 11 in Tabelle 35), die bei bis zu 17 der 19 Konfigurationen in der Lage waren, das Laufzeitergebnis des Referenzlaufes der Ebene 0 ohne statische Plazierungsinformation einzustellen.
Andererseits ist es jedoch auch bei jeder der 19 Konfigurationen möglich, mit Hilfe der höheren Ebenen die Laufzeit- bzw. Durchsatzergebnisse des Referenzlaufes zum Teil signifikant (bis zu 27,2%) zu verbessern. Dieses Ergebnis reiht sich somit nahtlos in die Ergebnisse ein, die in anderen Bereichen der Ressourcenverwaltung erzielt wurden [GaSh90]. So werden z.B. für die Seitenverdrängungstrategien im Zusammenhang mit der virtuellen Speicherverwaltung in modernen Betriebssystemen sehr einfache Strategien (LRU-Strategie, *Clock*, etc.) verwendet, da sich herausgestellt hat, daß diese einfachen Strategien für ein breites Anwendungsspektrum akzeptable Ergebnisse liefern. Andererseits werden für sehr spezielle Anwendungsfälle (z.B. Memory Mapped Files) Strategien verwendet, die Informationen über die Anwendungssemantik (Informationsebene 1) für ihre Entscheidungen verwenden und dadurch in der Lage sind, für diesen speziellen Anwendungsfall bessere Seitenverdrängungsentscheidungen zu treffen.
In Tabelle 33 sind für alle 19 Konfigurationen die besten und schlechtesten Ergebnisse aufgeführt, die mit Hilfe der höheren Ebenen erzielbar sind. Die schlechtesten Ergebnisse werden in dieser Tabelle aufgeführt, um auf das Spektrum der möglichen Auswirkungen hinzuweisen, die unter Verwendung des Ebenenkonzeptes auftreten.
Ein sehr anschauliches Beispiel ist die Konfiguration 3 in Tabelle 33. Bei dieser Konfiguration ist sowohl ein Laufzeitgewinn von 17% realisierbar als auch eine Laufzeitverschlechterung von 8%. Insgesamt konnte, basierend auf den Informationen der Ebenen 1, 2 und 3, die Laufzeit bzw. der Durchsatz des Referenzlaufen zwischen 0.5% und bis zu 27% verbessert werden. Dabei werden die minimalen Gewinne ≤1% bei Konfigurationen (Konfiguration 9 und 11) erzielt, bei denen bereits der Referenzlauf in der Lage ist, das Ergebnis im Vergleich zum Lauf ohne dynamische Lastbalancierung um mindestens 30% zu verbesseren.
Viel wichtiger als die Optimierung der Laufzeiten bzw. des Durchsatzes von Konfigurationen, bei denen bereits ein Lastbalancierer der Ebene 0 deutliche Verbesserungen erzielen kann, sind jedoch Anwendungslasten, bei denen aufgrund des komplexen Lastverhaltens ein Lastbalancierer der Ebene 0 keine leistungssteigernde Entscheidungen treffen kann. Ein typisches Beispiel für ein solches Lastverhalten ist die Konfiguration 14. Bei dieser Konfiguration kann der Lastbalancierer der Ebene 0 keinen Laufzeitgewinn erzielen. Unter Verwendung der höheren Ebenen, insbesondere die Informationen der Ebene 3 (Laufzeitabschätzungen eines Anwendungsprozesses), kann die Laufzeit um 14% verbessert werden. Auf diesen Lauf wird im nachfolgenden Abschnitt noch gesondert eingegangen.
In Tabelle 35 ist für jede Konfiguration die Informationskombination angegeben, mit der die mini-

Konfig.	Referenzlauf	bestes Ergebnis		schlechtestes Ergebnis	
Laufzeitoptimierung einer einzelnen Anwendung					
1	496.51 Sek.	440.69 Sek.	89%	632.06 Sek.	127%
2	280.95 Sek.	237.18 Sek.	84%	314.01 Sek.	112%
3	193.47 Sek.	161.22 Sek.	83%	209.22 Sek.	108%
4	364.45 Sek.	318.77 Sek.	87%	415.49 Sek.	114%
5	228.78 Sek.	220.84 Sek.	97%	268.14 Sek.	117%
6	155.02 Sek.	149.73 Sek.	97%	202.25 Sek.	130%
Anwendungspulk I					
7	313.78 Sek.	282.22 Sek.	90%	339.68 Sek.	108%
8	247.21 Sek.	231.16 Sek.	94%	298.46 Sek.	121%
9	328.98 Sek.	325.95 Sek.	99%	371.82 Sek.	113%
10	346.69 Sek.	329.96 Sek.	95%	393.03 Sek.	113%
11	341.53 Sek.	339.89 Sek.	99.5%	390.40 Sek.	114%
12	348.08 Sek.	334.37 Sek.	96%	375.73 Sek.	108%
Anwendungspulk II					
14	470.46 Sek.	403.09 Sek.	86%	511.36 Sek.	109%
15	562.91 Sek.	482.78 Sek.	86%	698.03 Sek.	124%
16	735.56 Sek.	693.17 Sek.	94%	860.75 Sek.	117%
17	581.38 Sek.	537.56 Sek.	92%	646.48 Sek.	111%
Durchsatz					
18	360 Auftr.	379 Auftr.	105%	304 Auftr.	84%
19	(646,815) Sek.	(462,621) Sek.	(72,76)%	(728,903) Sek.	(113,111)%
20	122 Auftr.	151 Auftr.	124 %	126 Auftr.	103 %

Tabelle 33: Ergebnisse der höheren Informationsebenen.

male Laufzeit erreicht wird. Nachdem in nahezu jeder der 19 Konfigurationen das beste Ergebnis mit einer anderen Informationskombination erzielt wird, stellt sich die Frage, ob ein Anwender nun die jeweils beste Informationskombination für seine spezielle Anwendung von Hand ermitteln muß, oder ob dieser Prozeß der Optimierung zumindest teilweise automatisiert werden kann. Im Vorgriff auf das nächste Kapitel, in dem ausführlich auf diese Frage eingegangen wird, wird hier nur darauf hingewiesen, daß der Prozeß der Informationsoptimierung weitgehend automatisierbar ist.

Bei einer Analyse der Meßergebnisse in Tabelle 33 stellt man fest, daß es mit Hilfe der höheren Informationsebenen möglich ist, in 9 der 19 Konfigurationen Ergebnisverbesserungen von mindestens 10% zu erzielen. Da es nahezu unmöglich ist, bei den komplexen Laufzeitverhalten der verwendeten Konfigurationen zu ermittelen, warum es mit Hilfe der Informationsebenen möglich ist, die Laufzeit bzw. den Durchsatz einer bestimmten Konfiguration um wenige Prozent zu verbesseren, werden im nachfolgenden nur diese oben erwähnten 9 Konfigurationen weiterbetrachtet.

Prinzipiell gibt es zwei Möglichkeiten, wie ein Lastbalancierer unter Verwendung der höheren Informationsebenen Ergebnisverbesserungen erzielen kann. Entweder ist er in der Lage, frühzeitiger auf Lastungleichgewichte zu reagieren (siehe Abschnitt 7.4), oder aber er trifft bessere

Konfig.	Referenzlauf			minimale Laufzeit		
	Anfragen	**Migrationen**	**Fehler**	**Anfragen**	**Migrationen**	**Fehler**
Laufzeitoptimierung einer einzelnen Anwendung						
1	43	13	5 (38%)	35	7	2 (28%)
2	33	19	5 (26%)	16	9	1 (11%)
3	39	32	5 (22%)	35	17	2 (12%)
4	49	3	2 (66%)	44	3	1 (33%)
Anwendungspulk I						
7	78	45	14 (31%)	92	39	6 (15%)
Anwendungspulk II						
15	124	51	25 (49%)	57	25	1 (4%)
Durchsatz						
19	113	111	38 (34%)	110	108	31 (28%)
20	108	106	28 (26%)	53	51	12 (23%)
Durchschn. Anteil von Fehlmigrationen: **37%**				Durchschn. Anteil von Fehlmigrationen: **19%**		

Tabelle 34: Auswirkung der Informationsebenen auf die Qualität der Entscheidungen.

Entscheidungen. In Tabelle 34 ist die Qualität der Migrationsentscheidungen für den Referenzlauf und den Lauf der höheren Ebenen mit dem besten Ergebnis anhand von drei Kriterien angegeben:

- Anzahl von Migrationsanfragen, die von unterbelasteten Blattkomponenten während des Programmlaufes gestellt wurden
- Anzahl der durchgeführten Prozeßmigrationen
- Anzahl der Prozeßfehlmigrationen

Unter einer *Prozeßfehlmigration* wird eine Migration verstanden, die nach Ablauf einer minimalen Verweilzeit revidiert wird, ohne daß sich dadurch der Lastzustand auf den beteiligten Knoten sigifikant verbessert, d.h. wenn der Lastzustand des Knotens nicht von *unterbelastet* in *normal belastet* wechselt. Zur Bestimmung der Prozeßfehlmigrationen wird das graphische Auswertungswerkzeug PaStatTool verwendet.
Wie in Tabelle 34 angegeben, sind über ein Drittel der Migrationenscheidungen bei einem Lastbalancierer der Ebene 0 Fehlentscheidungen. Diese erhöhen die durch das Lastbalancierungssystem verursachte Systemlast, ohne sich positiv auf die Lastverteilung und damit auf die Ressourcenauslastung auszuwirken. Dabei muß jedoch noch einmal darauf hingewiesen werden, daß es trotz dieser hohen Anzahl von Fehlentscheidungen mit einem Lastbalancierer der Ebene 0 bereits möglich ist, zum Teil drastische Ergebnisverbesserungen zu erzielen (vgl. Tabelle 23 auf Seite 102). Natürlich ist es möglich, durch eine defensivere Einstellung der Lastbalancierungsstrategie, d.h. im Falle des PaLaBer-Systems durch eine Erhöhung der Schwellwerte, die Anzahl der Fehlmigrationen zu verringern. Eine defensivere Einstellung der Strategie bewirkt jedoch nicht nur eine Verringerung der Fehlentscheidungen, sondern dadurch werden möglicherweise auch notwendige Entscheidungen verhindert, so daß sich eine defensivere Einstellung der Lastbalancierungsstrategie negativ auf die Leistungsfähigkeit des Gesamtsystems auswirkten kann.
Vergleicht man nun den besten Lauf, der mit Hilfe der höheren Ebenen erzielt wird, mit dem Referenzlauf, so stellt man fest, daß es bei der Verwendung der höheren Informationsebenen nicht nur

Konfiguration	Informationsebene 0		Informationsebene 1			Informationsebene 2	Informationsebene 3		
	Prozessorauslastung Hauptspeicherauslastung etc.	Berücksichtigung der Kommunikations-beziehungen	Statische Plazierungsinformation	Statistischer Prozessor-/ Hauptspeicherbedarf	Statische Kommunikationspartner	Kommunikationsphase Ein-/Ausgabephase Rechenphase	Vorzeitige Ankündigung der Terminierung	Abschätzung der Restlaufzeit	Durchschnittlicher Prozessorbedarf innerhalb der nächsten Bearbeitungsphase
Laufzeitoptimierung einer einzelnen Anwendung									
1	♦	♦	♦				♦	♦	♦
2	♦	♦	♦				♦		♦
3	♦	♦	♦	♦	♦	♦	♦		♦
4	♦	♦	♦	♦	♦	♦	♦	♦	♦
5	♦	♦	♦	♦	♦	♦	♦	♦	
6	♦	♦			♦			♦	♦
Anwendungspulk I									
7	♦	♦	♦				♦	♦	♦
8	♦	♦	♦				♦		
9	♦	♦	♦	♦	♦	♦	♦		♦
10	♦	♦	♦	♦	♦	♦		♦	
11	♦	♦	♦	♦	♦	♦	♦	♦	♦
12	♦	♦				♦	♦		♦
Anwendungspulk II									
14	♦	♦	♦		♦	♦	♦	♦	
15	♦	♦	♦						♦
16	♦	♦	♦	♦	♦	♦			
17	♦	♦		♦	♦		♦	♦	
Durchsatz									
18	♦	♦	♦	♦		♦		♦	
19	♦	♦	♦			♦	♦	♦	♦
20	♦	♦	♦		♦	♦			

Tabelle 35: Informationskombinationen mit dem jeweils besten Ergebnis.[a]

a. Das Zeichen ♦ deutet die Verwendung der jeweiligen Information an.

möglich ist, die Anzahl der durchgeführten Migrationen zum Teil erheblich zu senken (vgl. dazu Konfigurationen 3, 15 und 20), sondern auch die Qualität der Migrationsentscheidungen signifikant zu verbessern. Im Vergleich zu einem Lastbalancierer der Ebene 0 kann die Anzahl der Fehlmigratio-

Konfiguration	Referenzlauf ohne Migration		minimale Laufzeit (in Prozent)	
	Random	**Cluster**	**Random**	**Cluster**
Laufzeitoptimierung einer einzelnen Anwendung				
1	784.20 Sek.	647.95 Sek.	56	68
2	397.55 Sek.	328.76 Sek.	60	72
3	204.39 Sek.	181.56 Sek.	79	89
4	401.17 Sek.	413.45 Sek.	79	77
5	259.31 Sek.	261.88 Sek.	85	84
6	170.44 Sek.	176.39 Sek.	88	85
Anwendungspulk I				
7	406.53 Sek.	411.21 Sek.	69	69
8	260.24 Sek.	260.49 Sek.	89	89
9	477.41 Sek.	484.39 Sek.	68	67
10	486.26 Sek.	454.66 Sek.	68	73
11	489.79 Sek.	472.29 Sek.	69	72
12	493.67 Sek.	463.78 Sek.	68	72
Anwendungspulk II				
14	471.02 Sek.	473.60 Sek.	86	85
15	789.91 Sek.	657.54 Sek.	61	73
16	806.55 Sek.	782.87 Sek.	86	89
17	653.12 Sek.	642.77 Sek.	82	84
Durchsatz				
18	294 Auftr.	291 Auftr.	129	130
19	(685.2,848.8) Sek.	(720.1,901) Sek.	(67,73)	(64,69)
20	118Auftr.	126Auftr.	128	120
Der prozentuale Gewinn relativ zum Referenzlauf *Random* liegt zwischen **11%** und **44%**				
Der prozentualer Gewinn relativ zum Referenzlauf *Cluster* liegt zwischen **11%** und **36%**				

Tabelle 36: Laufzeitgewinne im Vergleich zu Referenzläufen ohne dynamische Lastbalancierung.

nen um über 50% verringert werden, d.h. ein Lastbalancierer der höheren Ebenen kann die ihm zur Verfügung stehende Information in qualitativ bessere Lastbalancierungsentscheidungen umsetzen. Die Konfiguration 20 weicht von diesem Schema allerdings erheblich ab, da hier die Anzahl der prozentualen Fehlmigrationen nur um 3% abnimmt. Da bei dieser Durchsatzkonfiguration jedoch nicht nur die Anzahl der Migrationsanforderungen sowie die Anzahl der durchgeführten Migrationen halbiert wurde, sondern auch die Anzahl der Fehlmigrationen um die Hälfte reduziert wurde, schlug sich auch hier die Verwendung der höheren Informationsebenen in qualitativ besseren Entscheidungen nieder.

Die wohl wichtigste Erkenntnis dieser Arbeit ist in Tabelle 36 und Abbildung 42 zusammengefaßt. Während in Tabelle 33 die Gewinne der höheren Informationsebenen im Vergleich zu einem Referen-

zlauf der Ebene 0 angegeben wurden, werden in Tabelle 36 die erzielten Gewinne mit zwei Referenzläufen ohne dynamische Lastbalancierung verglichen. Beim Referenzlauf *Random* werden die Prozesse der parallelen Anwendung mittels eines Zufallsgenerators unter Berücksichtigung der aktuellen Last im System auf die Blattkomponenten verteilt, im Referenzlauf *Cluster* wird der Clustering-Algorithmus des PaLaBer-Systems verwendet (vgl. dazu Kapitel 6). Wie in Abbildung 42 dargestellt ist, können mit Hilfe eines Lastbalancierers der Ebene 0, trotz der heterogenen Arbeitslast der 19 Konfigurationen, Gewinne von bis zu 37% erzielt werden. Unter Verwendung von statischer Plazierungsinformation können diese Gewinne noch weiter ausgebaut werden (2% bis 43%).
In der Literatur (vgl. dazu z.B. [Beck95], [Ludw93]) werden für einen dynamischen Lastbalancierer in der Regel Gewinne zwischen 10% und 20% als realistisch angesehen[1]. Allerdings handelt es sich bei diesen referenzierten Arbeiten häufig um anwendungsspezifische Lastbalancierungansätze, d.h. um Lastbalancierungsansätze, die nur ein bestimmtes Lastverhalten unterstützen. Dementsprechend ist es nicht verwunderlich, daß der Lastbalancierer der Ebene 0 nicht in der Lage ist, unter Berücksichtigung der heterogenen Arbeitslast der 19 Konfigurationen, diesen Leistungsbereich zu erreichen. Wird ein Lastbalancierer der Ebene 0 jedoch mit den Informationen der höheren Ebenen versorgt, kann er problemlos in den Leistungsbereich eines anwendungsspezifischen Lastbalancierers vordringen. Wichtig in diesem Zusammenhang ist dabei, daß es sich insbesondere bei den Informationsebenen 2 und 3 um Informationen handelt, die die Anwendungen kennen bzw. zur Laufzeit erfassen können.
Anders gesagt: Mit Hilfe der höheren Informationsebenen kann ein Lastbalancierer auf ein bestimmtes Laufzeitverhalten hin optimiert werden. Dabei muß die Lastbalancierungsstrategie nicht verändert werden, da die Optimierung der Lastbalancierungsstrategie von außen durch die Versorgung des Lastbalancierers mit entscheidungsrelevanter Information erfolgt. Somit kann ein dynamischer Lastbalancierer als eine Black-Box betrachtet werden, die durch die Informationsschnittstelle zu der Anwendung auf ein Lastverhalten hin optimiert werden kann.
Auf die Frage, wie die jeweils beste Informationskombination für eine gegebene Anwendungslast ermittelt werden kann, wird im nächsten Kapitel ausführlich eingegangen.

7.4 Auswertung ausgewählter Einzelläufe

In diesem Abschnitt wird die Verwendung der Informationen der höheren Ebenen an zwei exemplarisch ausgewählten Konfigurationen, der Konfiguration 1 und der Konfiguration 14, beschrieben. Es wurden diese ausgewählt, da sie nicht nur ein sehr unterschiedliches Lastverhalten aufweisen, sondern es sich hierbei um Konfigurationen mit einer geringen Knotenanzahl handelt.

7.4.1 Laufzeitauswertung der Konfiguration 1

Bei der Konfiguration 1 handelt es sich um die Strömungs-Anwendung mit 8 Prozessen verteilt auf 6 Blattkomponenten (siehe Abbildungen 43 und 44). Das Lastverhalten dieser Konfiguration läßt sich in zwei Phasen einteilen. In jeder Phase gibt es, aufgrund der Wahl des Berechnungsgitters der Anwendungsprozesse, eine optimale Lastverteilung. Wie in Abbildung 43 dargestellt, ist bereits der

1. Bei dieser Angabe handelt es sich nur um einen groben Richtwert, da die tatsächlich erzielbaren Leistungssteigerungen entscheidend von den Anwendungen und der Systemcharakteristik abhängen.

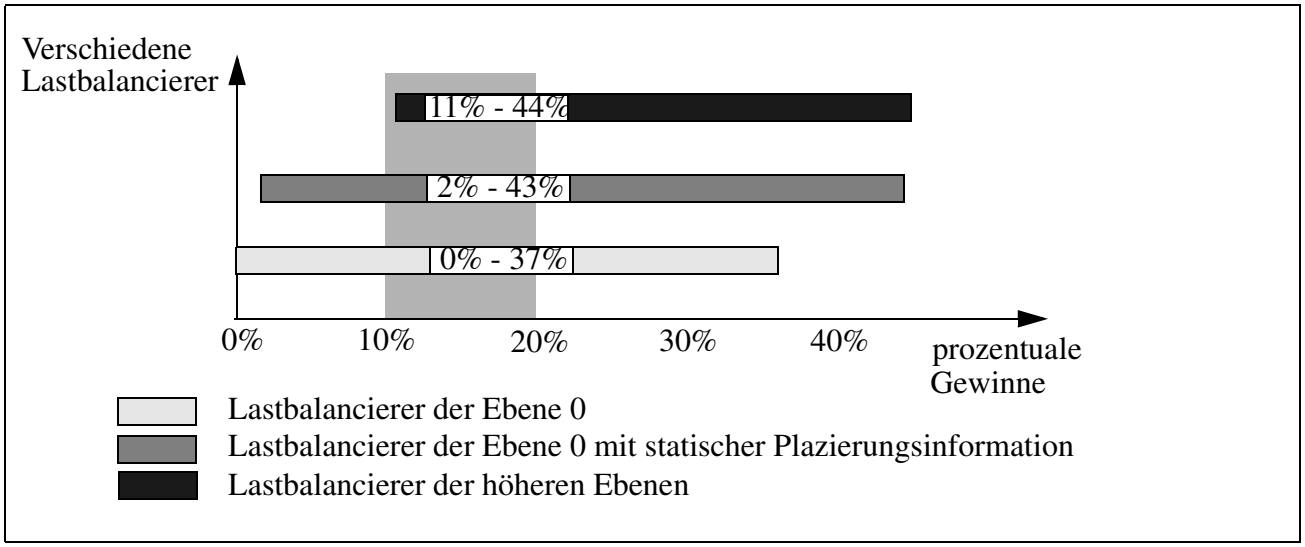


Abbildung 42: Vergleich von erzielten Laufzeitgewinnen.

Lastbalancierer der Ebene 0 in der Lage, in beiden Berechnungsphasen der Anwendung eine stabile Lastverteilung zu erreichen, bei der alle Knoten nahezu vollständig ausgelastet sind. Um mit Hilfe der höheren Informationsebenen einen zusätzlichen Laufzeitgewinn zu erzielen, muß ein entsprechender Lastbalancierer diese Lastverteilung schneller erreichen.

Der Lastbalancierer der Ebene 0 benötigt speziell zum Erreichen der ersten stabilen Lastverteilung unnötig viele Lasttransfers. Dafür gibt es zwei Gründe:

1. Bei einem Lastbalancierer der Ebene 0 werden die Anwendungsprozesse mittels eines Zufallsgenerators auf die verfügbaren Blattkomponenten verteilt. In diesem speziellen Fall führt die zufällige Verteilung der Anwendungsprozesse dazu, daß zwei Knoten (Knoten 4 und 5) von Beginn an überbelastet sind.
2. Zu Beginn der Berechnung ist das Lastverhalten der Anwendungsprozesse häufig noch nicht deutlich genug ausgeprägt, so daß der Lastbalancierer aufgrund ungenauer Informationen zu falschen Lasttransferentscheidungen kommt.

Insgesamt dauert die Einschwingphase 1 bei einem Lastbalancierer der Ebene 0 27 Meßintervalle, wobei ein Meßintervall 2.7 Sekunden entspricht. Um von der stabilen Lastverteilung der Phase 1 in die stabile Lastverteilung der Phase 2 zu gelangen, sind im optimalen Fall 4 Prozeßmigrationen erforderlich, da, wie in Abbildung 45 dargestellt, ein Austausch von Anwendungsprozessen erfolgen muß. Der Lastbalancierer der Ebene 0 benötigt dafür nur eine Prozeßmigration mehr, allerdings dauert die Einschwingphase 2 17 Meßintervalle lang. Außerdem findet am Ende der Berechnungsphase 2 noch eine unnötige Prozeßmigration zwischen Knoten 0 und Knoten 5 statt, da der Lastbalancierer der Ebene 0 keine Informationen über die Gesamtlaufzeit einer Anwendung bzw. eines einzelnen Anwendungsprozesses besitzt. Für die Abarbeitung dieser Anwendung benötigt der Lastbalancierer der Ebene 0 insgesamt 496.51 Sekunden.
In Abbildung 44 ist das Laufzeitprofil der Konfiguration unter Verwendung der höheren Informationsebenen mit der minimalen Laufzeit von 440.69 Sekunden dargestellt. Der Lastbalancierer der höheren Ebenen verwendet dabei die folgenden Informationen (vgl. Tabelle 35 auf Seite 127):

- statische Plazierungsinformation

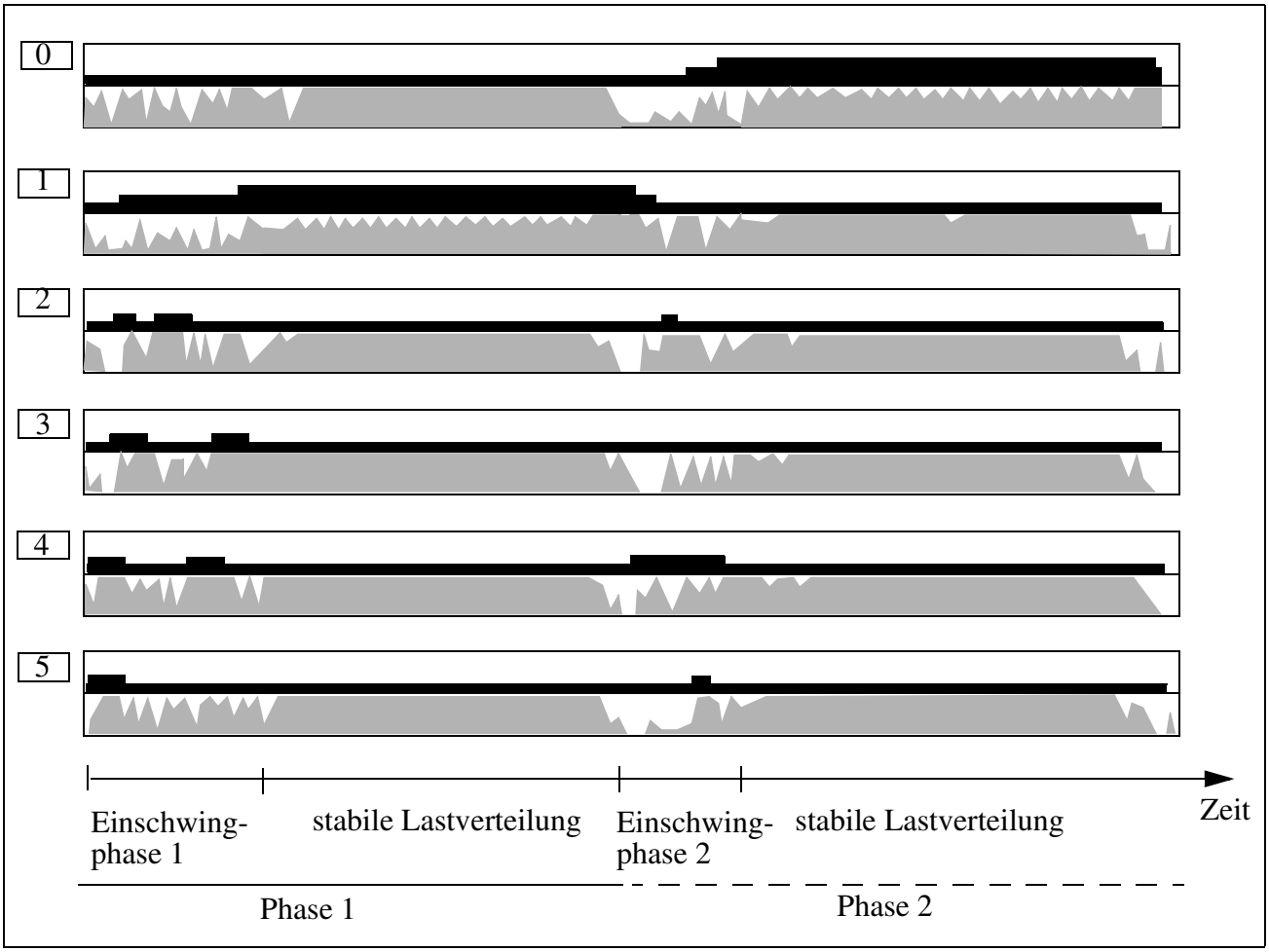


Abbildung 43: Konfiguration 1 - Referenzlauf

- vorzeitige Ankündigung der Terminierung
- Abschätzung der Restlaufzeit
- durchschnittlicher Prozessorbedarf innerhalb der nächsten Bearbeitungsphase

Ausgehend von der statischen Plazierungsinformation ist der Lastbalancierer in der Lage, eine bessere initiale Prozessverteilung zu erzielen. In Abbildung 44 befindet sich kurz nach dem Start nur der Knoten 5 in einem *überbelasteten* Lastzustand. Ausgehend von dieser besseren Ausgangslage kann der Lastbalancierer die Information bezüglich des Prozessorbedarfs innerhalb der nächsten Verarbeitungsphase dazu verwenden, innerhalb von 5 Meßintervallen die erste stabile Lastverteilung zu erreichen. Zur Erinnerung sei darauf hingeweisen, daß der Lastbalancierer der Ebene 0 zur Erreichung der ersten stabilen Lastverteilung 27 Meßintervalle benötigte.
Wenn am Ende der ersten stabilen Lastverteilung die Gitterumschaltung erfolgt, teilen dies die Anwendungsprozesse vor der Umschaltung dem Lastbalancierer mit. Diese Mitteilung versetzt den Lastbalancierer in die Lage, den zweiten stabilen Lastzustand ein Meßintervall schneller zu erreichen, obwohl er die gleiche Anzahl von Lasttransfers benötigt.
Durch die Informationen bzgl. der Gesamtlaufzeit der parallelen Anwendung werden im Vergleich zum Referenzlauf auch am Ende der zweiten stabilen Lastverteilung keine Anwendungsprozesse mehr transferiert. Insgesamt kann mit Hilfe der höheren Informationsebenen die Laufzeit um 11% gegenüber dem Referenzlauf verringert werden.

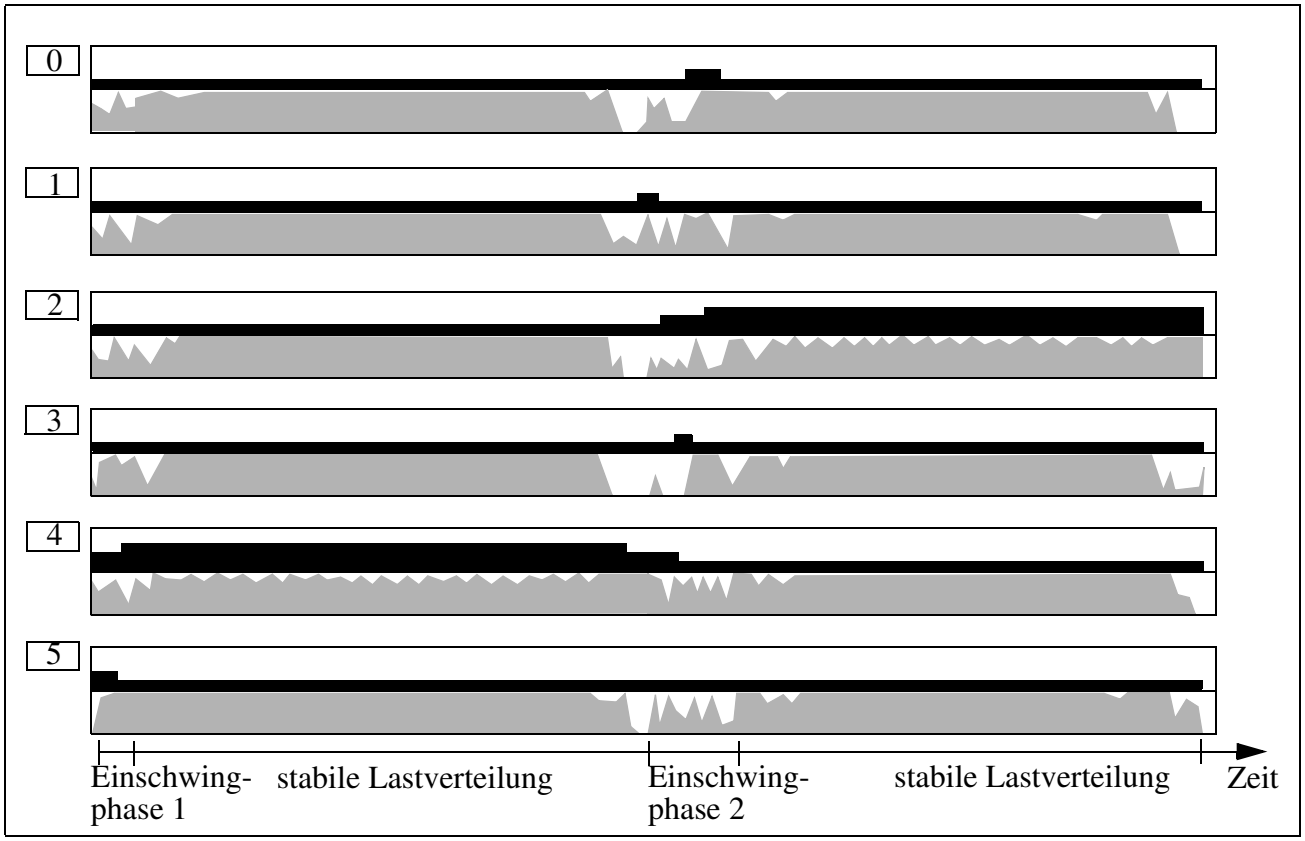


Abbildung 44: Konfiguration 1 - beste Informationskombination.

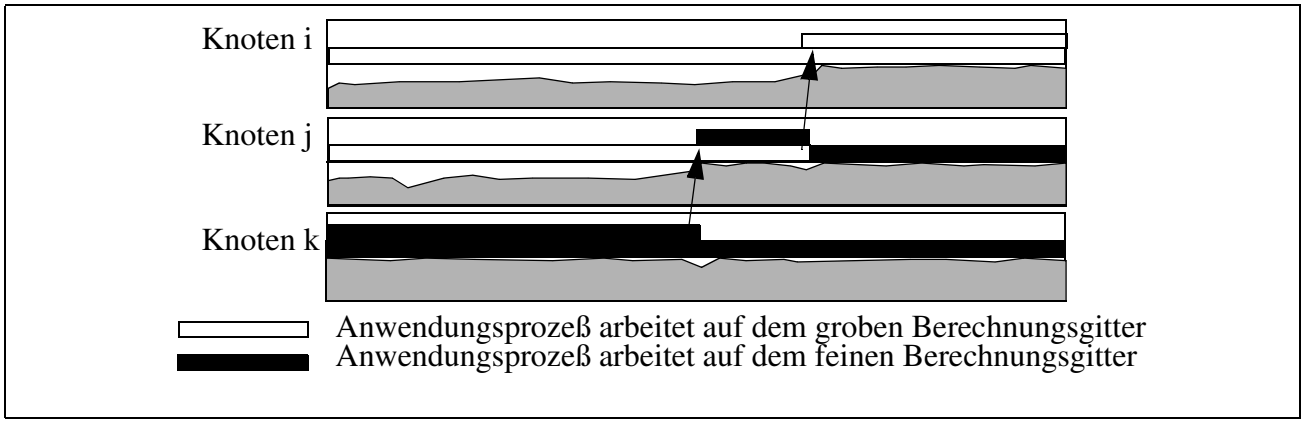


Abbildung 45: Austausch von Anwendungsprozessen.

7.4.2 Laufzeitauswertung der Konfiguration 14

Die zweite Konfiguration, die in diesem Abschnitt ausführlicher beschrieben wird, ist die Konfiguration 14. Dabei handelt es sich um eine Konfiguration der Klasse Anwendungspulk II. Bei dieser Konfiguration werden zwei parallele Raytracer-Anwendungen mit jeweils 12 Prozessen verteilt auf 9 Blattkomponenten hintereinander gestartet. Der Referenzlauf der Ebene 0 ist in Abbildung 46 dargestellt. Bei diesem Laufzeitprofil fallen zwei Laufzeitcharakteristiken sofort auf:

- Es finden Prozeßfehlmigrationen statt (siehe Pfeile).
- Im letzten Drittel der Bearbeitung der ersten Anwendung treten hohe Leerlaufzeiten auf 7 Knoten auf.

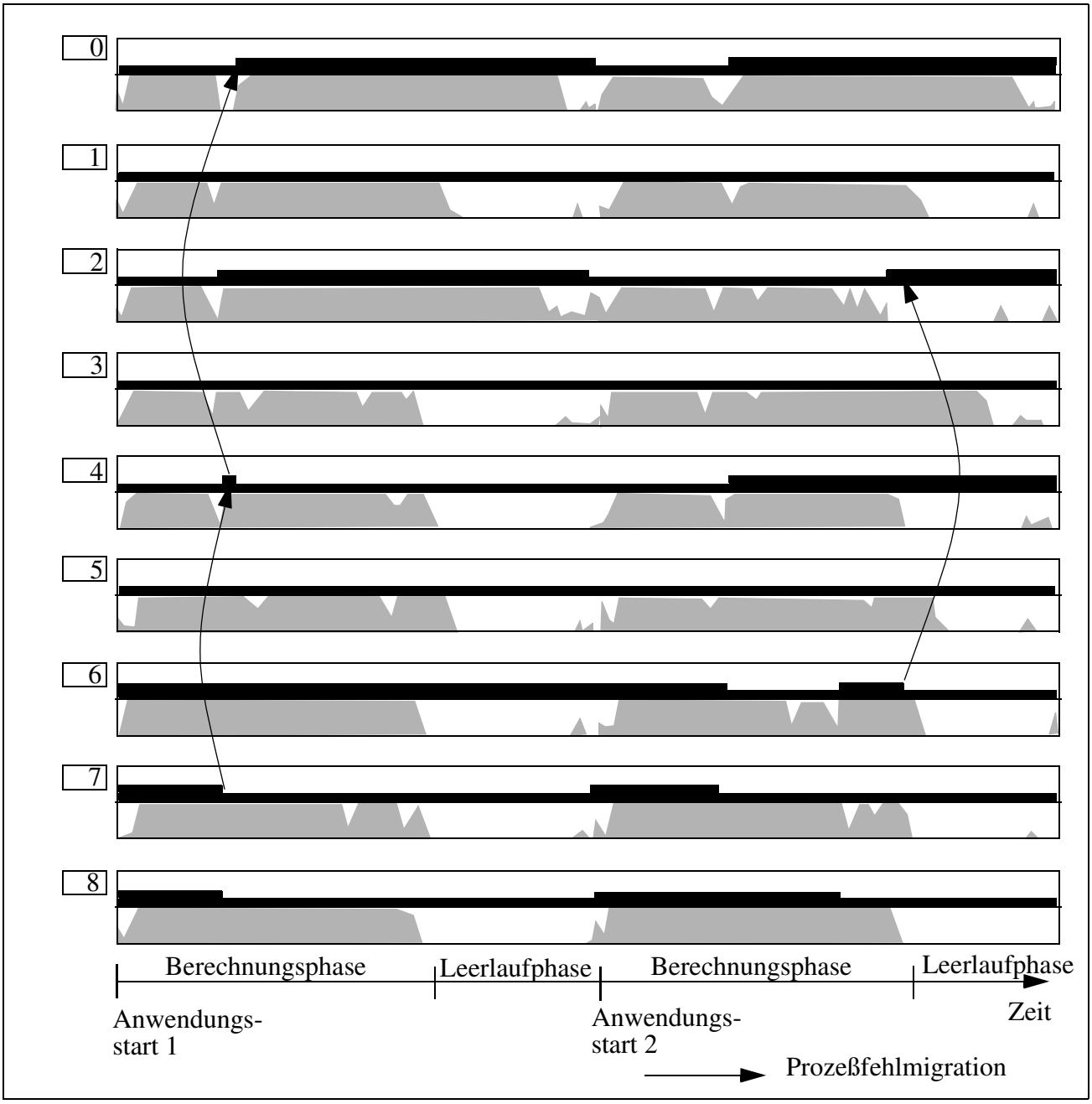


Abbildung 46: Konfiguration 14 - Referenzlauf.

Dabei wirken sich vor allem die hohen Leerlaufzeiten sehr negativ auf die Laufzeit der Konfiguration aus. Der Grund, warum der Lastbalancierer der Ebene 0 nicht auf diese Lastungleichgewichte durch den Start der nächsten Anwendung reagiert, ist, daß ein Lastbalancierer der Ebene 0 nicht erkennen kann, ob die Leerlaufzeiten ihre Ursache in lokalen bzw. globalen Synchronisationspunkten einer parallelen Anwendung haben oder in deren Terminierung. Aus diesem Mangel an Wissen über das Anwendungsverhalten ist ein Lastbalancierer der Ebene 0 gezwungen, auf die explizite Terminierung einer Anwendung zu warten, bevor er die nächste parallele Anwendung startet.
Die auftretenden Prozeßfehlmigrationen im Referenzlauf lassen sich ebenfalls auf mangelnde Informationen über das Anwendungsverhalten zurückführen. Ein sehr anschauliches Beispiel ist dafür die Prozeßmigration von Knoten 6 auf Knoten 2 bei der Abarbeitung der zweiten parallelen Anwendung. Während Knoten 2 mit einem Anwendungsprozeß frühzeitig leerläuft, ist Knoten 6 mit zwei

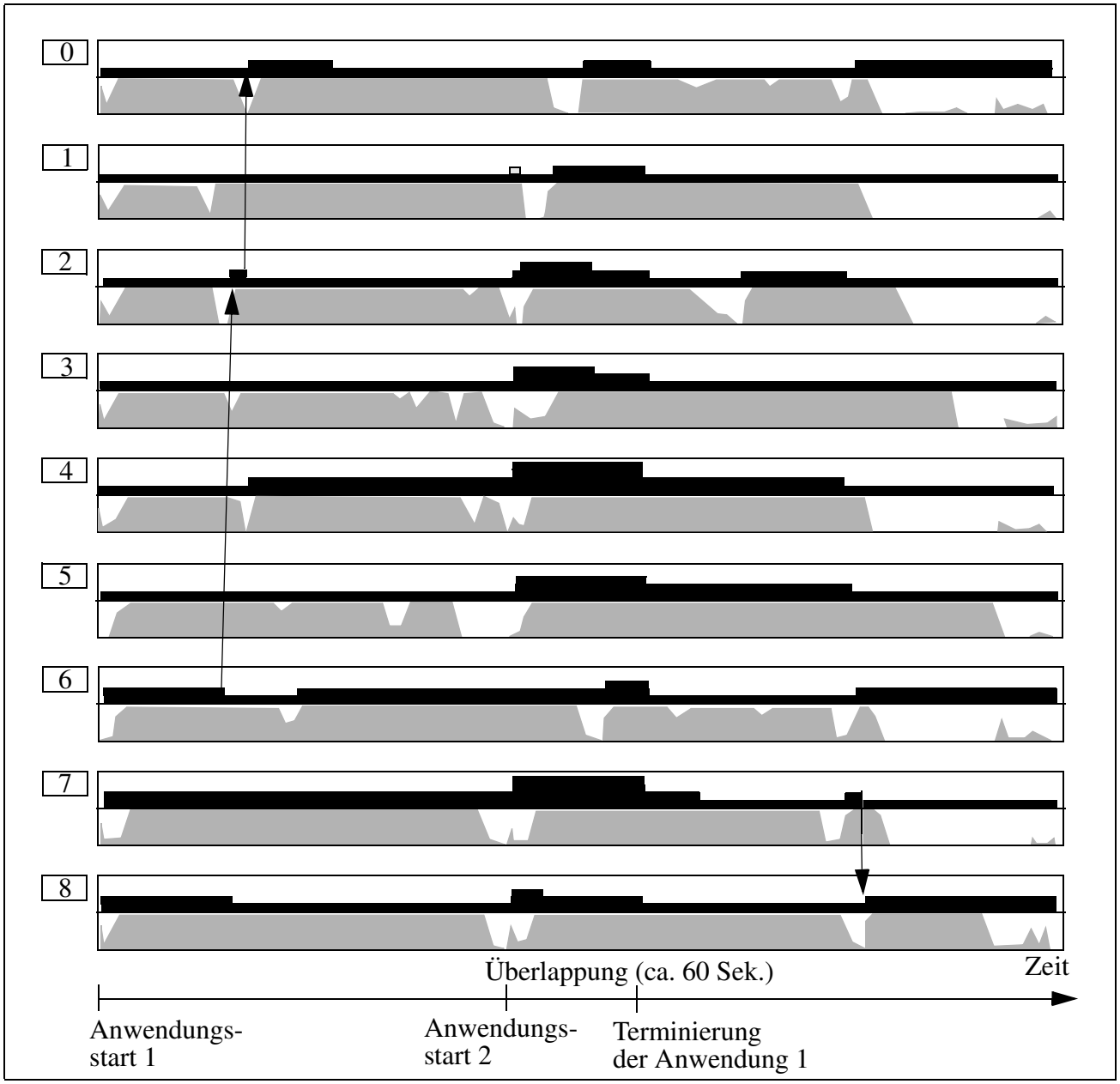


Abbildung 47: Konfiguration 14 - vorzeitige Ankündigung der Terminierung.

Anwendungsprozessen gut ausgelastet. Darauf veranlaßt der Lastbalancierer der Ebene 0 einen Lasttransfer zwischen diesen beiden Knoten. Kurz nach der Migration eines Prozesses von Knoten 6 auf Knoten 2 beendet der verbleibende Prozeß auf Knoten 6 seine Berechnungen, so daß nun auch Knoten 6 leerläuft. Da der migrierte Prozeß kurz nach seiner Migration auf Knoten 2 seine Berechnungen ebenfalls einstellt, wurde durch diese Lasttransferentscheidung der Lastzustand weder auf dem Ziel- noch auf dem Quellknoten verbessert, weshalb es sich hierbei um eine Fehlentscheidung handelt. Hätte der Lastbalancierer hingegen über Informationen bzgl. der Berechnungszeiten der Anwendungsprozesse verfügt (Informationen der Ebene 1 oder 3), hätte die Blattkomponente von Knoten 6 die Migrationsaufforderung von Knoten 2 ablehnen können.
Bei der Migration eines Anwendungsprozesses von Knoten 7 über Knoten 4 zu Knoten 0 handelt es sich ebenfalls um eine Fehlentscheidung, da der Serverprozeß auf Knoten 4 unmittelbar nach der erfolgten Migration wieder einen Bearbeitungsauftrag zugewiesen bekommt und somit alleine in der

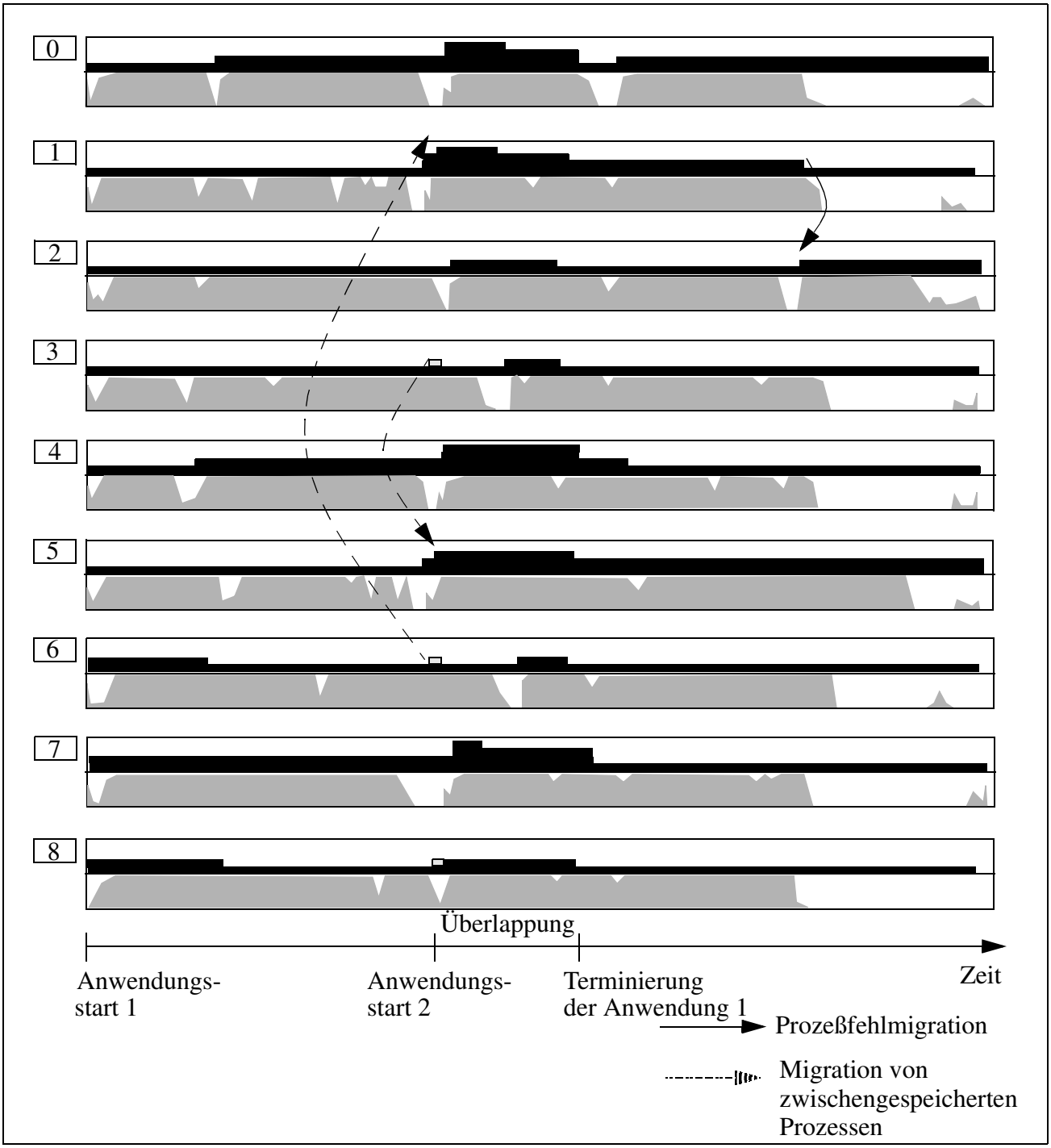


Abbildung 48: Informationskombination 14 - bestes Laufzeitergebnis der höheren Ebenen.

Lage ist, den Prozessor vollständig auszulasten. Stattdessen hätte die Migration direkt von Knoten 7 zu Knoten 0 erfolgen müssen.

Anhand von Abbildung 47 wird nun aufgezeigt, wie mit Hilfe der Information bzgl. der vorzeitigen Terminierung ein Lastbalancierer in die Lage versetzt wird, zwischen lokalen bzw. globalen Synchronisationspunkten und dem Berechnungsende einer parallelen Anwendung zu unterscheiden. Während beim Referenzlauf im letzten Drittel des Anwendungslaufes hohe Prozessorleerlaufzeiten auftreten, teilen die Anwendungsprozesse bei diesem Lauf am Ende ihrer Berechnung dem Lastbalancierer mit, daß sie bis zu ihrer endgültigen Terminierung keinen nennenswerten Prozessorbedarf mehr haben

werden. Dadurch ist der Lastbalancierer in der Lage, die zweite Anwendung 60 Sekunden früher zu starten als beim Referenzlauf. Damit kann die Laufzeit der gesamten Konfiguration um ca. 50 Sekunden verkürzt werden. Allerdings finden auch bei diesem Lauf mehrere unproduktive Prozeßmigrationen statt (z.B. von Knoten 7 auf Knoten 8 bzw. von Knoten 6 über Knoten 2 zu Knoten 0).

Beim besten Laufzeitergebnis, dargestellt in Abbildung 48, das bei dieser Konfiguration durch die Verwendung der höheren Informationsebenen (siehe Tabelle 35) erzielt werden kann, wird nicht nur durch den vorzeitigen Start der zweiten Anwendung bereits eine deutliche Laufzeitverbesserung erzielt, sondern auch durch die Abschätzung der Restlaufzeit die Anzahl der Prozeßfehlmigrationen auf eine reduziert. Diese Fehlmigration von Knoten 1 auf Knoten 2 resultiert aus der Tatsache, daß es sich bei den Informationen der Ebene 3 um Prognosen handelt, die immer einer gewissen Ungenauigkeit unterliegen. So war die Abschätzung der Restlaufzeit des Prozesses auf Knoten 1, der nicht migriert wurde, um mehrere Sekunden zu hoch, so daß die Blattkomponente davon ausging, daß sie auch nach der Abgabe eines Anwendungsprozesses noch ausreichend ausgelastet sei. Da die Nachricht von der vorzeitigen Terminierung dieses Prozesses erst unmittelbar nach der Initiierung der Migration bei der Blattkomponente ankam, hatte die Blattkomponente keine Möglichkeit mehr, ihre Entscheidung rückgängig zu machen.

Kapitel 8

Ausblick

Nachdem im letzten Kapitel, basierend auf umfangreichen Messungen, aufgezeigt wurde, daß sich die Verwendung der höheren Informationsebenen für die anwendungsspezifische Lastbalancierung lohnt, wird im nächsten Abschnitt auf die automatische Selektion der Informationskombination mit dem besten Laufzeitergebnis für eine gegebene Arbeitslast eingegangen. Das Kapitel schließt mit möglichen Weiterentwicklungen der vorgestellten Arbeit.

8.1 Automatische Selektion der Informationskombinationen

In Kapitel 7 wurde aufgezeigt, daß sich für die anwendungsspezifische Lastbalancierung die Verwendung der höheren Informationsebenen als lohnend erwiesen hat. In diesem Abschnitt wird nun aufgezeigt, inwieweit die Selektion der besten Informationskombination für eine Anwendungskonfiguration automatisiert, d.h. transparent für den Anwender durchgeführt werden kann. Die Selektion der Informationskombination besteht aus drei Phasen:

1. Instrumentierungsphase
2. Aktivierungsphase
3. Auswahlphase

Die *Instrumentierung* des Programmtextes mit den Routinen zur Erfassung und Weiterleitung der Informationen der verschiedenen Ebenen ist nicht vollständig automatisierbar, da die Vorhersage des Prozeßverhaltens unentscheidbar ist. Sie ist unentscheidbar, da bereits ein Teilproblem, das sogenannte *Halteproblem nach Kleene* [HoUl88] unentscheidbar ist. Die Instrumentierung des Programmtextes hat sich im übrigen als relativ unkritisch erwiesen, da der Anwendungsprogrammierer die Möglichkeit hat, die Informationen an den Lastbalancierer in bekannten Größen für die Anwendung auszudrücken.
Die *Aktivierung* der Routinen zur Informationssammlung bzw. Informationsweiterleitung ist im Vergleich zur Instrumentierung in einer transparenten Art und Weise möglich, da bestehende Betriebssysteme dazu bereits Möglichkeiten vorsehen (vgl. [Holl94]), indem sie Routinen zur Manipulation von Programmcodes zur Laufzeit bereitstellen. Im Falle des Betriebssystems Unix handelt es

sich hierbei um die Routine ptrace() [OSF92], die die Programmcodemodifikation zur Laufzeit ermöglicht.
Die *Auswahl* der Informationskombination mit dem besten Laufzeitergebnis ist für den Fall der anwendungsorientierten Lastbalancierung, d.h. wenn es sich um ein reproduzierbares Lastverhalten handelt, vollständig automatisierbar. Die Auswahl ist trivialerweise automatisierbar, da es nur eine endliche Anzahl von Informationskombinationen gibt, die bei reproduzierbaren Anwendungsverhalten notfalls sequentiell von einem Lastbalancierer durchgetestet werden kann. In diesem Zusammenhang sind jedoch eine Vielzahl von Optimierungen möglich, da es einen engen Zusammenhang zwischen dem Laufzeitprofil einer parallelen Anwendung bzw. Anwendungskombination und der *besten* Informationskombination gibt. Anhand der nachfolgenden Graphiken wird aufgezeigt, daß Anwendungslasten mit „ähnlichem“ Laufzeitprofil auch „ähnliche Informationskombinationen“ benötigen bzw. Anwendungslasten mit stark abweichenden Lastprofilen auch unterschiedliche Informationskombinationen benötigen. Das Laufzeitprofil wird wiedergegeben durch:

- Anzahl der Meßintervalle mit vergleichbarer Prozessorauslastung
- Länge der Intervalle mit vergleichbarer Prozessorauslastung

In Abbildung 49 sind die Laufzeitprofile der Konfigurationen 4 und 5 aus Tabelle 20 auf Seite 98 dargestellt. Bei diesen Konfigurationen handelt es sich um die Raytracing-Anwendung mit sechs bzw. neun Blattkomponenten. In Abbildung 49 a) sind bei beiden Konfigurationen deutlich die zwei Häufungspunkte zu erkennen. Beide Konfigurationen haben viele Meßintervalle mit einer sehr hohen Prozessorauslastung bzw. einer sehr geringen Prozessorauslastung. Die Häufungspunkte spiegeln die Tatsache wider, daß ein Prozeß der Raytracing-Anwendung sein Prozessorquantum vollständig ausnutzen kann, solange er noch Bildpunkte zu berechnen hat. Verfügt ein Prozeß dieser Anwendung über keinen Berechnungsauftrag, nimmt er keine Prozessorzeit mehr in Anspruch. In Abbildung 49 b) ist zu erkennen, daß diese Konfigurationen über relativ lange Berechnungsintervalle mit homogener Prozessorauslastung verfügen. Beide Konfigurationen erreichen ihre besten Laufzeitergebnisse bei der Verwendung vergleichbarer Informationskombinationen (siehe dazu Tabelle 35 auf Seite 127).
In Abbildung 50 sind die Laufzeitprofile der Konfigurationen 1 und 2 dargestellt. Zur Erinnerung sei gesagt, daß es sich bei den Konfigurationen 1 und 2 um die Strömungs-Anwendung handelt mit sechs bzw. zwölf Blattkomponenten. Auch diese Konfigurationen weisen ein vergleichbares Laufzeitprofil auf, sowohl in der Anzahl der Meßintervalle mit vergleichbarer Prozessorauslastung als auch bei der Länge der Meßintervalle mit homogener Prozessorauslastung. Beide Konfigurationen erreichen ihre besten Laufzeitergebnisse bei der Verwendung der Informationsebene 3, da sich der Wechsel des Bearbeitungsgitters (vgl. Abschnitt 6.1.1) signifikant auf das Laufzeitverhalten der Anwendung auswirkt und somit eine wichtige Information für einen dynamischen Lastbalancierer darstellt.
Vergleicht man die Laufzeitprofile in Abbildung 49 und 50, erkennt man, daß Konfigurationen mit unterschiedlichen Informationskombinationen auch unterschiedliche Laufzeitprofile aufweisen. Eine mögliche Optimierung zur automatischen Selektion der besten Informationskombination besteht nun darin, daß der Lastbalancierer sich von jeder Anwendung/Anwendungskombination das Laufzeitprofil der Ebene 0 zusammen mit der Information über die beste Informationskombination abspeichert. Wird dem Lastbalancierer eine neue Anwendungskonfiguration zur Ausführung übergeben, führt er einen Lauf basierend auf den Informationen der Ebene 0 durch und ermittelt das Laufzeitprofil der Anwendung. Im Anschluß an den Lauf kann er mit Hilfe des ermittelten Laufzeitprofils die beste Überdeckung mit bereits bekannten Laufzeitprofilen ermitteln und die entsprechende Informationskombination als Startkombination für den nächsten Lauf verwenden. Durch iterative Veränderungen, d.h. durch die Aktivierung bzw. Deaktivierung einzelner Informationen kann der Lastbalancierer dann die beste Informationskombination ermitteln. Für die

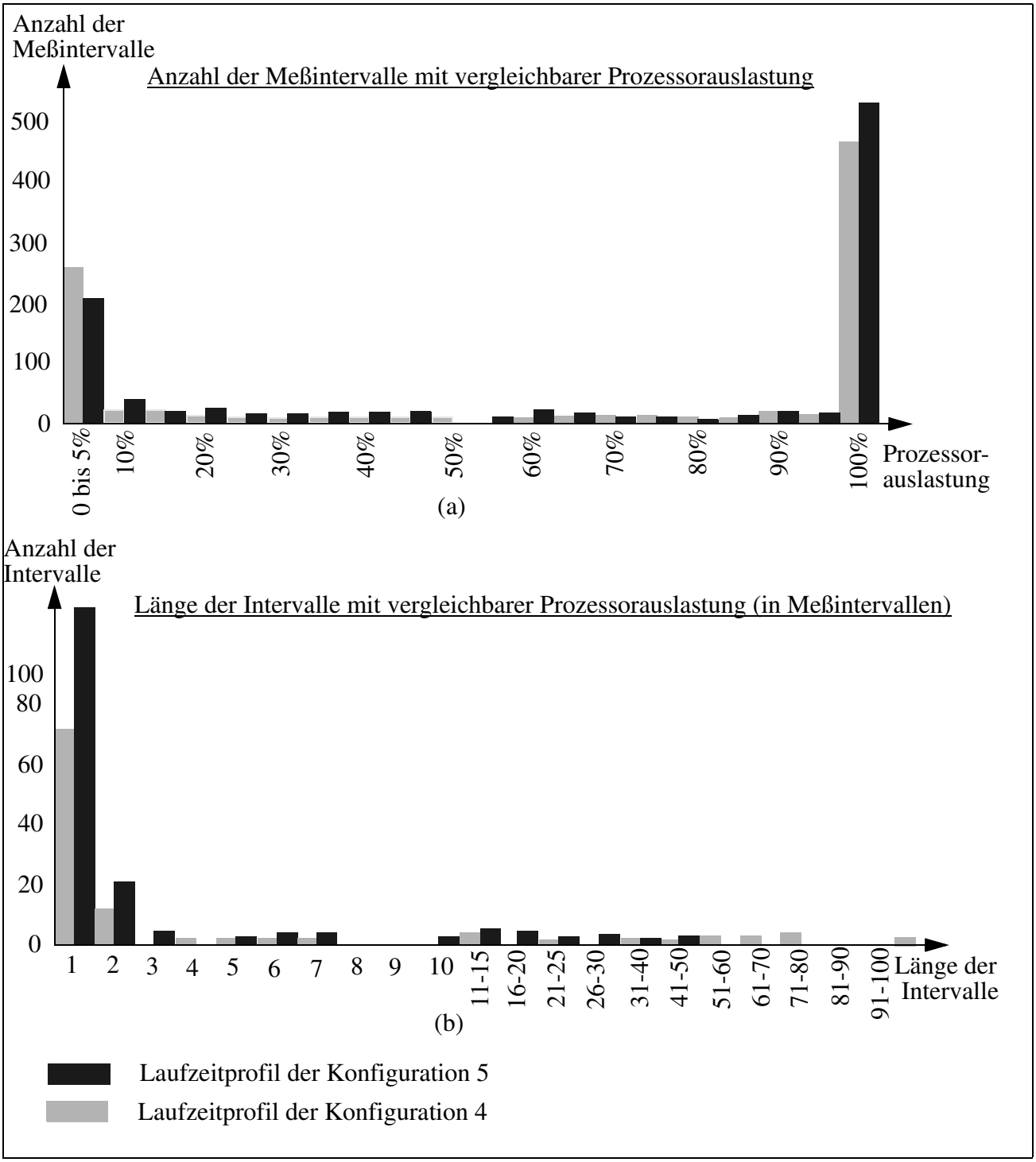


Abbildung 49: Anwendungen mit vergleichbarem Laufzeitprofil - Konfiguration 4 und 5.

Ermittlung können eine Vielzahl von Optimierungsverfahren z.B. Neuronale Netze [KGDK90][Zell94], Simulated Annealing [KiGV83], Mean Field Annealing [BuAy92], etc. verwendet werden.

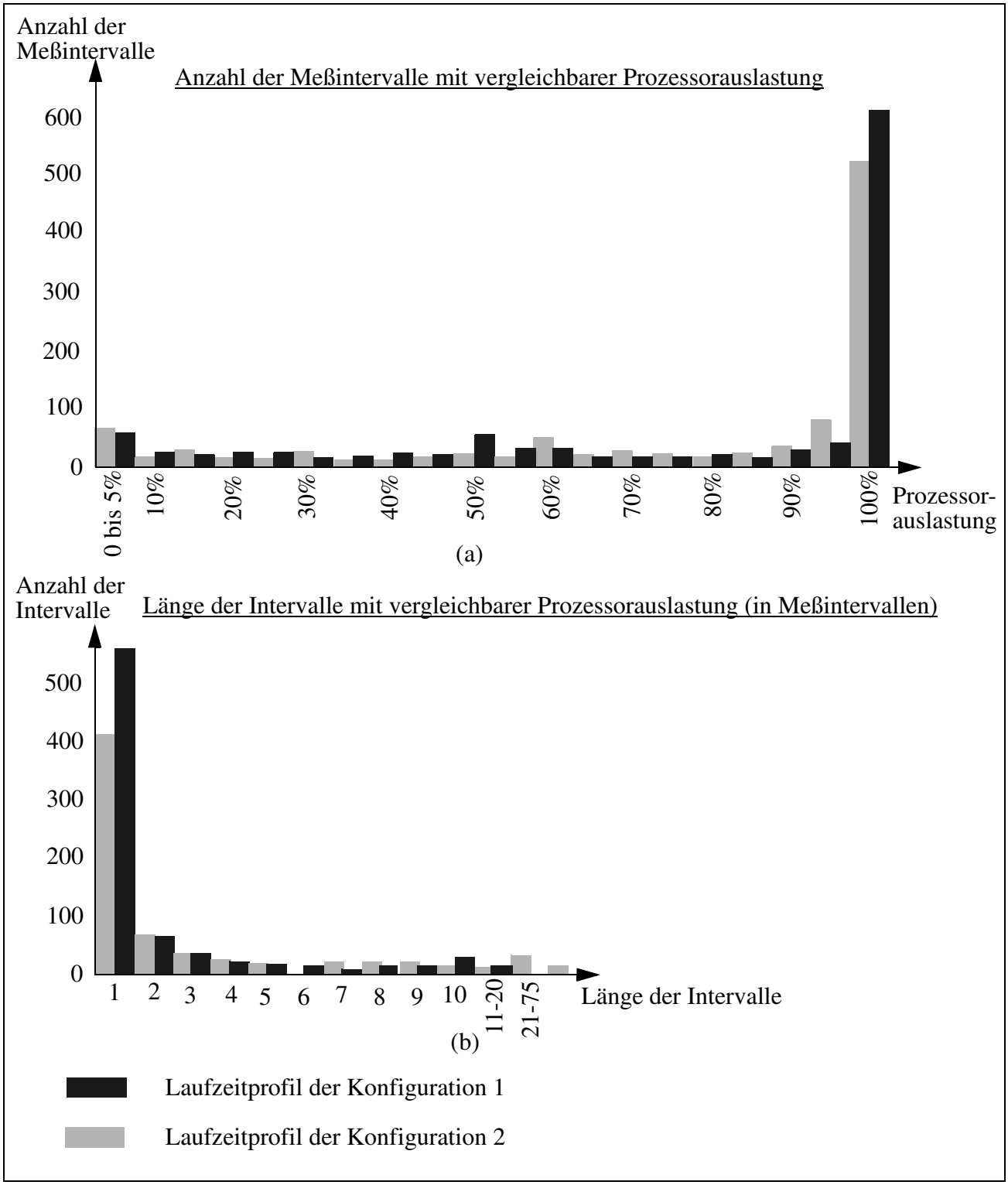


Abbildung 50: Anwendungen mit vergleichbarem Laufzeitprofil - Konfiguration 1 und 2.

8.2 Anmerkungen zu möglichen Weiterentwicklungen

In dieser Arbeit wurden die Auswirkungen verschiedener Informationsebenen auf die Effizienz der dynamischen Lastbalancierung untersucht. Dabei hat es sich herausgestellt, daß die Verwendung der höheren Informationsebenen, speziell der Ebenen 1 und 3, für die anwendungsspezifische Lastbalancierung sehr lohnenswert ist, weshalb ein betriebssystemintegrierter, dynamischer Lastbalancierer über entsprechende Informationsschnittstellen zu den Anwendungen verfügen sollte.
Im Zusammenhang mit der oben angeführten Untersuchung hat sich der Lastbalancierungsansatz der hierarchischen Lastbalancierungsumgebung PaLaBer für große, parallele und verteilte Systeme als äußerst geeignet erwiesen. Aus diesem Grund schließt diese Arbeit mit einem Überblick über die möglichen Weiterentwicklungen des PaLaBer-Systems.
Bei der Entwicklung des PaLaBer-Systems wurde auf eine modulare Realisierung der verschiedenen Protokolle geachtet, da nicht abgeschätzt werden konnte, welche der Protokolle sich als besonders zeitkritisch erweisen würden. Dieser modulare Entwurf bedingte eine offene Definition der Datenstrukturen, die dadurch nicht auf die jeweiligen Protokolle optimiert wurden. Im Hinblick auf eine Weiterentwicklung des PaLaBer-Systems müßten deshalb in einem ersten Schritt der Programmcode überarbeitet und die Protokolle sowie die dazugehörigen Datenstrukturen weiter optimiert werden.
Der zweite wichtige Schritt ist die Umstellung des Lasttransfermechanismus. Bislang basiert die preemptive Prozeßmigration auf der Funktionalität der NX-Bibliothek und somit auf der Basisfunktionalität des MACH-Mikrokernels. Damit die hierarchische Lastbalancierungsumgebung PaLaBer auch auf anderen Hardware-Plattformen verwendet werden kann, sollte z.B. ein Lasttransfermechanismus vergleichbar dem Condor-System verwendet werden, zumal dieser Ansatz im Rahmen verschiedener Arbeiten nicht nur optimiert, sondern auch auf unterschiedlichste Plattformen portiert wurde.
In engem Zusammenhang mit der Verfügbarmachung des PaLaBer-Systems auf verschiedenen Plattformen steht die Umstellung der Programmierschnittstelle für die Anwendungen von der NX-Bibliothek auf die MPI-Programmierschnittstelle. Diese Schnittstelle wird inzwischen von nahezu allen Hardware-Herstellern unterstützt, so daß zu erwarten ist, daß in Zukunft eine Vielzahl von parallelen Anwendungen diese Standardschnittstelle verwenden wird.
Der letzte und im Hinblick auf die in Kapitel 7 vorgestellte Untersuchung wichtigste Weiterentwicklung betrifft die Umsetzung der Informationsebenen. In dieser Arbeit wurden die höheren Informationsebenen durch 9 exemplarisch ausgewählte Informationen repräsentiert. Im Hinblick auf die Unterstützung von unterschiedlichsten Anwendungen sollte diese Informationsbasis weiter ausgebaut werden, so daß z.B. auch Abschätzungen über Ein-/Ausgabeaktivitäten, Hauptspeicherbedarfsänderungen etc. dem Lastbalancierer von den Anwendungen mitgeteilt werden können. Diese Informationen wurden in der vorliegenden Arbeit nicht verwendet, da sie für die beiden Versuchsanwendungen keine kritischen Größen darstellten.

Literaturverzeichnis

[AhGh91] I. Ahmad, A. Ghafoor, *Semi-Distributed Load Balancing For Massively Parallel Multicomputer Systems,* IEEE Transactions on Software Engineering, Vol. 17, No. 10, Oktober 1991, S. 987-1004

[AhGF94] I. Ahmad, A. Ghafoor, G. Fox, *Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers*, Journal of Parallel and Distributed Computing 20, 1994, S. 317-329

[AnHC90] F.D. Anger, J.J. Hwang, Y.C. Chow, *Scheduling with Sufficient Loosely Coupled Processors*, Journal of Parallel and Distributed Computing 9, 1990, S. 87-92

[ASLV95] P. Arbenz, C. Sprenger, H.P. Lüthi, S. Vogel, *SCIDDLE: A tool for large scale distributed computing*, Concurrency: Practice and Experience, Vol. 7(2), April 1995, S. 121-146

[ArFi89] Y. Artsy, R. Finkel, *Designing a Process Migration Facility - The Charlotte Experience*, IEEE Computer, Vol. 22 1/9, September 1989, S. 47-56

[BaSh85] A. Barak, A. Shiloh, *A Distributed Load-balancing Policy for a Multicomputer*, Software-Practice and Experience, Vol. 15, No. 9, September 1985, S. 901-913

[BaWh88] A. Barak, R. Wheeler, *MOSIX: An Integrated UNIX for Multiprocessor Workstations*, International Computer Science Institut, Berkeley, TR 88-004, Oktober 1988

[BaWa88] K.M. Baumgartner, B.W. Wah, *A Global Load Balancing Strategy for a Distributed Computer System*, Workshop on the Future Trends of Distributed Computing Systems in the 1990‘s, September 1988, S. 93-102

[Beck95] W. Becker, *Dynamische adaptive Lastbalancierung für große, heterogene konkurrierende Anwendungen*, Dissertation, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1995

[BePo94] W. Becker, R. Pollak, *Efficiency of Server Task Queueing for Dynamic Load Balancing*, Fakultätsbericht Nr. 1994/9, Universtät Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR), 1994

[BeWa94] W. Becker, G. Waldmann, *Exploiting Inter Task Dependencies for Dynamic Load Balancing*, Proc. of the Third IEEE Symposium on High Performance Distributed Computing, San Francisco, August 1994, S. 157-165

[BeDe94] K. Benmohammed-Mahieddine, P.M. Dew, *A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems*, Operating Systems Review Vol. 28, No, 1, ACM Press, Januar 1994, S. 66-76

[BeBo85] M.J. Berger, S.H. Bokhari, *A Partitioning Strategy for PDES Across Multiprocessors*, Proc. of the 1985 International Conference on Parallel Processing, August 1985, S. 166-170

[BeBo87] M.J. Berger, S.H. Bokhari, *A Partitioning Strategy for Nonuniform Problems on Multiprocessors*, IEEE Transactions on Computers, Vol. C-36, No. 5, Mai 1987, S. 570-580

[BeSS93] G. Bernardt, D. Steve, M. Simatic, *A survey of load balancing in networks of workstations*, Distributed Systems Engineering, Vol. 1, No. 2, Dezember 1993, S. 75-86

[BBDEGK94] R. Berrendorf, H.C. Burg, U. Detert, R. Esser, M. Gerndt, R. Knecht, *Intel Paragon XP/S - Architecture, Software Environment and Performance*, Forschungszentrum Jülich GmbH, Interner Bericht, KFA-ZAM-IB-9409, Mai 1994

[BeBD95] R. Berrendorf, H. Burg, U. Detert, *Leistungscharakteristika von Parallelrechnern: Fallstudie Intel Paragon*, Informationstechnik und Technische Informatik, it+ti 37 (1995), S. 37-45

[BlPa94] R.D. Blumofe, D.S. Park, *Scheduling Large-Scale Parallel Computations on Networks of Workstations*, Proc. of the Third IEEE Symposium on High Performance Distributed Computing, San Francisco, April 1994, S. 96-105

[Bogl92] Y.P. Boglaev, *Exact dynamic load balancing of MIMD architectures with linear programming algorithms*, Parallel Computing 18, 1992, S. 615-623

[Bokh81] S.H. Bokhari, *On the Mapping Problem*, IEEE Transactions on Computers, Vol. C-30, No. 3, März 1981, S. 207-214

[Bokh81/2] S.H. Bokhari, *A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System*, IEEE Transactions on Software Engineering, Vol. SE-7, No. 6, November 1981, S. 583-589

[Bokh88] S.H. Bokhari, *Partitioning Problems in Parallel, Pipelined, and Distributed Computing*, IEEE Transactions on Computers, Vol. 37, No. 1, Januar 1988, S. 48-57

[Bokh93] S.H. Bokhari, *A Network Flow Model for Load Balancing in Circuit-Switched Multicomputers*, IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 6, Juni 1993, S. 649-657

[BoMi88] S.W. Bollinger, S.F. Midkiff, *Processor and Link Assignment in Multicomputers using Simulated Annealing*, Proc. of the 1988 International Conference on Parallel Processing, August 1988, S. 1-7

[Bond93] A.M. Bond, *Adaptive Task Allocation in a Distributed Workstation Environment*, PhD Thesis, Victoria University of Wellington, 1993

[BoKu90] F. Bonomi, A. Kumar, *Adaptive Optimal Load Balancing in a Nonhomogenous Multiserver System with a Central Job Scheduler*, IEEE Transactions on Computers, Vol. 39, No. 10, Oktober 1990, S. 1232-1250

[BoNG88] N.S. Bowen, C.N. Nikolaou, A. Ghafoor, *Hierarchical Workload Allocation for Distributed Systems*, Proc. of the 1988 International Conference on Parallel Processing, August 1988, S. 102-109

[BKLL93] J. Boykin, D. Kirschen, A. Langerman, S. LoVerso, *Programming under Mach*, Addison-Wesley, 1993

[Bräu93] T. Bräunl, *Parallele Programmierung: Eine Einführung*, Vieweg, 1993.

[BuAy92] T. Bultan, C. Aykanat, *A New Mapping Heuristic Based on Mean Field Annealing*, Journal of Parallel and Distributed Computing 16, 1992, S. 292-305

[BuMa93] C. Burdorf, J. Marti, *Load Balancing Strategies for Time Warp on Multi-User Workstations*, The Computer Journal, Vol. 36, No. 2, 1993, S. 168-176

[CaKO94] J. Casas, R. Konuru, S.W. Otto, *Adaptive Load Migration Systems for PVM*, Proc. Supercomputing‘94, Washington, November 1994, S. 390-399

[CaKu88] T. Casavant, J. Kuhl, *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Engineering, Vol. 14, No. 2, Februar 1988, S. 141-154

[ChNa91] H.A. Choi, B. Narahari, *Algorithms for Mapping and Partitioning Chain Structured Parallel Computations*, Proc. of the 1991 International Conference on Parallel Processing, August 1991, S. I-625 - I-628

[CoNK93] A.N. Choudhary, B. Narahari, R. Krishnamurti, *An efficient heuristic scheme for dynamic remapping of parallel computations*, Parallel Computing 19, 1993, S. 621-632

[ChYL95] Y.-C. Chung, Y.-J. Yeh, J.-S. Liu, *A parallel dynamic load-balancing algorithm for solution-adaptive finite element meshes on 2D tori*, Concurrency: Practice and Experience, Vol. 7(7), Oktober 1995, S. 615-631

[Cybe89] G. Cybenko, *Dynamic Load Balancing for Distributed Memory Multiprocessors*, Journal of Parallel and Distributed Computing 7, 1989, S. 279-301

[DeIy89] M.V. Devarakonda, R.K. Iyer, *Predictability of Process Resource Usage: A Measurement-Based Study on UNIX*, IEEE Transactions on Software Engineering, Vol. 15, No. 12, Dezember 1989, S. 1579-1586

[DRCR92] K.S. DiBella, K. Ramamritham, P. Chrysanthis, S. Raghuram, *Scheduling Algorithms and their Performance on Shared Memory Multiprocessors*, Proc. of the ISMM International Conference Parallel and Distributed Computing Systems, Pittsburgh, PA, USA, Oktober 1992, S. 133-139

[DoEG96] E. Doncker, P. Ealy, A. Gupta, *Two Methods for Load Balanced Distributed Adaptive Integration*, HPCN Europe 1996, Springer-Verlag 1996, S.562-570

[DoOu91] F. Douglis, J. Ousterhout, *Transparent Process Migration: Design Alternatives and the Sprite Implementation*, Software-Practice and Experience , Vol. 21, No. 8, August 1991, S. 757-785

[DoHB95] A.B. Downey, M. Harchol-Balter, *A note on „The Limited Performance Benefits of Migrating Active Processes for Load Sharing*, Report No. UCB/CSD-95-888, November 1995, Computer Science Division (EECS), University of California Berkeley, California 94720

[EaLZ86] D.L. Eager, E.D. Lazowska, J. Zahorjan, *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, Performance Evaluation, Vol. 6, März 1986, S. 53-68

[EaLZ86/2] D.L. Eager, E.D. Lazowska, J. Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 12, No. 5, Mai 1986, S. 662-675

[EaLZ88] D.L. Eager, E.D. Lazowska, J. Zahorjan, *The Limited Performance Benefits of Migrating Active Processes for Load Sharing*, ACM SIGMETRICS, Performance Evaluation Review, Mai 1988, S. 63-72

[ErMS94] A. Erzmann, C. Müller-Schloer, *Zur Beurteilung dynamischer Lastausgleichsverfahren*, PARS Workshop, Potsdam, September 1994

[EHMS95] A. Erzmann, M. Hadeler, C. Müller-Schloer, *A Model for Efficient Programming of Dynamic Applications on Distributed Memory Multiprocessors*, EURO-PAR‘95, Stockholm, August 1995

[EsKn93] R. Esser, R. Knecht, *Intel Paragon XP/S - Architecture and Software Environment*, Informatik aktuell, H.W. Heuer, Supercomputer ‘93, Springer-Verlag, 1993, S. 121-141

[EvBu93] D.J. Evans, W.U.N. Butt, *Dynamic load balancing using task-transfer probabilities*, Parallel Computing 19, 1993, S. 897-916

[EvBu94] D.J. Evans, W.U.N. Butt, *Load balancing with network partitioning using host groups*, Parallel Computing 20, 1994, S. 325-345

[EzBP86] A.K. Ezzat, R.D. Bergeron, J.L. Pokoski, *Task Allocation Heuristics for Distributed Computing Systems*, The 6th International Conference on Distributed Computing Systems, 1986, S. 337-346

[FeRu90] D.G. Feitelson, L. Rudolph, *Mapping and Scheduling in a Shared Parallel Environment Using Distributed Hierarchical Control*, Proc. of the 1990 International Conference on Parallel Processing, August 1990, S. I-1 - I-8

[FeYN88] D. Ferguson, Y. Yemini, C. Nikolaou, *Microeconomic Algorithms for Load Balancing in Distributed Computer Systems*, The 8th International Conference on Distributed Computing Systems, Juni 1988, S. 491-499

[FeZh86] D. Ferrari, S. Zhou, *A Load Index for Dynamic Load Balancing*, Proc. Fall Joint Computer Conference, Dallas, Texas, 1986, S. 684-690

[Fost95] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995

[Gait90] J. Gait, *Scheduling and Process Migration in Partitioned Multiprocessors*, Journal of Parallel and Distributed Computing 8, 1990, S. 274-279

[GaGJ78] M.R. Garey, R.L. Graham, D.S. Johnson, *Performance Guarantees for Scheduling Algorithms*, Operations Research Vol. 26, No. 1, January/February 1978

[GaJo79] M.R. Garey, D.S. Johnson, *Computer and Intractability - A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979

[GaSh90] B. Gavish, O.R.L Sheng, *Dynamic File Migration in Distributed Computer Systems*, Communications of the ACM, Vol. 33, No. 2, Februar 1990, S. 177-189

[GHWZ94] U. Geuder, M. Härdtner, B. Wörner, R. Zink, *Scalable Execution Control of Grid-Based Scientific Applications on Parallel Systems*, Proc. 1994 Scalable High-Performance Computing Conference, Knoxville (TN), USA, 23-25 Mai 1994

[Gilo93] W.K. Giloi, *Rechnerarchitektur*, 2. Auflage, Springer-Verlag, Berlin Heidelberg New York, 1993

[Gosc91] A. Goscinski, *Distributed Operating Systems, The Logical Design*, Addison-Wesley, 1991

[GoDI93] K.K. Goswami, M. Devarakonda, R.K. Iyer, *Prediction-Based Dynamic Load Sharing Heuristics*, IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 6, Juni 1993, S. 638-648

[GALSTW93] T.P. Graf, R.G. Assini, J.M. Lewis, E.J. Sharpe, J.J. Turner, M.C. Ward, *HP Task Broker: A Tool for Distributing Computational Tasks*, Hewlett-Packard Journal, August 1993, S. 15-22

[GrNR90] D.C. Grunwald, B.A.A. Nazief, D.A. Reed, *Empirical Comparison of Heuristic Load Distribution in Point-to-Point Multicomputer Networks*, The Fifth Distributed Memory Computing Conference, Charleston, South Carolina, April 1990, S. 984-993

[GuGo90] R. Gupta, P. Gopinath, *A Hierarchical Approach to Load Balancing in Distributed Systems*, Proc. of the Fifth Distributed Memory Computing Conference, Charleston, South Carolina, April 1990, S. 1000-1005

[GuGo91] R. Gupta, P. Gopinath, *A Hybrid Approach to Load Balancing in Distributed Systems*, Proc. of the USENIX Symposium on Experiences with Distributed and Multiprocessor Systems, März 1991, S. 133-148

[HaAL95] B. Hamidzadeh, Y. Atif, D.J. Lilja, *Dynamic scheduling techniques for heterogeneous computing systems*, Concurrency: Practice and Experience, Vol. 7(7), Oktober 1995, S. 633-652

[He89] X. He, *Eine Übersicht über die Lastverteilung in verteilten Systemen*, Interner Bericht 190/89, Universität Kaiserslautern, Fachbereich Informatik, 1989

[Heis94] H.-U. Heiss, *Prozessorzuteilung in Parallelrechnern*, BI Wissenschaftsverlag, Reihe Informatik Band 98, Mannheim 1994

[Hert93] F. Hertweck, *A Comparison of Some Current Parallel Computer Architectures*, Informatik aktuell, H.W. Heuer, Supercomputer ‘93, Springer-Verlag, 1993, S. 104-120

[Hibb93] K. Hibbard, *OSF/1 AD Performance Projects*, Proc. of the Intel Supercomputer User‘s Group, 1993 Annual North America Users Conference, St. Louis, Missouri, Oktober 1993, S. 125-127

[Holl94] J.K. Hollingsworth, *Finding Bottlenecks in Large Scale Parallel Programs*, PhD at the University of Wisconsin-Madison, 1994

[HoUl88] J.E. Hopcroft, J.D. Ullman, *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*, Addison-Wesley, 1988

[HuSu93] K.A. Hua, J.X.W. Su, *Dynamic Load Balancing in Very Large Shared-Nothing Hypercube Database Computers*, IEEE Transactions on Computers, Vol. 42, No. 12, Dezember 1993, S. 1425-1439

[Hwan93] K. Hwang, *Advanced Computer Architecture, Parallelism, Scalability, Programmability*, McGray-Hill, 1993

[HwCh87] K. Hwang, R. Chowkwanyun, *Dynamic Load Balancing for Distributed Supercomputing and AI Applications*, Technical Report CRI-87-04, University of Southern California, Los Angeles, 1987

[HwXu90] K. Hwang, J. Xu, *Mapping Partitioned Program Modules onto Multicomputer Nodes Using Simulated Annealing*, Proc. of the 1990 International Conference on Parallel Processing, August 1990, S. II-292 - II-293

[IBM95] IBM Sales Manual (US) - Document ‘5765-145‘

[Inte92] *Intel Paragon Supercomputer, Specification Sheet*, Order No. 206, Intel Corporation Supercomputer Systems Division, 1992

[Inte93] *Intel Paragon XP/S Supercomputer, Specification Sheet*, Order No. 203/(Rev.2)/3-93/ 7.5K/GA, Intel Corporation Supercomputer Systems Division, 1993

[Inte94] *Intel Paragon User's Guide*, Order No. 312489-003, Intel Corporation Supercomputer System Division, June 1994

[Inte94/2] *Intel Paragon C System Calls Reference Manual*, Order No. 312487-003, Intel Corporation Supercomputer System Division, June 1994

[Kalé88] L.V. Kalé, *Comparing the Performance of two Dynamic Load Distribution Methods*, Proc. Parallel Processing, 1988, S. 8-12

[Kalé90] L.V. Kalé, *The Chare Kernel Parallel Programming Language and System*, Proc. of the 1990 International Conference on Parallel Processing, August 1990, S. II-17 - II-25

[KaKa90] Behzad Kamgar-Parsi, Behrooz Kamgar-Parsi, *On Problem Solving with Hopfield Neural Networks*, Biology Cybernetics, v62, 1990, Springer-Verlag, Berlin Heidelberg New York, S. 415-423

[KGDK90] B. Kamgar-Parsi, J.A. Gualtieri, J.E. Devaney, B. Kamgar-Parsi, *Clustering with Neural Networks*, Biology Cybernetics, v63, 1990, Springer-Verlag, Berlin Heidelberg New York, S. 201-208

[KaNa84] H. Kasahara, S. Narita, *Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing*, IEEE Transactions on Computers, Vol. C-33, No. 11, November 1984, S. 1023-1029

[KiGV83] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, *Optimization by Simulated Annealing*, Science, Vol. 220, No. 4598, Mai 1983, S. 671-680

[KoHa96] P. Kougiouris, G. Hamilton, *Buffer Management Issues in the Implementation of Fast Interprocess Communication*, Software-Practice and Experience, Vol. 26(2), Februar 1996, S. 195-211

[KrLi88] P. Krueger, M. Livny, *A Comparison of Preemptive and Non-Preemptive Load Distributing*, The 8th International Conference on Distributed Computing Systems, Juni 1988, S. 123-130

[Kübl94] A. Kübler, *Ortstransparente Kommunikation*, Studienarbeit Nr. 1369, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1994

[KuWa90] H. Kuchen, A. Wagner, *Comparison of Dynamic Load Balancing Strategies*, Bericht 5/ 90, Lehrstuhl für Informatik II, RWTH Aachen, 1990

[Kunz91] T. Kunz, *The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Transactions on Software Engineering, Vol. 17, No. 7, Juli 1991, S. 725-730

[LeOt86] W.E. Leland, T.K. Ott, *Load-balancing Heuristics and Process Behavior*, Proc. ACM SIGMETRICS Performance 1986, S. 54-69

[LeEl93] T. Lewis, H. El-Rewini, *Parallax: A Tool for Parallel Program Scheduling*, IEEE Parallel & Distributed Technology, Mai 1993, S. 62-72

[LiKe87] F.C.H. Lin, R.M. Keller, *The Gradient Model Load Balancing Method*, IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, Januar 1987, S. 32-38

[LiRa92] H.C. Lin, C.S. Raghavendra, *A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC)*, IEEE Transactions on Software Engineering, Vol. 18, No. 2, Februar 1992, S. 148-158

[LiCL90] K.J. Lin, L.Y. Chung, J.W.S. Liu, *Scheduling Real-Time Computations on Hypercubes with Load Balancing*, The Fifth Distributed Memory Conference, Charleston, South Carolina, April 1990, S. 975-983

[LiLM88] M. Litzkow, M. Livny, M. Mutka, *Condor - A Hunter for Idle Workstations*, Proceedings of the 8th. International Conference on Distributed Computing Systems, San Jose, California, Juni 1988.

[Lo88] V.M. Lo, *Heuristic Algorithms for Task Assignment in Distributed Systems*, IEEE Transactions on Computers, Vol. 37, No. 11, November 1988, S. 1384-1397

[LuLa95] C. Lu, S.M. Lau, *An adaptive algorithm for resolving processor thrashing in load distribution*, Concurrency: Practice and Experience, vol. 7(7), Oktober 1995, S. 653-670

[Ludw93] T. Ludwig, *Automatische Lastverwaltung für Parallelrechner*, BI Wissenschaftsverlag Reihe Informatik Band 94, Mannheim 1993

[LüMR91] R. Lüling, B. Monien, F. Ramme, *Load Balancing in Large Networks: A Comparative Study* , Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, 1991, S. 686-699

[McVZ93] C. McCann, R. Vaswani, J. Zahorjan, *A Dynamic Processor Allocation Policy for Multiprogrammed Shared Memory Multiprocessors*, ACM Transactions on Computer Systems, Vol. 11, No. 2, Mai 1993, S. 146-178

[MRRTT53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computer machines, J. Chem. Phys. 21, 1087, 1953, S. 1087-1092

[Meue95] H.-W. Meuer, *Supercomputer 1995, Anwendungen, Architekturen, Trends*, Fokus, Praxis Information und Kommunikation, Band 13, Saur, 1995

[Milo94] D.S. Milojicic, *Load Distribution, Implementation for the Mach Microkernel*, Vieweg Verlag, 1994

[MiTS89] R. Mirchandaney, D. Towsley, J.A. Stankovic, *Adaptive Load Sharing in Heterogeneous Systems*, The 9th International Conference on Distributed Computing Systems, Juni 1989, S. 298-306

[MuZa95] F.J. Muniz, E.J. Zaluska, *Parallel load-balancing: An extension to the gradient model*, Parallel Computing 21(1995), S. 287-301

[Mutk92] M.W. Mutka, *Estimating Capacity For Sharing in a Privately Owned Workstation Environment*, IEEE Transactions on Software Engineering, Vol. 18, No. 4, April 1992, S. 319-328

[NaDS92] A.K. Nanda, D. DeGroot, D.L. Stenger, *Scheduling Directed Task Graphs on Multiprocessors using Simulated Annealing*, The 12th International Conference on Distributed Computing Systems, Juni 1992, S. 20-27

[NiXG85] L.M. Ni, C.W. Xu, T.B. Gendreau, *Drafting Algorithm - A Dynamic Process Migration Protocol for Distributed Systems*, Proc. of the 5th International Conference on Distributed Computing Systems, Mai 1985, S. 539-546

[NiXG85/2] L.M. Ni, C.W. Xu, T.B. Gendreau, *A Distributed Drafting Algorithm for Load Balancing*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 10, Oktober 1985, S. 1153-1161

[NiMc93] L.M. Ni, P.K. McKinley, *A Survey of Wormhole Routing Techniques in Direct Networks*, IEEE Computer, Feb. 1993, S. 62-76

[NiSa88] D.M. Nicol, J.H. Saltz, *Dynamic Remapping of Parallel Computations with Varying Resource Demands*, IEEE Transactions on Computers, Vol. 37, No. 9, September 1988, S. 1073-1087

[NoTh93] M.G. Norman, P. Thanisch, *Models of Machines and Computation for Mapping in Multicomputers*, ACM Computing Surveys, Vol. 25, No. 3, September 1993, S, 263-302

[OSF92] *OSF/1 Operating System, User's Guide*, Open Software Foundation, Prentice Hall, 1992

[Osse92] W. Osser, *Automatic Process Selection for Load Balancing*, Master Thesis, University of California, Santa Cruz, Juni 1992

[PaHB96] D. J. Palermo, E. W. Hodges IV, P. Banerjee, *Dynamic Data Partitioning for Distributed-Memory Multicomputers*, Journal of Parallel and Distributed Computing 38, 1996, S. 158-175

[PaBH95] K. Pankhurst, P. Belli, R.J. Hynds, *A Project to Investigate the Running of Serial Batch Work in a Workstation Cluster Environment*, Studie of the Center of Computing Services, Imperial College of Science, Technology and Medicine, England 1995

[Pätz93] G. Pätzold, *Parallel Lattice-Boltzmann Simulation of Three-Dimensional Convective Flow*, Proc. of the Conference on Parallel Computational Fluid Dynamics, Paris, Frankreich, Mai 1993

[Pier94] P. Pierce, *The NX message passing interface*, Parallel Computing 20, 1994, S. 463-480

[PiRe95] P. Pierce, G. Regnier, *The Paragon Implementation of the NX Message Passing Interface*, Proc. of the Scalable High-Performance Computing Conference, Knoxville Tennessee, Mai 1995, S. 184-190

[Pleier96] C. Pleier, *Prozeßverlagerung in heterogenen Rechnernetzen basierend auf einer speziellen Übersetzungstechnik*, Herbert Utz Verlag Wissenschaft, 1996

[Poll95] R. Pollak, *A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer*, Proc. of the PDSC'95: International Symposium on Parallel and Distributed Supercomputing, Fukuoka, Japan, September 1995, S. 87-95

[PoReWa97] R. Pollak, A. Reuter, S. Wagner, *Some effects of various information levels on efficiency of dynamic load balancing*, Informatics, Cybernetics and Computer Science (ICCS-97). Collected Volume of Scientific Papers. Donetsk State Technical University. Donetsk, 1997. ISBN 5-7763-8557-1

[RTYGBB88] R. Rashid, A. Tevanian, jr., M. Young, D. Golub, R. Baron, D. Black, W.J. Bolosky, J. Chew, *Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures*, IEEE Transactions on Computers, Vol. 37, No. 8, August 1988, S. 896-907

[RaJe88] T.M. Ravi, D. Jefferson, *A Basic Protocol for Routing Messages to Migrating Processes*, Proc. on the 1988 International Conference on Parallel Processing, August 1988, S. 188-197

[Rich92] J.P. Richter, *Parallelisierung des Raytracing Verfahrens*, Studienarbeit im Fach Informatik, Lehrstuhl für Informatik VII, Universität Erlangen, 1992

[RoMc90] A. Ross, B. McMillin, *Experimental Comparison of Bidding and Drafting Load Sharing Protocols*, The Fifth Distributed Memory Computing Conference, South Carolina, April 1990, S. 968-974

[RuWa87] C.H. Russell, P.J. Waterman, *Variations on UNIX for Parallel-Processing Computers*, Communications of the ACM, Vol. 30, No. 12, Dezember 1987, S. 1048-1055

[SaEr87] P. Sadayappan, F. Ercal, *Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes*, IEEE Transactions on Computers, Vol. C-36, No. 12, Dezember 1987, S. 1408-1424

[Sale90] V.A. Saletore, *A Distributed and Adaptive Dynamic Load Balancing Scheme for Parallel Processing of Medium-Grain Tasks*, The Fifth Distributed Memory Computing Conference, South Carolina, April 1990, S. 994-999

[SaJP94] V.A. Saletore, J. Jacob, M. Padala, *Parallel Computations on the CHARM Heterogenous Workstation Cluster*, Proc. IEEE International Symposium on High Performance Distributed Computing, San Francisco, California, 1994, S. 203-210

[Scha92] J. Schabernack, *Lastausgleichsverfahren in verteilten Systemen - Überblick und Klassifikation*, Informationstechnik it 34(1992)5, S. 280-295

[ScWZ94] P. Scheuermann, G. Weikum, P. Zabback, *Data Partitioning and Load Balancing in Parallel Disk Systems*, Eidgenössische Technische Hochschule Zürich, Januar 1994

[ScSC88] T. Schwederski, H.J. Siegel, T.L. Casavant, *A Model of Task Migration in Partitionable Parallel Processing Systems*, Frontiers‘88: 2nd Symposium on the Frontiers of Massively Parallel Computations, Oktober 1988, S. 211-214

[SeGS95] C.N. Sekharan, V. Goel. R. Sridhar, *Load balancing methods for ray tracing and binary tree computing using PVM*, Parallel Computing 21 (1995), S. 1963-1978

[ShSi95] N.G. Shivaratri, M. Singhal, *A load index and transfer policy for global scheduling tasks with deadlines*, Concurrrency: Practice and Experience, Vol. 7(7), Oktober 1995, S. 671-688

[Simo92] K. Simon, *Effiziente Algorithmen für perfekte Graphen*, B.G. Teubner Stuttgart 1992

[SPJSM91] P.K. Sinha, K.S. Park, X. Jia, K. Shimizu, M. Maekawa, *Process Migration in the GALAXY Distributed Operating System*, Proc. of the Fifth International Parallel Processing Symposium, Anaheim, California, April 1991, S.611-618

[SrPa94] M. Srinivas, L.M. Patnaik, *Genetic Algorithms: A Survey*, IEEE Computer, Juni 1994, S. 17-26

[Stan84] J.A. Stankovic, *Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms*, Computer Networks 8, 1984, S. 199-217

[Stan85] J.A. Stankovic, *Stability and Distributed Scheduling Algorithms*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 10, Oktober 1985, S. 1141-1152

[StSi84] J.A. Stankovic, I.S. Sidhu, *An adaptive bidding algorithm for processes, clusters and distributed groups*, Forth International Conference on Distributed computing Systems, San Francisco, CA, Mai 1984, S. 49-59

[Sven90] A. Svensson, *History, an Intelligent Load Sharing Filter*, Proc. of the 10th International Conference on Distributed Computing Systems, Mai 1990, S. 546-553

[Tane87] A.S. Tanenbaum, *Operating Systems: design and implementation*, Prentice-Hall, 1987

[Tärn94] E. Tärnvik, *Dynamo - a portable tool for dynamic load balancing on distributed memory multicomputers*, Concurrency: Practice and Experience, Vol. 6(8), Dezember 1994, S. 613-639

[TeRa87] A. Tevanian, Jr., R.F. Rashid, *MACH: A Basis for Future UNIX Development*, CMU-CS-87-139, Computer Science Department, Carnegie Mellon University, Juni 1987

[TiWi84] A.M. Van Tilborg, L.D. Wittie, *Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers*, IEEE Transactions on Computers, Vol. C-33, No. 9, September 1984, S. 835-844

[Tril93] M. Triller, *Dynamischer Lastausgleich in verteilten Echtzeit-Systemen*, Informationstechnik und Technische Informatik 35 (1993)5, S. 35-41

[Trit90] S. Tritscher, *Dynamischer Lastausgleich auf dem IPSC/2 mittels Prozeßmigration*, Diplomarbeit, Technische Universität München, Januar 1990

[TrZB94] S. Tritscher, R. Zajcew, M. Barnett, *Load Leveling on the Paragon Multicomputer*, Proc. of the High-Performance Computing and Networking (HPCN), April 1994, S. 330-337

[Unge89] T. Ungerer, *Innovative Rechnerarchitekturen - Bestandsaufnahme, Trends, Möglichkeiten*, McGraw-Hill, 1989

[Wald95] K. Waldschmidt, *Parallelrechner, Architekturen-Systeme-Werkzeuge*, B.G. Teubner Stuttgart, 1995

[WaBe95] C. Walshaw, M. Berzins, *Dynamic load-balancing for PDE solvers on adaptive unstructured meshes*, Concurrency: Practice and Experience, Vol. 7(1), Februar 1995, S. 17-28

[Wu95] Min-You Wu, *Symmetrical Hopping: a scalable scheduling algorithm for irregular problems*, Concurrency: Practice and Experience, Vol. 7(7), Oktober 1995, S. 689-706

[XuHw93] J. Xu, K. Hwang, *Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer*, Journal of Parallel and Distributed Computing 18, 1993, S. 1-13

[XMLL95] C. Xu, B. Monien, R. Lüling, F.M. Lau, *Nearest-neighbor algorithms for load balancing in parallel computers*, Concurrency: Practice and Experience, Vol. 7(7), Oktober 1995, S. 707-736

[YuLL91] P.S. Yu, A. Leff, Y.H. Lee, *On Robust Transaction Routing and Load Sharing*, ACM Transactions on Database Systems, Vol. 16, No. 3, 1991

[Zell94] A. Zell, *Simulation Neuronaler Netze*, Addison-Wesley, 1994

[ZhFe87] S. Zhou, D. Ferrari, *An Experimental Study of Load Balancing Performance*, Report No. 87/336, Computer Science Division, University of California, Berkeley, 1987

[Zhou88] S. Zhou, *A Trace-Driven Simulation Study of Dynamic Load Balancing*, IEEE Transactions on Software Engineering, Vol. 14, No. 9, September 1988, S. 1327-1341

[ZZWD93] S. Zhou, X. Zheng, J. Wang, P. Delisle, *Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems*, Software-Practice and Experience, Vol. 23(12), Dezember 1993, S. 1305-1336

[ZiKr93] C. Zimmermann, A.W. Kraas, *Mach, Konzepte und Programmierung*, Springer-Verlag, Berlin Heidelberg New York, 1993

[Zoma95] A.Y.H. Zomaya, *Parallel & Distributed Computing Handbook*, McGraw-Hill, 1995

Anhang A

Definition eines Lastbalancierungsbaumes

Ziel dieses Anhanges ist, es eine formale Definition einer gültigen Konfiguration eines Lastbalancierungsbaumes im Sinne des PaLaBer-Systems anzugeben. Zur Definition eines Lastbalancierungsbaumes wird aus naheliegenen Gründen ein graphentheoretischer Ansatz gewählt. Die einführenden Definitionen orientieren sich dabei an [Simo92].

Definition 9: Ein **(ungerichteter) Graph** G =(V,E) besteht aus einer endlichen Menge V und einer zweistelligen Relation $E \subseteq V \times V$. Die Elemente von V heißen Knoten des Graphen G und die Elemente von E heißen Kanten des Graphen G.

Definition 10: Sei G = (V,E) ein Graph und sei $x \in V$ ein Knoten. Dann heißt $N(x) = \{y_i \in V \mid (x,y_i) \in E\}$ die **Nachfolgermenge** von Knoten x.

Definition 11: Sei G = (V,E) ein Graph und $x \in V$ ein Knoten. Dann heißt $VZG(x) = \left|\{x \in V \mid ((x,y) \in E)\}\right|$ **Verzweigungsgrad** des Knotens v.

Definition 12: Sei G = (V,E) ein Graph und $v_i \in V$ für $i = 0,...,s$. Dann wird die endliche Folge $\text{Pfad}(v_0,v_{s-1}) = v_0,...,v_{s-1}$ mit $(v_i,v_{i+1}) \in E$ als **Pfad** von v_o nach v_{s-1} bezeichnet. Dabei bezeichnet $s = \left|\text{Pfad}(v_0,v_{s-1})\right|$ die **Länge** des Pfades.

Definition 13: Ein Graph G = (V,E) heißt genau dann **zusammenhängend**, wenn für jedes Paar $x, y \in V$ ein Pfad in G existiert, der die Knoten x und y verbindet.

Definition 14: Ein Graph G = (V,E) heißt genau dann **azyklisch**, wenn es keinen Pfad in G gibt mit $v_0 = v_{s-1}$ und $s \geq 1$.

Definition 15: Ein **Baum** ist ein zusammenhängender, ungerichteter und azyklischer Graph G = (V,E).

Definition 16: Ein **Lastbalancierungsbaum** im Sinne des PaLaBer-Systems ist ein zusammenhängender, ungerichteter und azyklischer Graph G(V,E) der die folgenden sieben Bedingungen erfüllt:

(1) $V = V_W \cup V_I \cup V_B$

und

(2) $V_W \cap V_I = V_W \cap V_B = V_I \cap V_B = \phi$

und

(3) $|V_W| = 1 \, , |V_B| \geq |V_I| \geq 1$

und

(4) $VZG(x \in V_W) \geq 1 \quad , \quad VZG(y \in V_I) \geq 2 \quad , \quad VZG(z \in V_B) = 1$

und

(5) $\forall((x,y) \in E \wedge x \in V_W) \Rightarrow y \in V_I$

und

(6) $\forall((x,y) \in E \wedge x \in V_B) \Rightarrow y \in V_I$

und

(7) $\forall(x \in V_I)$ gilt

$\exists(((x,y_i) \in E) \text{mit } (y_i \in V_B)) \Rightarrow (|N(x) \cap V_I| = 1) \oplus (|N(x) \cap V_W| = 1)$ [1].

Anmerkung zur Definition eines Lastbalancierungsbaumes:

- Die Bedingungen (1) - (3) stellen sicher, daß es genau einen Wurzelknoten, mindestens einen inneren Knoten und mindestens eine Blattkomponente gibt. V_W ist dabei die einelementige Wurzelmenge, V_I ist die Menge der inneren Komponenten und V_B ist die Menge der Blattkomponenten.
- Die Bedingung (5) stellt sicher, daß der Wurzelknoten nur innere Knoten als direkte Nachbarknoten hat.
- Die Bedingungen (4) und (6) stellen sicher, daß eine Blattknoten nur einen Nachbarknoten, einen inneren Knoten, hat.
- Die Bedingung (7) stellt sicher, daß ein innerer Knoten entweder nur Blattknoten oder nur innere Knoten als Nachbarknoten hat.

1. $\oplus$ steht dabei für EXKLUSIVE ODER.

Anhang B

PaLaBer - Referenzhandbuch

Ziel dieses Anhanges ist es, eine kurze Beschreibung der Routinen der Anwendungsbibliothek PaLib zu geben. Für eine ausführliche Beschreibung der Routinen und ihrer Aufrufparameter wird auf das *Paragon C System Calls Reference Manual* [Inte94/2] verwiesen.

Kommunikationsroutinen

Für die Kommunikation von Anwendungsprozessen innerhalb einer parallelen Anwendung stehen die folgenden acht Routinen zur Verfügung.

long pa_cprobe(long typ); Synchrones Warten auf eine Nachricht vom Typ *typ*.
void pa_crecv(long type, char *buffer, long count); Synchrones Empfangen einer Nachricht.
void pa_csend(long type, char *buffer, long count, long v_node, long v_ptype); Synchrones Senden einer Nachricht.
long pa_infocount(void); Länge der zuletzt empfangenen Nachricht.
long pa_infonode(void); Knoten des Senders der zuletzt empfangenen Nachricht.
long pa_infoptype(void); Ptype des Senders der zuletzt empfangenen Nachricht.
long pa_infotype(void); Typ der zuletzt empfangenen Nachricht.
long pa_iprobe(long typesel); Nicht-blockierender Test, ob eine Nachricht zum Empfang ansteht.

Prozeßverwaltungsroutinen

Zur Verwaltung der Prozesse einer parallelen Anwendung stehen die folgenden Routinen zur Verfügung. Die Routinen pa_application_exit(), pa_exit() und palib_init() sind dabei nicht Bestandteil der NX-Bibliothek von Intel, sondern dienen zum An- bzw. Abmelden der Anwendungsprozesse bei der dynamischen Lastbalancierungsumgebung PaLaBer.

void pa_application_exit(long r_value); Terminierungsnachricht einer Anwendung an den Lastbalancierer.
void pa_exit(long r_value); Terminierungsnachricht eines Anwendungsprozesses an den Lastbalancierer.
void palib_init(int *argc, char *argv[]); Initialisierung eines Anwendungsprozesses.
long pa_load(long node_list[],long numnodes, long ptype, long pid_list[],char *pathname); Starten eines oder mehrerer Anwendungsprozesse zur Laufzeit.
long pa_numnodes(void); Anzahl der Knoten innerhalb der virtuellen Partition.
long pa_mynode(void); Knotennummer des Anwendungsprozesses innerhalb der virtuellen Partition.
long pa_myptype(void); Prozeßtyp des Anwendungprozesses.

Informationsroutinen

Zur Umsetzung der Informationsebenen im PaLaBer-System stehen dem Anwender die folgenden Routinen zur Verfügung.

StaticCommunicationPartner(long NumberOfPartners, long ListOfPartners[]); Information der Ebene 1 zur Übermittlung der statischen Kommunikationspartner eines Anwendungsprozesses.
CommunicatonPhase(void); Information der Ebene 2, die dem Lastbalancierer mitteilt, daß der Anwendungsprozeß eine kommunikationsintensive Verarbeitungsphase betritt.
IOPhase(void); Information der Ebene 2, die dem Lastbalancierer mitteilt, daß der Anwendungsprozeß nun eine Phase mit hoher Ein-/Ausgabetätigkeit betritt.
ComputationPhase(void); Information der Ebene 2, die dem Lastbalancierer mitteilt, daß der Anwendungsprozeß nun eine rechenintinsive Bearbeitungsphase betritt.

AdvanceTerminationNotice(void);	
	Information der Ebene 3, die dem Lastbalancierer mitteilt, daß der Anwendungsprozeß in nächster Zeit terminieren wird.
StartRunTimeMeasurement(void)	
	Information der Ebene 3 zur Erfassen der Restlaufzeit eines Anwendungsprozesses.
EndRunTimeMeasurement(void)	
	Information der Ebene 3 zur Erfassen der Restlaufzeit eines Anwendungsprozesses.
RemainingRunTime(long Quantum);	
	Information der Ebene 3 zur Erfassen der Restlaufzeit eines Anwendungsprozesses.
StartBehaviorMeasurement(double EstimatedCpuBehavior);	
	Information der Ebene 3 zur Bestimmung des durchschnittlichen Prozessorbedarfs eines Anwendungsprozesses.
EndBehaviorMeasurement(void);	
	Information der Ebene 3 zur Bestimmung des durchschnittlichen Prozessorbedarfs eines Anwendungsprozesses.
UnknownBehavior(void);	
	Information der Ebene 3, die dem Lastbalancierer mitteilt, daß der Anwendungsprozeß nun keine Aussage machen kann über sein durchschnittliches Lastverhalten.

Anhang C

Ortstransparente Kommunikation

Ortstransparente Kommunikation liegt dann vor, wenn der Sender einer Nachricht nicht die aktuelle Lokation des Empfängers kennen muß [RaJe88]. Stattdessen erfolgt die Adressierung mit Hilfe einer systemweit eindeutigen, virtuellen Adresse, die zur Laufzeit vom Kommunikationssystem auf die reale Adresse abgebildet wird. Die Einführung der Ortstransparenz ist unerläßlich für die anwendungstransparente Prozeßmigration durch einen dynamischen Lastbalancierer. Sie erleichtert jedoch auch die Eingliederung von Mechanismen zur Softwarefehlertoleranz in parallelen und verteilten Systemen. Ebenso stellt sie einen erheblichen Programmierkomfort dar, da der Anwendungsprogrammierer von Detailkenntnissen über die Hardware entlastet wird.
An die Ortstransparenz werden speziell im Zusammenhang mit der dynamischen Lastbalancierung die folgenden Bedingungen gestellt:

- Der Systemaufwand pro Anwendungsnachricht sollte möglichst gering sein.
- Der Sender einer Nachricht sollte durch die Migration des Empfängers nicht beeinträchtigt werden.
- Das Nachrichtensystem sollte ordnungserhaltend sein.

„Ordnungserhaltend“ bedeutet dabei, daß Nachrichten, die ein Prozeß A an einen Prozeß B sendet, in derselben Reihenfolge bei Prozeß B ankommen wie sie der Prozeß A abgeschickt hat. Um die Realisierung der ortstransparenten Kommunikation in der hierarchischen Lastbalancierungsumgebung PaLaBer besser einordnen zu können, werden in diesem Anhang verschiedene technische Ansätze zur Realisierung der ortstransparenten Kommunikation vorgestellt.

C.1 Technische Ansätze zur Realisierung der Ortstransparenz

Zur Realisierung der ortstransparenten Kommunikation gibt es prinzipiell zwei Möglichkeiten:

1. Jede Anwendungsnachricht wird grundsätzlich über den Kommunikationsmanager geleitet, der die aktuelle Lokation des Empfängers ermittelt und die Nachricht an diesen weiterleitet. Diese Realisierungsvariante wird nachfolgend als **indirekte Kommunikation** bezeichnet.
2. Die Anwendungsprozesse kommunizieren direkt miteinander, der Kommunikationsmanager muß in diesem Fall jedoch sicherstellen, daß während einer Migration keine Nachrichten an den migrierenden Prozeß geschickt werden. Diese Realisierungsvariante wird nachfolgend als **direkte Kommunikation** bezeichnet.

Beide Ansätze finden in der Praxis Anwendung. So wird im ENX-System [Ludw93] ein indirektes Kommunikationsverfahren und im MPVM-System [CaKO94] ein direktes Kommunikationsverfahren verwendet, weshalb beide Realisierungsmöglichkeiten nachfolgend kurz diskutiert werden.
Bei der indirekten Kommunikation (siehe Abbildung 51) werden für eine Anwendungsnachricht mindestens zwei Systemnachrichten benötigt. Der Vorteil dieser Realisierung liegt darin, daß das Senden einer Nachricht immer möglich ist. Der Kommunikationsmanager kann für den Fall, daß sich der Empfänger der Nachricht momentan in einem Migrationsvorgang befindet, die Anwendungsnachricht zwischenspeichern und nach erfolgter Migration dem Empfänger zustellen.
Der Kommunikationsmanager muß dazu zwei Dienste bereitstellen:

1. Adreßverwaltung
2. Weiterleitung der Anwendungsnachrichten

Im einfachsten Fall werden beide Dienste von einer zentralen Instanz realisiert [Kübl94]. Dabei schicken die Anwendungsprozesse alle Nachrichten an den Kommunikationsmanager. Dieser ermittelt mit Hilfe einer globalen Adreßtabelle die aktuelle Lokation des Empfängers und leitet die Nachricht an diesen weiter. Bevor ein Anwendungsprozeß migriert, benachrichtigt er den Kommunikationsmanager, der daraufhin sämtliche Nachrichten für diesen Prozeß zwischenspeichert und erst nach erfolgter Migration diese an die neue Adresse weiterleitet.
Im Vergleich dazu verwendet das ENX - System ([Ludw93]) für die Adreßverwaltung einen zentralen Administrationsprozeß, für die Nachrichten-Weiterleitung hingegen sind lokale Komponenten auf den Knoten verantwortlich. Um den Aufwand für die Adreßumsetzung zu verringern, verfügt jede lokale Komponente über einen Adreß-Cache. Nur wenn die Lokation eines Prozesses nicht aus der lokalen Adreßtabelle ermittelt werden kann, wird die Lokation mit Hilfe des Administrationsprozesses bestimmt.
Da für die Umsetzung der Ortstransparenz in großen parallelen und verteilten Systemen zentrale Instanzen zwangsläufig zum Engpaß werden, wurden in [Kübl94] und [SPJSM91] völlig verteilte Ansätze untersucht, bei denen jede Komponente nur über einen Teil der Adreßtabelle verfügt. Findet eine Komponente den Empfänger einer Nachricht nicht in ihrer lokalen Adreßtabelle, wird ein entsprechendes Suchprotokoll angestoßen. Die Effizienz dieser verteilten Ansätze hängt dabei entscheident von zwei Faktoren ab:

1. Wie häufig werden benötigte Adressen in den lokalen Tabellen gefunden?
2. Wie hoch ist der Aufwand für das Suchprotokoll?

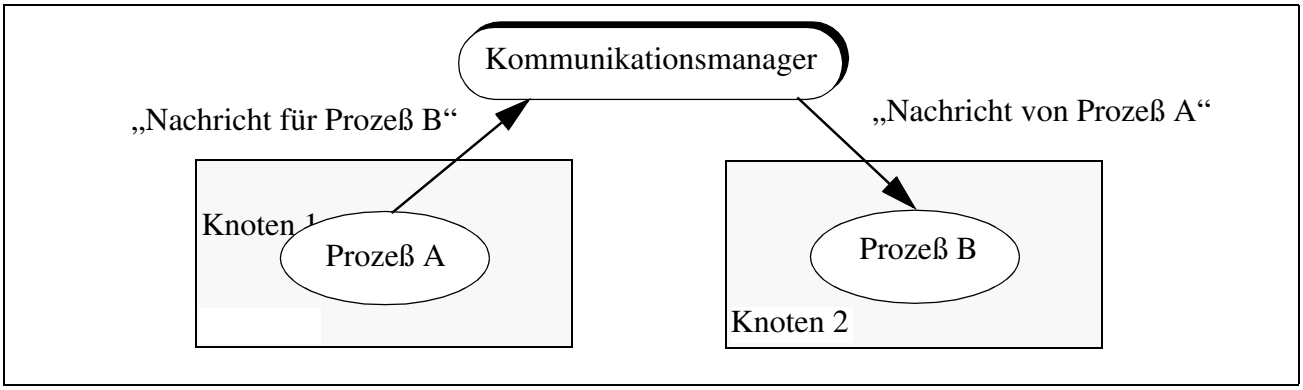


Abbildung 51: Indirekte Kommunikation.

Da der Realisierungsaufwand für die indirekte Kommunikation sehr hoch ist, wird für kleinere Systeme, bei denen die Vorteile dieser Kommunikationsart nicht voll zum Tragen kommen, häufig die direkte Kommunikation verwendet. Dabei wird die Kommunikation direkt zwischen den Anwendungsprozessen abgewickelt. Der Systemaufwand pro Anwendungsnachricht ist bei diesem Ansatz minimal, da nur bei der ersten Nachricht an einen Prozeß die Hilfe des Kommunikationsmanagers zur Ermittlung der aktuellen Lokation benötigt wird. Allerdings muß der Kommunikationsmanager bei diesem Ansatz sicherstellen, daß während der Migration eines Prozesses keine Nachricht mehr an diesen geschickt wird. Im MPVM - System [CaKO94] wird dies dadurch erreicht, daß der Kommunikationsmanager vor der Migration alle bisherigen Kommunikationspartner des Migranden darüber informiert, daß sie an diesen Prozeß jetzt keine Nachrichten schicken können. Erst nachdem alle benachrichtigten Prozesse eine Empfangsbestätigung an den Kommunikationsmanager geschickt haben, wird der Prozeß migriert. Nach erfolgreicher Migration werden wiederum alle Kommunikationspartner darüber informiert, daß der migrierte Prozeß nun wieder Nachrichten empfangen kann. Ein solches Protokoll setzt gewisse Bedingungen an das Anwendungsverhalten bzw an die Systemdynamik. Haben die Anwendungsprozesse viele Kommunikationspartner bzw. ist die Systemdynamik und somit die Anzahl der Lastverschiebungen hoch , kann der Ansatz der direkten Kommunikation nicht verwendet werden.
Ausgehend von den Vor- und Nachteilen der beiden Ansätze wurde für das PaLaBer-System ein indirekter Kommunikationsansatz gewählt.