

Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen

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

Vorgelegt von Wolfgang Becker aus Hoengen

Hauptberichter	Prof. Dr. Dr. A. Reuter
Mitberichter	Prof. Dr. K. Rothermel
Tag der mündlichen Prüfung	7. Dezember 1995

Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR)
der Universität Stuttgart
1995

CR-Klassifikation C.2.4, D.4.1, D.4.8

1	Kurzfassung	5
2	Grundkonzepte der Lastbalancierung	6
2.1	Parallele Rechnersysteme	6
2.2	Lastverteilung im Mehrbenutzerbetrieb	7
2.3	Lastverteilung in parallelen Anwendungen	8
2.4	Ziele und Größen in der Lastbalancierung	8
2.4.1	Motivation des Lastbegriffs	8
2.4.2	Zweck und Potential der Lastbalancierung	10
2.4.3	Rechenknoten als Lastempfänger	11
2.4.4	Aufträge als Lasterzeuger	13
2.4.5	Netzwerk und Kommunikation	14
2.4.6	Daten als besondere Ressourcen	15
2.4.7	Weitere Elemente in der Lastbalancierung	15
2.4.8	Das Modell zur Bearbeitung von Aufträgen	16
2.5	Aufgaben und Struktur der Lastbalancierung	18
2.5.1	Aufgaben der Lastbalancierung	18
2.5.2	Komponenten in der Lastbalancierung	19
2.5.3	Struktur der Lastbalancierung	20
2.5.4	Statische Ablaufplanung	26
2.5.5	Dynamische Ablaufregelung	34
2.5.6	Adaptive Ablaufregelung	37
2.5.7	Optimierungskriterien für dynamische Lastbalancierung	38
3	Das HiCon Konzept zur dynamischen Lastbalancierung	43
3.1	Motivation	43
3.2	Klassifikation des Ansatzes	46
3.3	Kurzvorstellung des Gesamtsystems	49
3.4	Verarbeitungsmodell für Anwendungen	50
3.5	Das Betriebssystem zur Verwaltung der Anwendungsläufe	55
3.6	Ein Netzwerk kooperierender Lastbalancierungsagenten	58
3.7	Aufbau und Ablauf der Lastbalancierung	59
3.7.1	Informationssammlung	61
3.7.2	Entscheidung: Bewertung und Zuweisung	70
3.7.3	Entscheidung: Austausch von Anwendungen zwischen Clustern	75
3.7.4	Dynamische Einplanung von Auftragsgruppen	77
3.7.5	Dynamische Adaption	80
4	Vergleich mit relevanten Forschungsarbeiten	85
4.1	Zentrale adaptive Transaktionsplatzierung im Datenbankbereich	85
4.2	Lastbalancierung in Workstation-Netzen	86
4.3	Dezentrale Lastbalancierung für Parallelrechner	88
4.4	Dynamische Lastbalancierung unter Verwendung von Vorabinformationen	89
4.5	Komplexe zentrale dynamische Lastbalancierung	89
5	Leistungsbewertung des Ansatzes	91
5.1	Die prototypische Lastbalancierungsumgebung	91
5.2	Untersuchte Anwendungstypen	94
5.2.1	Parallele Wegesuche in gerichteten Graphen	94
5.2.2	Parallele Flächenerkennung in Punktrasterbildern	96
5.2.3	Parallele Verarbeitung komplexer relationaler Anfragen	98
5.2.4	Datenbankoperationen auf geometrischen Objekten	100
5.2.5	Parallele Spannungsberechnung nach der Methode der finiten Elemente	101
5.3	Leistungssteigerung durch dynamische Lastbalancierung	105

5.4	Zusatzaufwand durch dynamische Lastbalancierung	108
5.4.1	Rechenaufwand für Lastbalancierungsentscheidungen	109
5.4.2	Verzögerung durch Entscheidungsfindung	110
5.4.3	Kommunikationslast durch Informations- und Auftragsaustausch	110
5.5	Skalierbarkeit der Lastbalancierung	111
5.6	Flexibilität der Lastbalancierung	114
5.6.1	Berücksichtigung von Auftragsabhängigkeiten	115
5.6.2	Automatische Anpassung von Entscheidungsparametern	116
6	Zusammenfassung der Ergebnisse	123
6.1	Zusammenfassung	123
6.2	Ausblick	124
7	Literaturverzeichnis	127

Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen

Selbst ein beschränkter Geist, der sich einem einzigen wissenschaftlichen Arbeitsgebiet widmet, muß darin unweigerlich zu großen Fortschritten gelangen. - Mary Shelley: Frankenstein.

1 Kurzfassung

In dieser Arbeit wird ein Konzept entwickelt, das eine automatische Verteilung der Rechenlast auf parallelen Rechnersystemen ermöglicht. Durch die strukturelle und algorithmische Flexibilität des entwickelten Lastbalancierungskonzeptes kann für einen großen Bereich von Anwendungen auf verschiedenen Systemen durch dynamische Planung und Regelung der Gesamtdurchsatz deutlich gesteigert werden.

Das *HiCon*-Lastbalancierungskonzept weist gegenüber bisher bekannten Ansätzen vier Eigenschaften auf, die eine größere Flexibilität und ein erhöhtes Optimierungspotential bewirken. Die Aufgabe der Lastbalancierung kann den Systemcharakteristiken angepaßt in einer gemischt zentralen und verteilten Kooperationsstruktur konfiguriert werden, um die Vorteile zentraler Strukturen bei uneingeschränkter Skalierbarkeit zu erhalten. Durch Messung des System- und Anwendungsverhaltens zur Laufzeit und Ausnutzung von Vorabschätzungen seitens der Anwendungen zur Laufzeit kann der Lastbalancierungsalgorithmus sowohl planend als auch reagierend wirken. Mit Hilfe einiger Regelparameter kann die Lastbalancierung ihren Entscheidungsalgorithmus dynamisch auf das tatsächliche System- und Anwendungsverhalten abstimmen. Die Lastbalancierung berücksichtigt zur Durchsatzoptimierung neben der Ressourcenauslastung auch Datenkommunikation.

Die Untersuchungen basieren auf einem Client - Server strukturierten Ablaufmodell mit Kooperation auf gemeinsamen Daten. Das entwickelte Konzept wurde durch eine prototypische Laufzeitumgebung auf Workstation-Netzen validiert, auf der verschiedenartige parallele Anwendungen einzeln und konkurrierend unter Einwirkung der Lastbalancierung beobachtet wurden. Im *HiCon*-Konzept wurden neue Ansätze entwickelt und einige allgemein interessante Ergebnisse gewonnen (Abschnitt 6.1).

2 Grundkonzepte der Lastbalancierung

In diesem Kapitel wird die Problematik und Notwendigkeit automatischer Lastbalancierung erläutert, die Aufgaben und Ziele der Lastbalancierung ausgearbeitet und die wichtigsten Grundkonzepte vorgestellt. Einzelne Facetten der Begriffsbildung und Klassifikation lehnen sich an die vorhandene Literatur an; eine Übersicht in ähnlicher Klarheit und Allgemeinheit existiert jedoch bislang nicht. Die Begriffsbildung ist, ohne die Allgemeinheit einzuschränken, auf das Konzept dieser Arbeit zugeschnitten. In den weiteren Kapiteln wird das in dieser Arbeit entwickelte Konzept vorgestellt und untersucht.

2.1 Parallele Rechnersysteme

Die Rechnerarchitekturen und Strukturen, die sich derzeit zur Berechnung komplexer Anwendungen und zum Arbeiten auf gemeinsamen Ressourcen durchsetzen, sind lose gekoppelte Rechnersysteme mit hoher Parallelität und große heterogene Workstation-Netze. Darunter werden zunehmend anstelle von Einprozessor-Rechenknoten Mehrprozessorsysteme mit gemeinsamem Haupt- und Sekundärspeicher und mäßiger Parallelität eingesetzt. Der Grund liegt vor allem darin, daß die Leistung einzelner Prozessoren nicht grenzenlos weiter gesteigert werden kann, während es weniger problematisch ist, sehr viele Prozessoren zu einem geeigneten Netzwerk zusammen zu fügen und so die akkumulierte Rechenleistung beliebig zu steigern. Große Zahlen von Standardkomponenten sind durch Massenproduktion sehr preisgünstig. Systeme mit gemeinsamen Ressourcen wie Hauptspeicher oder Sekundärspeicher sind Softwareseitig leichter zu handhaben, jedoch nicht beliebig skalierbar, weil die Synchronisation und die Nutzung zentraler Bussysteme und Controller zum Engpaß werden. Im Rahmen dieser Arbeit soll auf Vektor- und Pipeline-Strukturen sowie SIMD-Parallelrechner nicht weiter eingegangen werden, da keine Lastbalancierung auf dieser Ebene betrachtet wird. Weiterhin sollen Architekturaspekte in Bezug auf Hardware- und Software-seitige Fehlertoleranz nicht näher beleuchtet werden.

Die seither verwendeten Großrechner mit nur einem sehr leistungsfähigen Prozessor sind vergleichsweise leicht effizient zu nutzen, da einfache sequentielle Programme die volle Kapazität ausschöpfen können und mehrere Anwendungen durch Prozeßwechsel quasi-parallel ablaufen können. In parallelen und verteilten Systemen können viele konkurrierende Anwendungen tatsächlich parallel bearbeitet werden, ohne daß große Änderungen der Anwendungsprogramme erforderlich sind. Eine einzelne sequentielle Anwendung kann jedoch nur die Rechenleistung einer der Prozessoren, nicht die akkumulierte Systemleistung, ausnutzen. Außerdem sind die Daten, auf denen die Anwen-

dungen operieren, über das System verteilt und müssen entsprechend verwaltet werden.

Wenn verschiedene Anwendungen mit gemeinsamen Datenbeständen arbeiten und Anwendungen mit hohem Ressourcenbedarf parallelisiert werden, um die Rechenleistung mehrerer Prozessoren zu nutzen, stellen Kommunikation, Synchronisation und Datentransfer neue Probleme dar, die die volle Ausnutzung der Systemleistung beschränken. Anstatt die zur Verfügung stehende Rechenkapazität voll zu verwenden, warten die Prozesse der Anwendungen einen Teil der Zeit auf Daten oder auf Meldungen anderer Prozesse derselben Anwendung.

Die Verwaltung gemeinsamer Daten, der Start, der Ablauf und die Terminierung paralleler Anwendungen, die Kommunikation und Synchronisation auf parallelen und verteilten Systemen sollte durch ein verteiltes Betriebssystem unterstützt werden. Derzeit wird die Funktionalität noch oft durch Laufzeitsysteme, die auf herkömmliche monolithische Betriebssysteme aufgesetzt sind, realisiert (Beispiele hierfür sind Transaction-Processing-Monitore für Datenbankanwendungen oder Umgebungen wie PVM oder DCE für heterogen paralleles bzw. verteiltes Rechnen). Auf der Ebene des verteilten Betriebssystems oder eines entsprechenden Laufzeitsystems wird üblicherweise auch die Funktion der automatischen, anwendungsunabhängigen Lastbalancierung angesiedelt.

2.2 Lastverteilung im Mehrbenutzerbetrieb

Große Rechnersysteme und Rechnernetze werden in der Regel nicht exklusiv durch eine Anwendung belegt, sondern viele Benutzer und Rechenaufgaben sind gleichzeitig aktiv. Die Anwendungen laufen im Prinzip unabhängig voneinander; sie benutzen evtl. gemeinsame globale Daten mit gewissen Synchronisationsbedingungen (z.B. Datenbankanwendungen).

Die Vielzahl unabhängiger Anforderungen ermöglicht eine gute Ausnutzung der vorhandenen Rechenkapazitäten, sofern die anstehende Last gut über das System verteilt ist. Das ist jedoch nicht selbstverständlich, da die verschiedenen Anwendungen gegenseitig nicht voneinander wissen und auch von verschiedenen Stellen aus in das Rechnersystem gelangen. Die Lastbalancierung der Anwendungen im System besteht in der Zuordnung und Verteilung der Anwendungen, so daß eine gleichmäßige Auslastung der Ressourcen erreicht wird. Das verspricht den größtmöglichen Durchsatz der Anwendungen insgesamt.

2.3 Lastverteilung in parallelen Anwendungen

Während ein einzelner Rechner durch eine Anwendung voll genutzt werden kann, ist es sehr schwierig, eine Anwendung so in verschiedene Teile zu zerlegen, daß sie, auf ein paralleles System verteilt, tatsächlich schneller abläuft. Probleme wie Datenabhängigkeiten und Kommunikationsaufwand zwischen den Teilprozessen der Anwendung hängen vom verwendeten Algorithmus ab. Darüber hinaus stellt sich aber auch die Frage, wie die Teilabläufe und Daten sinnvoll auf das parallele System zu verteilen sind. Die Lastbalancierung einer parallelisierten Anwendung besteht in der Ausnutzung der Parallelität, soweit sie die Anwendung beschleunigt, und in der Zuordnung und Gruppierung der parallelen Anwendungsteile auf das System, so daß der Zusatzaufwand, der durch Synchronisation und Datenaustausch verursacht wird, gering gehalten wird. Gegenüber der Balancierung vieler unabhängiger Anwendungen bringt die Balancierung parallelisierter Anwendungen die Probleme der Datenkommunikation und der Synchronisationswartezeiten mit sich. Das Ziel der Lastbalancierung besteht hier meist in der schnellstmöglichen Abwicklung der Gesamtanwendung.

2.4 Ziele und Größen in der Lastbalancierung

Dieser Abschnitt wird, ausgehend von einer groben Zieldarstellung für die Lastbalancierung, eine genauere Gliederung der relevanten Elemente und Größen vorstellen. Für eine globale Klassifikation von Lastbalancierungsverfahren sei lediglich auf [Casa88] verwiesen.

2.4.1 Motivation des Lastbegriffs

Der Begriff *Lastbalancierung* wird im Bereich des parallelen Rechnens, ebenso wie die ähnlich klingenden Begriffe *Lastausgleich* und *Lastverteilung*, für die Aufgabe verwendet, Last in Form von Rechenaufgaben geschickt über ein System mit mehreren Rechenkomponenten zu verteilen. Dahinter steht die Beobachtung, daß die anstehende Rechenlast ohne Zutun einer Lastbalancierung ungünstiger verteilt ist. Es ist relativ leicht erkennbar, daß in parallelen und verteilten Systemen oft Rechenknoten nichts sinnvolles zu tun haben. Das bedeutet, daß einige Rechenknoten (Abschnitt 2.4.3) zeitweise keine Rechenaufgaben bekommen haben, oder daß sich alle Rechenaufgaben, die sie bearbeiten, gerade in einem Wartezustand befinden. Ebenso leicht läßt sich feststellen, daß oft Rechenknoten überlastet sind. Als Überlast bezeichnet man in diesem Zusammenhang die Situation, daß der Prozessor aus Sicht der einzelnen Aufträge unerwartet langsam arbeitet. Es ist klar, daß ein Rechenknoten sich nicht in dem Sinne überlasten kann, daß er mehr arbeitet, als seine maximale Rechenleistung zuläßt. Er kann nur pausenlos sinnvoll für Anwendungen tätig sein.

Während also der Begriff der Unterbelastung eines Rechenknotens meist einfach definiert wird durch den Zeitanteil, in dem er nichts sinnvolles für Anwendungen zu tun hat, ist es schwierig, Überlast konkret zu fassen. Interessant ist im Rahmen der Lastbalancierung, wieviel Prozessorleistung für einen neuen Auftrag (Abschnitt 2.4.4) effektiv zur Verfügung stehen würde. Wenn ein neuer Auftrag die volle Leistung des Prozessor erhalten würde, so betrachtet man den Prozessor als unbelastet (als Faktor gesehen ein Lastwert von 1). Wenn der Auftrag nur ein Drittel der vollen Prozessorleistung erhalten würde, d.h. daß er sich die Leistung wohl mit zwei weiteren Prozessen teilen müßte, die dort zur Zeit bearbeitet werden, so kann man ihm einen Lastwert von 3 zumessen. Das am häufigsten verwendete Maß für die Belastung eines Prozessors über ein Zeitintervall ist die mittlere Anzahl laufbereiter Prozesse. Dabei ist ein Prozeß auf einem Prozessor laufbereit, wenn er gerade ausgeführt wird oder wenn er auf eine Zeitscheibe zur Ausführung wartet. Er ist nicht laufbereit, wenn er auf Ein- / Ausgabe, Nachrichten oder Synchronisationsbedingungen (etwa Sperren oder Semaphore) wartet. Dieses Maß setzt voraus, daß jeder Rechenknoten ein Multi-Tasking Betriebssystem einsetzt, d.h. im Zeitscheiben- oder Prioritätenverfahren zwischen mehreren Anwendungsprozessen wechseln kann.

Da die meisten Betriebssysteme eine virtuelle Speicherverwaltung anbieten, kann jeder Anwendungsprozeß und alle Anwendungsprozesse in der Summe mehr Speicher benutzen, als Hauptspeicher auf dem Knoten vorhanden ist. Dies wird durch automatische Ein- und Auslagerung von Hauptspeicherseiten auf einen bzw. von einem Sekundärspeicher realisiert. Verwenden nun die Prozesse auf einem Knoten tatsächlich in der Summe mehr Hauptspeicher, als real vorhanden ist, d.h. sie sprechen diese Speicherseiten häufig an, dann beobachtet man einen Leistungszusammenbruch aus Sicht der Anwendungsprozesse. Bei Speicherüberlastung müssen die Anwendungsprozesse oft warten, bis die angesprochene Speicherseite vom Sekundärspeicher nachgeladen wurde. So ist zwar die Anzahl der laufbereiten Prozesse gering, aber die Rechenleistung des Rechenknotens sinkt drastisch. Oft wird daher der oben vorgestellte Lastfaktor korrigiert, indem man solche Prozesse, die auf Einlagerung ihrer Speicherseite warten, mit zu den laufbereiten Prozessen zählt. Dabei handelt es sich freilich um eine einfache Heuristik.

Nicht alle Anwendungsprozesse verlangen nur Prozessorrechenleistung. Viele Anwendungen enthalten einen beträchtlichen Zeitanteil an Ein- / Ausgabeoperationen. Nun können die Ein- / Ausgabegeräte des Rechenknotens (z.B. Plattenlaufwerke oder Drucker) ebenfalls durch mehrere konkurrierende Anwendungsprozesse belastet werden. Hierbei gelten entsprechende Maße der Unter- und Überlastung, die aussagen, welche Leistung ein Rechenknoten einem neuen Anwendungsprozeß zur Verfügung stellen würde. Insgesamt muß bei Betrachtung mehrerer Ressourcen eines Rechenknotens (z.B. Rechenleistung, Speicherkapazität und Bandbreite des Sekundärspeichers) meist

die am stärksten belastete Ressource als Gesamtlast verwendet werden, da sie die verfügbare Leistung des Knotens für den Anwendungsprozeß beschränkt.

In parallelen und verteilten Systemen genügt es meist nicht, die Last einzelner Rechenknoten zu beurteilen. Das Gesamtsystem wird auch durch die Kommunikation zwischen den Rechenknoten belastet. Analog zur Belastung einer Ressource eines Rechenknotens kann man die Belastung einer Netzverbindung zwischen mehreren Rechenknoten dadurch charakterisieren, welcher Durchsatz an Nachrichten seitens der Anwendungsprozesse vom Netz gefordert wird. So kann eine Netzverbindung mit dem Lastwert 3 beziffert werden, wenn dreimal so viele Daten pro Zeiteinheit zur Übertragung anstehen wie die Verbindung übertragen kann.

2.4.2 Zweck und Potential der Lastbalancierung

Nach der Motivation des Lastbegriffs ist leicht feststellbar, daß die Ressourcen eines parallelen Systems im normalen Betrieb ohne Lastbalancierung ungleich belastet sind. Nun ist automatische Lastbalancierung kein Selbstzweck, sondern soll helfen, die Leistung des gesamten parallelen Systems besser auszunutzen. Auch das ist noch nicht genau ausgedrückt, denn die Anwendungen im System sollen möglichst schnell ablaufen, unabhängig davon, ob tatsächlich jede Ressource im System voll bzw. gleichstark ausgelastet wird. Die Begriffe *Lastbalancierung*, *Lastausgleich* und *Lastverteilung* werden jedoch allgemein für die Aufgabe verwendet, konkurrierende und parallele Anwendungen so auf dem System abzuwickeln, daß sie für sich isoliert oder im Mittel zusammen möglichst schnell ablaufen. Oft wird auch die Stabilisierung der Antwortzeiten einzelner Anwendungen (d.h. Verringerung der Laufzeitschwankungen) als Ziel der Lastbalancierung deklariert. In vielen Ansätzen wird das Ziel der Durchsatzsteigerung tatsächlich allein dadurch angestrebt, daß Lastbalancierung die oben definierte Auslastung der Prozessoren ausgleicht.

Lastbalancierung unterscheidet sich üblicherweise dadurch vom sogenannten *Scheduling*, daß sie die Verarbeitung eines endlosen Flusses von Aufträgen optimiert, während im *Scheduling* eine endliche, oft statisch festgelegte Menge von Aufträgen mit maximalem Durchsatz abzuwickeln ist. Es besteht jedoch in der Literatur eine gewisse Überlappung der Begriffe *Scheduling* und *statische Lastbalancierung*; in dieser Arbeit wird die Unterscheidung gemäß Kapitel 2.5 verwendet.

Inwiefern Lastbalancierung ihr Ziel erreicht, ist schwer festzustellen. Im Bereich des parallelen Rechnens ist es gebräuchlich, die effektive Beschleunigung der Anwendungen zur verfügbaren parallelen Rechenleistung ins Verhältnis zu setzen (der sogenannte *Speedup* dient oft als Maß). Außerdem wird durch den Begriff der Skalierbarkeit (als Maß wird der sogenannte *Scaleup* verwendet) bewertet, wie weit man bei Vergröße-

rung der Anwendungen durch Zufügen weiterer paralleler Rechenleistung die Laufzeiten gleich erhalten kann. Lastbalancierung hilft zwar auch, den *Speedup* und *Scaleup* von Anwendungen in parallelen Systemen zu verbessern, aber das Potential der Lastbalancierung und die Güte, d.h. der tatsächlich erbrachte Nutzen läßt sich nicht direkt durch *Speedup* oder *Scaleup* messen. Das Potential für Lastbalancierung hängt davon ab, wie groß die theoretisch behebbare Ungleichverteilung im System ist. Die Güte einer automatischen Lastbalancierung kann daran gemessen werden, inwieweit sie die Ungleichverteilung erkennt und wieviel davon sie tatsächlich verbessern kann.

In einer Zeit, die im Workstation-Bereich durch jährliche Geschwindigkeitssteigerungen der Rechenleistung um etwa 50% gekennzeichnet ist, erscheint es zunächst müßig, Konzepte zur automatischen, anwendungsunabhängigen dynamischen Lastbalancierung zu entwickeln, die erfahrungsgemäß im langfristigen Mittel Durchsatzsteigerungen zwischen 5 und 10% erbringen können. Während also die Lastbalancierungskonzepte durch Messung dieser Durchsatzsteigerungen validiert und untereinander verglichen werden, besteht die hauptsächliche Motivation für derartige Lastbalancierung in drei anderen Vorteilen:

1. Dynamische Lastbalancierung verhindert katastrophales Durchsatzverhalten und extrem große Antwortzeitschwankungen, die durch unkoordinierte bzw. zufällige Lastverteilung häufig entstehen.
2. Automatische Lastbalancierung reduziert die Komplexität paralleler Anwendungen und erhöht deren Flexibilität und Portabilität, indem sie die Anwendungslogik von der wiederholten, meist systemspezifischen Lösung der Lastverteilungsproblematik entbindet.
3. Dynamische Lastbalancierung ist ein Betriebssystemdienst, der anwendungsübergreifend die Ressourcennutzung verschiedener, konkurrierend ablaufender sequentieller und parallelisierter Anwendungen in parallelen verteilten Systemen optimiert.

2.4.3 Rechenknoten als Lastempfänger

Für die weiteren Kapitel dieser Arbeit sollen nun die für die Lastbalancierung relevanten Elemente und Größen erklärt werden. In der Literatur finden sich unterschiedliche Namensgebungen, daher wird hier eine Namensgebung gewählt, die verbreitet ist und sich für die im Rahmen dieser Arbeit entwickelten Konzepte eignet.

Die wichtigsten Elemente sind die Rechenknoten des Systems. Das parallele und verteilte System wird als ein Netz von Rechenknoten betrachtet. Üblicherweise versteht man unter einem Rechenknoten ein Prozessormodul, d.h. einen Prozessor mit seinem Hauptspeicher und seiner Peripherie. Es ist aber sinnvoller, die Einheit Rechenknoten zu nennen, die von der Lastbalancierung als ein belastbares Element betrachtet wird,

auf dem Anwendungsprozesse laufen. Dadurch kann die Lastbalancierung z.B. enggekoppelte Multiprozessoren als einen Rechenknoten betrachten, wenn sie sich um die Abläufe und die Lastverteilung innerhalb dieses Multiprozessors nicht kümmert. So überläßt man in den meisten bekannten Lastbalancierungsansätzen die Prozeßverteilung innerhalb eines Multiprozessors mit gemeinsamem Speicher der vorhandenen Prozeßverwaltung des Betriebssystems, da sie meist zentral abläuft und sehr gut balanciert. Die Idee besteht darin, eine Warteschlange lauffertiger Prozesse zu verwalten, und den ersten Prozeß daraus auf den Prozessor zu legen, der als nächstes frei wird. Jeder Prozeß, der seine Zeitscheibe verbraucht oder aus anderen Gründen angehalten wird, kommt wieder in die zentrale Warteschlange. Abbildung 1 zeigt ein paralleles Beispielsystem mit Rechenknoten, Abbildung 2 zeigt Beispiele für den internen Aufbau von Rechenknoten.

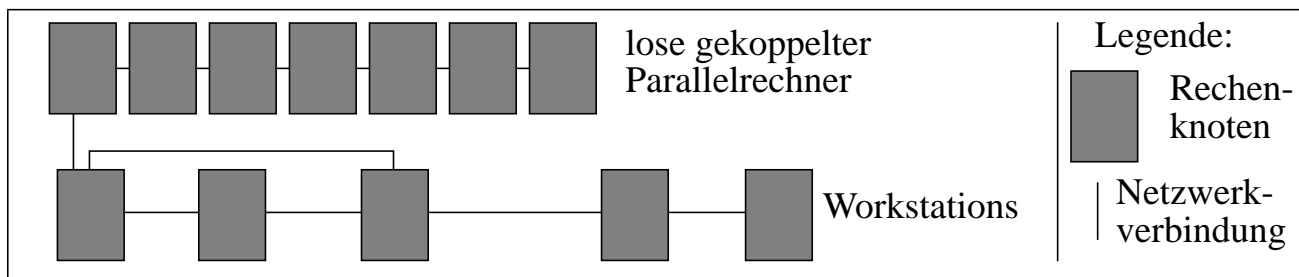


Abbildung 1: Paralleles System als Netzwerk von Rechenknoten.

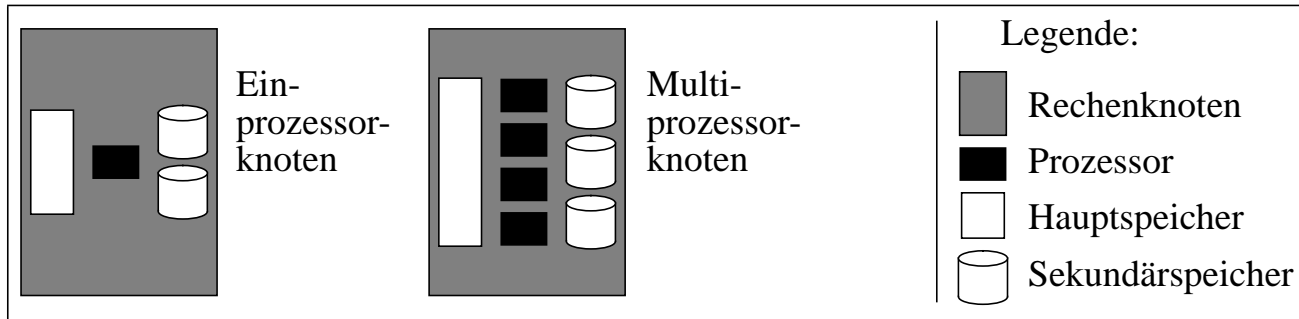


Abbildung 2: Interner Aufbau von Rechenknoten.

Ein Rechenknoten hat für Lastbalancierungszwecke einige statische Attribute, die sich während der Laufzeit von Anwendungen nicht ändern. Am wichtigsten ist die Anzahl der Prozessoren auf dem Rechenknoten und ihre Rechenleistungen. Prozessorleistungen werden gewöhnlich durch sogenannte *Benchmark*-Programme bestimmt und in Einheiten wie MIPS oder MFLOPS angegeben. Weitere Attribute sind die Größe des Hauptspeichers, meist in Byte gemessen, und die Größen und Geschwindigkeiten der angeschlossenen Sekundärspeicher. Während die Größe (Speicherkapazität) in Byte gemessen wird, interessiert für die Geschwindigkeit von Sekundärspeichern die Zugriffszeit für das Auffinden eines Datensatzes und der Durchsatz (gemessen in Bytes pro Sekunde). Die Größe des realen Hauptspeichers ist für Lastbalancierung gewöhn-

lich nur wegen der Effekte der virtuellen Speicherverwaltung interessant (Abschnitt 2.4.1). Die Plattenspeicherkapazität wird äußerst selten beachtet; gewöhnlich sieht man die Aufgabe der Lastbalancierung nicht darin, Speicherüberlauf zu vermeiden, sondern den Systemdurchsatz zu steigern unter der Annahme, daß genügend Speichermedien vorhanden sind.

Die Attribute eines Rechenknotens, die sich zur Laufzeit ändern, charakterisieren die Belastung des Knotens. Dazu wird hier der Begriff der Aufträge verwendet, der unten genauer spezifiziert wird. Wie in den Abschnitten 2.4.1 und 2.4.8 erläutert, kann ein Rechenknoten mehrere Aufträge quasi-parallel bearbeiten, indem er zwischen den laufbereiten Prozessen wechselt. Für die Lastbalancierung ist wichtig, wieviele Aufträge momentan auf dem Rechenknoten zur Bearbeitung liegen und wie stark sie welche Ressourcen des Rechenknotens tatsächlich auslasten (Abschnitt 2.4.1).

2.4.4 Aufträge als Lasterzeuger

Die Anwendungen, Gruppen von Anwendungen oder Teile von Anwendungen, die aus Sicht der Lastbalancierung Last-erzeugende Objekte sind, unterscheiden sich in den verschiedenen veröffentlichten Ansätzen. In dieser Arbeit wird mit Auftrag der kleinste Objekttyp bezeichnet, den die Lastbalancierung identifizieren, beobachten und evtl. beeinflussen kann. Das muß nicht ein Auftrag (oder eine Aufgabe, Anwendung) im Sinne eines Benutzers sein. In der Literatur werden oft die Betriebssystem-Prozesse (oder auch sogenannte Threads) als Aufträge betrachtet, seltener Gruppen von Prozessen, die eine Anwendung ausführen. In Datenbank-Umgebungen werden oft Datenbank-Transaktionen als Aufträge betrachtet. Im folgenden soll die Einschränkung gelten, daß jeder Auftrag im Prinzip auf einem einzigen Rechenknoten ablaufen kann. „Aufträge“, die mehrere Rechenknoten - parallel oder nacheinander - verwenden, müssen in diverse Aufträge zerteilt werden. Da in dieser Arbeit anwendungsunabhängige Lastbalancierung untersucht wird, ist das Auftragsgranulat allein durch die Anwendung bestimmt.

Für die Belange der Lastbalancierung hat jeder Auftrag statische Attribute, d.h. Eigenschaften, die während seiner Existenz unveränderlich sind. Wichtig ist vor allem die Größe des Auftrags, d.h. der Gesamtbedarf an verschiedenen Ressourcen. Die genauere Ablaufstruktur innerhalb eines Auftrages wird von der Lastbalancierung gewöhnlich nicht betrachtet, da nach obiger Definition des Auftragsbegriffs Aufträge nur als ganze beobachtet und balanciert werden. Innerhalb von Auftragsgruppen kann jeder Auftrag Vorgänger- und Nachfolgeaufträge besitzen; d.h. Aufträge, auf deren Beendigung er warten muß und Aufträge, die auf seine Beendigung warten. Schließlich kann ein Auftrag vom Anwender eine gewisse Priorität zugemessen bekommen. Bei Verwendung von Prioritäten gewichten die Anwender normalerweise die Wartezei-

ten (Laufzeiten der Aufträge) nach den Prioritäten. Zur Beurteilung des Gesamtdurchsatzes ist es dann wichtig, daß die Aufträge mit hoher Priorität besonders schnell ablaufen. Es sei noch erwähnt, daß für Aufträge auch Zeitgrenzen bestehen können, die bei der Bearbeitung einzuhalten sind. In dieser Arbeit werden jedoch Benutzer-gegebene Prioritäten und Zeitgrenzen für Aufträge nicht weiter betrachtet.

Dynamische Attribute beschreiben den Bearbeitungszustand eines Auftrags. Relevant sind dabei der bisher bereits abgelaufene Arbeitsanteil sowie der Rechenknoten, auf dem der Auftrag momentan wartet oder bearbeitet wird. Weiterhin ist die Zeit wichtig, die seit Absendung des Auftrages bereits verstrichen ist. Bei einzelnen Aufträge ist das nur interessant für die Antwortzeiten, aber innerhalb von zusammenhängenden Auftragsgruppen kann es auch den Durchsatz beeinflussen, da an jedem Auftrag Folgeaufträge hängen können, die auf ihn warten.

Aufträge sind zwar die Objekte, die von der Lastbalancierung direkt betrachtet und manipuliert werden, aber es ist oft vorteilhaft, die Zusammenhänge zwischen verschiedenen Aufträgen zu berücksichtigen. Zwischen den Aufträgen einer Gruppe können, wie oben erwähnt, Reihenfolgebeziehungen bestehen. Dadurch kann die Lastbalancierung ermitteln, welche Auftragslast wann entstehen wird. Weiterhin können zwischen Aufträgen einer Gruppe Kommunikationsbeziehungen bestehen. Meist wird spezifiziert, wie häufig welche Datenmengen zwischen Auftragspaaren ausgetauscht werden. Alternativ kann ein Datenfluß spezifiziert werden, d.h. welche oder wieviel Ergebnisdaten für welche Folgeaufträge als Eingabedaten verwendet werden. Als dynamisches Attribut ist für Lastbalancierungszwecke der Bearbeitungszustand der Aufträge in einer Gruppe relevant.

2.4.5 Netzwerk und Kommunikation

Die Netzwerkverbindungen zwischen den Rechenknoten des parallelen und verteilten Systems sind wichtige Objekte für die Lastbalancierung. Hier soll eine bidirektionale Verbindung zwischen mehreren Rechenknoten als ein Netzwerkanal bezeichnet werden. Abhängig von der Netzwerktopologie kann ein Kanal eine Punkt-zu-Punkt Verbindung zwischen zwei Rechenknoten, oder eine Stern-, Ring- oder Busverbindung zwischen mehreren Rechenknoten sein. Wichtig ist jedoch, daß ein Netzwerkanal genau ein Medium ist, d.h. eine eindeutige Nachrichtenbelastung aufweist.

Für die Lastbalancierung sind als statische Attribute der Durchsatz und die Latenzzeit eines Kanals wichtig. Der Durchsatz ist die Datenmenge, die in einer bestimmten Zeit übertragen werden kann, gemessen in der Einheiten wie Bytes pro Sekunde. Die Latenzzeit ist der Zeitbedarf, um eine kurze Nachricht zwischen Rechenknoten auszutauschen. Dabei wird meist die sogenannte Software-Latenzzeit verwendet; sie enthält

neben der reinen Netzübertragung auch die Zeit auf den Rechenknoten, die vom Betriebssystem benötigt wird, um die Nachrichten den Prozessen abzunehmen bzw. zugänglich zu machen.

Last auf Kanälen wird durch Aufträge erzeugt, die miteinander kommunizieren. Diese Netzbelastung wird in den diversen Lastbalancierungskonzepten unterschiedlich gefaßt. Einige Ansätze betrachten die Häufigkeit und mittlere Größe der Nachrichten zwischen Aufträgen, andere unterscheiden lediglich, ob Kommunikation zwischen Auftragspaaren stattfindet, d.h. eine logische Verbindung besteht. Schließlich betrachten einige Lastbalancierungskonzepte Datensätze, die zwischen Aufträgen ausgetauscht oder weitergegeben werden und dabei Netzlast erzeugen (Abschnitt 2.4.6).

In Lastbalancierungsansätzen, die als belastbare Elemente nur Rechenknoten betrachten, müssen Netzverbindungen bei jedem Rechenknoten als Ein- / Ausgabegerät betrachtet werden.

2.4.6 Daten als besondere Ressourcen

Einige Ansätze zur Lastbalancierung enthalten Daten explizit als Elemente im Modell. Die Objektorientierung verlangt, daß alle Daten durch darauf definierte Funktionen gekapselt werden, doch im Bereich der Datenverwaltung und in allen gängigen Betriebssystemen gibt es globale Daten. Flüchtige Daten werden in gemeinsamen Hauptspeicherseiten, persistente Daten in Dateien aufbewahrt. Relevant für Lastbalancierung ist die Größe der Daten und der Aufenthaltsort. Das können je nach System statische oder dynamische Eigenschaften sein. Die Menge der Daten ist wichtig, um den Speicherbedarf, den Aufwand für Zugriffe und den Aufwand zur Bewegung der Daten zwischen Rechenknoten abzuschätzen. Wenn von Daten Kopien (allgemeiner Versionen) existieren können, so ist auch der Aufenthaltsort der Kopien relevant für die Lastbalancierung, denn sie kann die lokale Verfügbarkeit bei lesenden Zugriffen abschätzen und bei Modifikation der Daten den Aufwand abschätzen, um alle Kopien zu invalidieren oder zu aktualisieren.

2.4.7 Weitere Elemente in der Lastbalancierung

Wenn die Aufgabe der Lastbalancierung nicht zentral für das gesamte System wahrgenommen wird, sondern in mehrere Komponenten aufgeteilt wird, so sind Nachbar-Cluster und benachbarte Lastbalancierungskomponenten ebenfalls relevante Elemente für die Lastbalancierung. Die Zusammenarbeit ist natürlich nicht auf unmittelbare physische Nachbarschaft beschränkt. Abschnitt 2.5.3 stellt Verfahren zur Strukturierung der Lastbalancierungsaufgabe vor. Über ganze Cluster (Abschnitt 2.5.3.2) benötigt Lastbalancierung möglichst aggregierte statische und dynamische Informationen, die aussa-

gen, welches Leistungspotential und welche Belastung in den Clustern aktuell vorhanden ist.

Schließlich sollte sich die Lastbalancierung ihrer selbst bewußt sein, d.h. sie sollte den Nutzen und die Kosten, die sie mit sich bringt, einbeziehen. Informationen über den Nutzen der Lastbalancierung bestehen hauptsächlich aus Vorabschätzungen, wie die Lastsituation und die Anwendungsverläufe sich ändern, wenn die Lastbalancierung in den Systemablauf eingreift. Dadurch kann für weitere Planung schon von einer durch die Lastbalancierung verbesserten Situation ausgegangen werden. Information über die Kosten der Lastbalancierung ist etwa die Rechenlast, die bei der Durchführung der Lastbalancierungsaufgabe entstehen, denn die Lastbalancierung findet üblicherweise auf Betriebssystemebene selbst statt und kostet Rechenzeit. Weiterhin verursacht die Lastbalancierung durch die Entscheidungs-, Zuweisungs- und Umverteilungsprozesse auch Verzögerungen der Auftragsausführung.

2.4.8 Das Modell zur Bearbeitung von Aufträgen

Die Abläufe in dem parallelen und verteilten Rechnersystem sollen im folgenden mit den Elementen modelliert werden, die in den obigen Abschnitten eingeführt wurden. Aus Sicht der Lastbalancierung ist eine nicht endende Menge irgendwann in das Rechnersystem kommender Aufträge abzuarbeiten, so daß das Rechnersystem möglichst viel Auftragsgröße pro Zeiteinheit erledigt (Maximierung des Systemdurchsatzes). Die Bearbeitung eines Auftrages kann im Prinzip auf einem beliebigen Rechenknoten des Systems erfolgen (siehe auch unten), indem der Rechenknoten dem Auftrag eine Zeitlang gewisse Ressourcen zur Verfügung stellt. Wie ein Auftrag auf einem Rechenknoten genau verarbeitet wird, entzieht sich dem Zuständigkeitsbereich der Lastbalancierung. Sie kann aber statische und dynamische Informationen über den (verbleibenden) Ressourcenbedarf eines Auftrags und das (momentane) Ressourcenangebot eines Rechenknotens haben, wie in den Abschnitten 2.4.3 und 2.4.4 aufgelistet wurde.

Aufträge werden auf Rechenknoten berechnet und erzeugen dabei Last. Zwei Ablaufmodelle sind im Bereich der Lastbalancierung am stärksten verbreitet: das Warteschlangenmodell und das *Multitasking*-Modell. Im Warteschlangenmodell bearbeitet ein Rechenknoten die Aufträge streng sequentiell und besitzt eine Warteschlange für die eingetroffenen Aufträge. Das *Multitasking*-Modell entspricht dem Verarbeitungsprinzip der heute üblichen Betriebssysteme. Alle Aufträge (Prozesse) auf dem Rechenknoten werden im Zeitscheiben- oder Prioritätsverfahren quasi-parallel bearbeitet. Dabei können auf Multiprozessorknoten entsprechend mehrere Aufträge wirklich parallel bearbeitet werden. Die Aufträge beanspruchen nicht alle dauernd die eigentliche Prozessorleistung des Rechenknotens, sondern tätigen auch Zugriffe auf Ein- / Ausga-

begeräte oder warten auf das Eintreten bestimmter Synchronisationsbedingungen. Das übliche Ablaufmodell befähigt daher jeden Rechenknoten, die verschiedenen Ressourcenbedürfnisse der Aufträge gleichzeitig zu befriedigen, solange sie sich nicht überschneiden. Das bedeutet zum Beispiel, daß ein Auftrag Daten von einer Festplatte lesen kann, während ein anderer Auftrag im Prozessor rechnet. In der Realität ist zu beachten, daß das häufige Umschalten zwischen Auftragsbearbeitungen auf einem Prozessor (Kontextwechsel) erheblichen Zusatzrechenaufwand im Betriebssystem mit sich bringt, d.h. den Durchsatz der Anwendungen reduziert.

Oft wird zwischen preemptiven und nicht-preemptiven Ablaufmodellen unterschieden. Bei preemptiven Abläufen kann ein laufender Auftrag jederzeit unterbrochen und später, evtl. auf einem anderen Rechenknoten fortgesetzt werden. In nicht-preemptiven Abläufen belegt jeder Auftrag den Rechenknoten ununterbrochen, bis er abgeschlossen ist. Die meisten Betriebssysteme ermöglichen heute preemptive Abläufe, wobei Prozessorwechsel nur innerhalb eines Rechenknotens möglich sind (dieses Ablaufmodell wird auch als lokal-preemptiv bezeichnet).

Ein weiteres für Lastbalancierung relevantes Ablaufkonzept ist das Client - Server Modell. Es gibt im System eine Reihe von Diensten, die man auch Funktionen oder Serverklassen nennt. Jeder Auftrag verlangt die Ausführung eines bestimmten Dienstes mit bestimmten Parametern. Das Modell ist für Lastbalancierung dann interessant, wenn die Instanzen solcher Dienste als feststehende Prozesse im System warten und nicht *multi threaded* sind, d.h. wirklich einen Auftrag nach dem anderen sequentiell abarbeiten. Ein Auftrag bewirkt dann die zeitweilige Aktivierung einer der zur Verfügung stehenden Dienstprozesse. Das Modell wird eingesetzt, wenn Aufträge sehr häufig und kurz sind, oder aus anderen Gründen das Erzeugen und Beenden von Prozessen für das Betriebssystem sehr aufwendig ist. Für die Lastbalancierung bedeutet das, daß auf jedem Rechenknoten durch Aufträge höchstens alle dort wartenden Serverprozesse aktiviert werden können, während weitere Aufträge für diesen Rechenknoten an irgendeiner Stelle warten müssen. Andernfalls müssen weitere Prozesse für den Dienst gestartet werden.

Zuletzt muß festgelegt werden, ob jeder Auftrag von jedem Rechenknoten alleine seine gesamten Ressourcenbedürfnisse erhalten kann. Es gibt Lastbalancierungsansätze, in denen Aufträge nur auf bestimmten Rechenknoten ausgeführt werden können (z.B. weil dort der ausführbare Programmcode oder bestimmte Daten vorhanden sind). Im folgenden soll das Modell verwendet werden, daß jeder Rechenknoten alle Aufträge ausführen kann. Das setzt voraus, daß alle Daten im verteilten Rechnersystem von jedem Knoten aus zugreifbar sind (nicht unbedingt mit gleichem Aufwand). Das Modell ist nicht anwendbar, wenn einige Rechenknoten besondere Ressourcen, wie z.B. bestimmte Ein- / Ausgabegeräte besitzen, die andere Knoten nicht aufweisen.

Die Bearbeitung eines Auftrags kann also alleine auf einem Knoten ablaufen unter der Annahme, daß jeder Knoten alle Ressourcen verfügbar hat. Der Auftrag kann aber neben dem Konsum verschiedener Ressourcen mit anderen Aufträgen zusammenarbeiten, d.h. mit ihnen Nachrichten austauschen oder auf bestimmte Synchronisationsbedingungen warten. In Lastbalancierungskonzepten, die gemeinsame bzw. globale Daten kennen, muß auch der Zugriff auf gemeinsame Daten modelliert werden. Zugriffe auf Daten, die lokal auf dem Rechenknoten vorhanden sind, werden üblicherweise als kostenlos betrachtet. Zugriffe auf Daten, die auf anderen Rechenknoten liegen, erzeugen Wartezeiten für den zugreifenden Auftrag, Kommunikationslast auf den Verbindungskanälen zu dem Rechenknoten, bei dem die Daten liegen und evtl. Rechenlast auf einem oder auf beiden Rechenknoten. In manchen Lastbalancierungsansätzen bleiben Daten statisch auf ihren Knoten liegen, in anderen können sie sich bewegen. Außerdem muß im Verarbeitungsmodell festgelegt werden, ob ein Datenzugriff eines Auftrags die Daten heranholt, um auf ihnen zu arbeiten, oder ob er eine Zugriffsoperation an den Rechenknoten sendet, der dann den Zugriff ausführt.

2.5 Aufgaben und Struktur der Lastbalancierung

Das Ziel der Lastbalancierung ist, wie bereits erklärt, die Verbesserung des Systemdurchsatzes. Sie soll durch Regelung der Prozessorlasten, der Kommunikationslasten und der Belastung der Ein- / Ausgabegeräte sinnvolle Arbeitspunkte für die Rechenknoten und damit maximalen Durchsatz erreichen. Bestehende Verteilungen sind zu verbessern und durch Vorplanung sind Ressourcen für erwartete Auftragslasten geeignet zu reservieren.

In diesem Kapitel werden zunächst die Erwartungen an Lastbalancierungsmechanismen vorgestellt. Dann wird untersucht, wie diese Funktionalität einzuteilen ist und wie Lastbalancierung auf das System verteilt werden kann. Schließlich werden mögliche Flexibilitätsstufen der Lastbalancierung vorgestellt.

2.5.1 Aufgaben der Lastbalancierung

Unter dem oben beschriebenen System- und Verarbeitungsmodell hat Lastbalancierung hauptsächlich die Aufgabe, Aufträge geschickt auf das System zu verteilen bzw. umzuverteilen. Sie hat zu entscheiden, wann und wo Aufträge bearbeitet werden sollen.

Bei preemptiven Verfahren kann Lastbalancierung jeden Auftrag stückweise auf verschiedenen Rechenknoten bearbeiten lassen (Migration) und ihn evtl. zwischendurch warten lassen (suspendieren). Bei nicht-preemptiven Verfahren kann sie jeden Auftrag nur zu einem beliebigen Zeitpunkt auf einem beliebigen Rechenknoten starten.

Es gibt im Bereich der rein funktionalen Aufträge auch die Möglichkeit, Aufträge zu duplizieren. Ein Auftrag wird auf mehreren Rechenknoten, d.h. mehrfach gestartet und entweder wird das Ergebnis des ersten verwendet oder jeder der duplizierten Aufträge gibt sein Ergebnis an einen der Folgeaufträge weiter. Auf die Möglichkeit der Auftragsduplikation zur Lastbalancierung soll hier nicht weiter eingegangen werden, da in der Realität fast alle Auftragsstypen Seiteneffekte haben, weil sie auf globalen Daten arbeiten oder einen globalen Bearbeitungszustand ändern.

Viele Betriebssysteme berücksichtigen bei der (quasi-) parallelen Bearbeitung der Prozesse auf einem Rechenknoten auch Prozeßprioritäten. So bekommen Aufträge höherer Priorität öfter Zeitscheiben oder längere Zeitscheiben, in manchen Systemen wird die Bearbeitung eines Prozesses unterbrochen zugunsten eines höher-priorisierten, der lafbereit wurde. Lastbalancierung kann den Aufträgen entsprechende Prozeßprioritäten zuweisen, um kritischere Aufträge schnell zu Ende zu führen auch wenn auf dem Rechenknoten noch andere Aufträge konkurrierend laufen.

Anwendungsspezifische Lastbalancierungsverfahren haben teilweise Einfluß auf das Auftragsgranulat. Sie können z.B. Aufträgen mitteilen, daß sie sich in mehrere Aufträge feineren Granulats zerspalten sollen, oder die Lastbalancierungsverfahren geben den Aufträgen Hinweise über ein sinnvolles Granulat etwa entstehender Folgeaufträge. In dieser Arbeit soll jedoch nur anwendungsunabhängige Lastbalancierung verfolgt werden.

Wenn das Modell für die Lastbalancierung explizit Daten kennt (Abschnitt 2.4.6), so kann die Lastbalancierung den Verarbeitungsdurchsatz der Aufträge auch dadurch beeinflussen, daß sie Datensätze auf bestimmte Rechenknoten plaziert, sie migriert, oder geeignet Kopien anlegen läßt. Anwendungsspezifische Lastbalancierung könnte weiterhin auch Einfluß auf das Granulat der Datensätze nehmen.

Lastbalancierung kann auch die Konfiguration des Systems kontrollieren, d.h. Rechenknoten zu- oder abschalten, Server auf Rechenknoten zuaddieren oder stillegen oder den Grad an genutzter Quasi-Parallelität (Multitasking) auf den Rechenknoten ändern.

2.5.2 Komponenten in der Lastbalancierung

Die Aufgaben der Lastbalancierung werden gewöhnlich in drei Teilbereiche gegliedert: die Sammlung und Verwaltung relevanter Informationen (die sogenannte Informationssammel-Strategie), die Entscheidung, ob das Lastungleichgewicht so groß ist, daß Aufträge verlagert werden sollten zusammen mit der Entscheidung, welche Aufträge zu verlagern sind (Transferstrategie) und schließlich die Entscheidung, wohin diese Aufträge zu migrieren bzw. zuzuweisen sind (Lokationsstrategie). Obwohl sich

diese Einteilung in der Literatur durchgesetzt hat, ist sie nicht allgemein und entstammt den dezentralen Lastbalancierungsansätzen (siehe unten). Etwas allgemeiner erfüllt Lastbalancierung ihre Aufgabe durch folgende Aktivitäten.

Die Informationsverwaltung sammelt, speichert und interpretiert die gemessenen (a posteriori) Zustandsinformationen der Rechenknoten, der Netzwerkkanäle, der Aufträge und der Daten sowie die von Aufträgen oder Auftragsgruppen erhaltenen Vorabschätzungen. Sie entscheidet, welche Meßwerte relevant sind und wie akkurat bzw. aktuell sie sein müssen. Die Informationsverwaltung aktiviert die anderen Komponenten der Lastbalancierung periodisch oder aufgrund neuer Informationen, die Handlungsbedarf fordern. Die Informationsverwaltung selbst wird entweder periodisch aktiv, um Meßwerte zu sammeln, oder wird durch eintreffende Meßwerte, Prognosen oder Meldungen angestoßen.

Eine Zuweisungskomponente entscheidet, wann welche Aufträge auf welchem Rechenknoten gestartet werden sollen. Sie verwaltet daher die Aufträge, die noch nicht in Bearbeitung sind oder die zwischenzeitlich wieder aus der Bearbeitung suspendiert worden sind. Sie wird aktiv, wenn neue Aufträge in das System gelangen oder sich der Systemlastzustand signifikant ändert.

Eine Korrekturkomponente schließlich entscheidet, wann der Systemzustand nicht mehr akzeptabel ist bzw. wann sich ein Eingriff durch die Lastbalancierung lohnt. Die Lastbalancierung kann den Verarbeitungsverlauf beeinflussen, indem sie laufende Aufträge suspendiert und der Zuweisungskomponente übergibt. Die Zuweisungskomponente kann suspendierte Aufträge später weiterarbeiten lassen und sie evtl. auch vorher auf andere Rechenknoten migrieren.

Entsprechende Komponenten werden benötigt, wenn die Lastbalancierung Daten kennt und sich um deren günstige Verteilung kümmert. Weiterhin ist je nach Aufgabenbereich der Lastbalancierung eine Komponente für längerfristige Änderungen der Systemkonfiguration verantwortlich. Sie wird seltener aktiviert, nämlich dann, wenn durch Eingriffe der Zuweisungs- und Korrekturkomponenten keine Verbesserung der Systemlastsituation erreichbar ist. In adaptiven Lastbalancierungsansätzen ist eine weitere Komponente dafür zuständig, die Strategie an die aktuellen Gegebenheiten anzupassen (Abschnitt 2.5.6).

2.5.3 Struktur der Lastbalancierung

Während im Abschnitt 2.5.2 die Funktionalität der Lastbalancierung logisch in Teilfunktionen gegliedert wurde, soll hier untersucht werden, wie Lastbalancierung physisch auf großen parallelen und verteilten Systemen realisiert werden kann. Damit sind

zentrale Strukturen bzw. Kooperationsstrukturen mehrerer - im Prinzip vollständiger und autonomer - Lastbalancierungsagenten gemeint.

2.5.3.1 Zentrale Lastbalancierung

In zentraler (oft auch als ‘global’ bezeichneter) Lastbalancierung [Chow79], [Cope88], [Efe89], [Lin92] ist ein einziger Agent für die gesamte Funktionalität, d.h. Informationssammlung und Entscheidungsfindung für das gesamte System zuständig (Abbildung 3). Rein logisch betrachtet ist diese Struktur aus mehreren Gründen optimal:

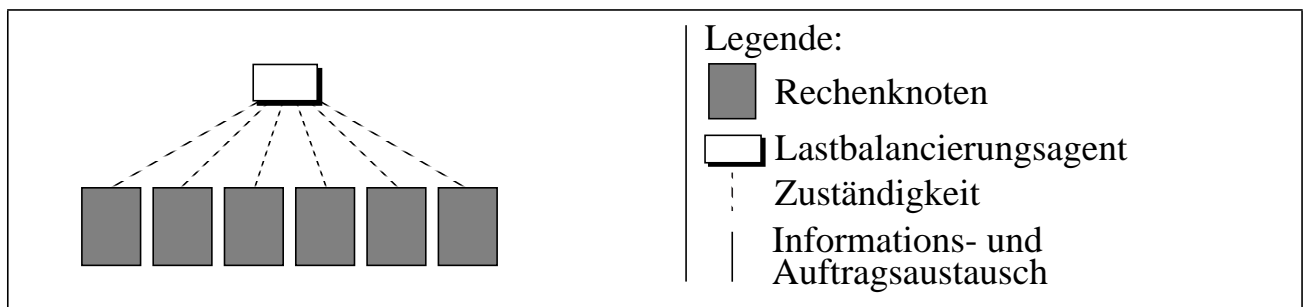


Abbildung 3: Zentrale (globale) Lastbalancierungsstruktur.

- Die gemessenen Informationen und Prognosen über den Systemlast- und Anwendungszustand sind eindeutig beim zentralen Agenten verfügbar und müssen nicht über das System verteilt oder repliziert werden. Die Verteilung und Replikation der Informationen würde dagegen Nachrichtenverkehr erzeugen und hätte unterschiedlich genaue und verschieden alte Informationen zur Folge. Das wiederum würde zu widersprechenden Entscheidungen führen.
- Die Lastbalancierung kann das globale Wissen über den Zustand des Systems, über den Verlauf der Anwendungen und die Abhängigkeiten zwischen den Aufträgen der Anwendungen ausnutzen. Das ermöglicht globale Lastbalancierung im Zusammenspiel der Ressourcen und Aufträge und vermeidet kontra-produktive Entscheidungen, wie sie durch verschiedene Agenten, die mit Teilinformationen arbeiten, auftreten würden. Fortgeschrittene komplexe Strategien können realisiert werden.
- Eine zentrale Zuweisung neu angekommener Aufträge kann starke Lastungleichgewichte von Anfang an vermeiden, wenn ein Balancierungsagent alle Aufträge zentral günstig auf das System verteilt. Lastungleichgewicht tritt dann nur durch mangelnde Vorhersehbarkeit der Auftragsprofile oder suboptimale Entscheidungsalgorithmen auf. In dezentralen Strukturen ebenso wie in zentralen Strukturen, bei denen die Aufträge zunächst auf ihren Ursprungsknoten verbleiben, entsteht dagegen Last zunächst auf den Knoten, auf denen die Aufträge erzeugt wurden, woraufhin die Lastbalancierung versucht, die Last auszugleichen.

Es gibt keine logischen Nachteile zentraler Lastbalancierungsstrukturen (in Bezug auf das Lastbalancierungspotential; Aspekte wie Fehlertoleranz werden hier nicht betrachtet). Die Grenzen der zentralen Lastbalancierung sind die Anzahl entstehender Aufträge und die Anzahl der Rechenknoten im System. Zentrale Lastbalancierung ist nicht beliebig skalierbar, denn der mit Lastbalancierung verbundene Rechenaufwand und die entstehenden Verzögerungen wachsen zumindest linear mit der Ankunftsrate neuer Aufträge und der Anzahl verfügbarer Rechenknoten. Eine genauere Aufwandsabschätzung läßt sich nur für konkrete Balancierungsalgorithmen angeben, generell müssen aber gemessene und prognostizierte Informationen von allen Rechenknoten und Aufträgen (sofern verfügbar) gesammelt werden. Bei jeder Zuweisungs- oder Migrationsentscheidung, genauer gesagt bei jedem Versuch einer solchen Entscheidung muß - bei wachsender Systemgröße und Auftragszahl - mehr Information durchgearbeitet und mehr Alternativen gegeneinander erwogen werden. Dadurch verbraucht der Lastbalancierungsagent selbst zunehmend Rechenzeit, die eigentlich zur Bearbeitung von Aufträgen genutzt werden sollte und die Zeitspanne zwischen der Entstehung eines Auftrags und dem Bearbeitungsbeginn wächst. Diese Verluste verringern die durch Lastbalancierung theoretisch mögliche Durchsatzsteigerung.

2.5.3.2 Dezentrale Lastbalancierung

Dezentrale Lastbalancierung (oft auch als ‘verteilt’ bezeichnet) verteilt die Informationsverwaltung oder die Zuweisungs- und Migrationsentscheidungen (zumeist alle drei) auf das Rechnersystem. Am häufigsten wird die völlig dezentrale Struktur gewählt [Baum88], [Kale88], [Lin87], [Lüli91], bei der jeder Rechenknoten einen eigenen, vollständigen Lastbalancierungsagenten beherbergt (Abbildung 4 links), es gibt aber auch Ansätze [Evan94], [Gopi91], [Zhou92], bei denen Cluster durch zentrale Balancierungsagenten verwaltet werden und die Agenten untereinander dezentral kooperieren (Abbildung 4 rechts). Die Idee besteht darin, daß sehr große Systeme in Teile zerspalten werden, so daß immer Rechenknoten, die durch leistungsfähige Kommunikationskanäle verbunden sind, ein Cluster ergeben. Innerhalb eines Clusters wird zentrale akkurate Lastbalancierung durchgeführt, während zwischen Clustern nur lose Interaktion für groben Lastausgleich stattfindet. Wenn diese Zerteilung des Systems nicht flach ist, sondern rekursiv erfolgt, erhält man eine hierarchische Struktur (Abschnitt 2.5.3.3). In jedem Falle muß bei großen Systemen die Kooperation zwischen Lastbalancierungsagenten auf direkte geometrische oder etwas erweiterte Nachbarschaftsbeziehungen eingeschränkt werden. Ansonsten entstehen dieselben Engpässe wie bei zentraler Balancierung, d.h. der Balancierungsaufwand steigt mit der Zahl der Nachbar-Cluster.

In der dezentralen Struktur hat jeder Lastbalancierungsagent Informationen über seine Rechenknoten und dortige Aufträge sowie über die Lastsituation der Nachbar-Cluster.

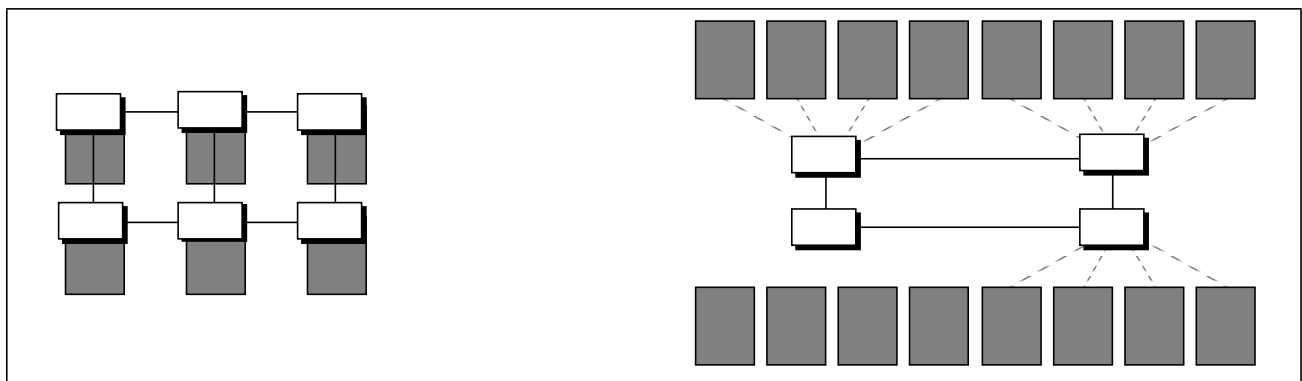


Abbildung 4: Vollständig dezentrale und allgemein dezentrale Balancierungsstruktur.

Er teilt den Nachbar-Clustern signifikante Änderungen seiner Systemlast mit. Jeder Agent kann Aufträge von Nachbar-Clustern anfordern, wenn er sich unterbelastet fühlt (sogenannte Empfänger-initiierte Lastbalancierung [Eage86]) oder er kann Aufträge an Nachbarn abgeben, wenn er meint, daß diese Nachbar-Cluster signifikant weniger Last aufweisen (Sender-initiierte Lastbalancierung). Einige Ansätze bedienen sich komplexerer Verhandlungsschemata zwischen den Lastbalancierungsagenten.

Es gibt in der dezentralen Struktur keinen Ort im System, an dem globale Last- oder Zustandsinformationen verfügbar sind; alle Entscheidungen werden autonom von den Balancierungsagenten getroffen. Diese Struktur ist offensichtlich skalierbar, da wachsende Prozessorzahlen keinen Mehraufwand für die einzelnen Lastbalancierungsagenten zur Folge haben. Der andere Hauptvorteil der dezentralen Struktur liegt darin, daß Lastbalancierung nicht viel zu tun hat, wenn die Last auf dem System einigermaßen gleichverteilt ist; sie erzeugt dann auch keine Störungen oder Behinderungen der Auftragsbearbeitung. In dezentralen Verfahren wird meist jeder Auftrag zuerst dort gestartet bzw. in eine Warteschlange eingereiht, wo er entstand. Nur bei Lastausgleich weist die Lastbalancierung ihn einem anderen Knoten zu bzw. migriert ihn. Speziell in Situationen hoher Gesamlast im System ist es wichtig, daß Lastbalancierung keine unnötige Zusatzlast erzeugt. Zentrale Lastbalancierung wird dagegen um so aktiver, je mehr und je häufiger Aufträge im System entstehen, denn sie muß über jeden Auftrag entscheiden.

Derartige Gegenüberstellung zentraler und dezentraler Strukturen [Thei88] fällt in der Literatur oft zu grob aus. Eine stabile Hochlastsituation durch langlaufende Aufträge macht der zentralen Lastbalancierung noch keine Probleme, da nicht viele Entscheidungen zu treffen sind. Kritisch sind dagegen Wellen hoher Ankunftsdaten, die meist sehr schiefe Lastverteilung bewirken, weil sie durch eine oder wenige Anwendungen verursacht werden. Hier wird zentrale Lastbalancierung zwar schnell überlastet, aber sie kann die Ungleichverteilung von vornherein abfangen, während bei dezentraler Lastbalancierung eine längere Periode von Lastausgleichsaktionen folgt, bis sich die

Auftragslast sukzessive einigermaßen auf das System verteilt hat. Gegenüber zentraler Balancierung besteht außerdem die Gefahr, daß Aufträge in kritischen Situationen mehrfach zwischen Rechenknoten ausgetauscht werden, bevor sie endlich bearbeitet werden.

Sehr kurze Aufträge können nicht gut auf parallele Systeme verteilt werden. Dezentrale Ansätze lassen kurze Aufträge meist grundsätzlich lokal ablaufen und vermeiden dadurch Lastbalancierungsaufwand, der sich für die kurzen Aufträge nicht lohnt; Es gibt aber nur wenige dezentrale Ansätze, die solche Vorabschätzungen überhaupt verwenden - meist werden alle Aufträge als gleich groß angenommen. Zentrale Strukturen haben hingegen einen festen Mindestaufwand auch für kurze Aufträge, so daß sie bei zu feinem Auftragsgranulat und hohen Auftragsankunftsrate leicht überlastet werden.

2.5.3.3 Hierarchische Lastbalancierungsstrukturen

In hierarchischen Strukturen wird jeweils eine Gruppe von Lastbalancierungsagenten durch einen Agenten einer höheren Ebene verwaltet (Abbildung 5). Agenten höherer Ebenen haben einen größeren Überblick und sollten abstrakte Entscheidungen treffen. Sie verwalten meist Rechenknoten-Gruppen und Auftragsgruppen und treffen Zuweisungsentscheidungen, die auf niedrigeren Ebenen verfeinert werden [Ahma94], [Tilb81], [Tilb84]. Die hierarchische Verteilung erfordert zunächst mehr Informationsverwaltung und mehr Entscheidungen von Agenten höherer Ebene. Daher ist es unbedingt notwendig, daß Informationen nach oben hin aggregiert werden und Entscheidungen nur für sehr große Aufträge oder für ganze Auftragsgruppen weiter oben getroffen werden.

Gewöhnlich müssen für die Balancierungsagenten unterster Ebene andere Strategien verwendet werden wie auf höheren Ebenen. In manchen Ansätzen werden Rechenknoten allein für Lastbalancierungsagenten reserviert. Neben der zusätzlichen algorithmischen Komplexität muß, wie in dezentralen Strukturen allgemein, die zusätzliche Pfadlänge für Migrationen und Entscheidungen mehrerer Agenten entlang des Pfades in Kauf genommen werden.

2.5.3.4 Implizite und explizite Ansätze für dezentrale Strukturen

Wie aus obigen Abschnitten deutlich wurde, hat zentrale Lastbalancierung einige Vorzüge. Für große Rechnersysteme ist aber eine dezentrale Struktur unbedingt notwendig. Die allgemeine dezentrale Struktur besteht aus kooperierenden Lastbalancierungsagenten, die je ein Cluster zentral verwalten. Auf Ebene der Informationsverwaltung und Entscheidungsfindung gibt es zwei Alternativen, um die Balancierung innerhalb eines Clusters und den Lastausgleich zwischen Clustern zu

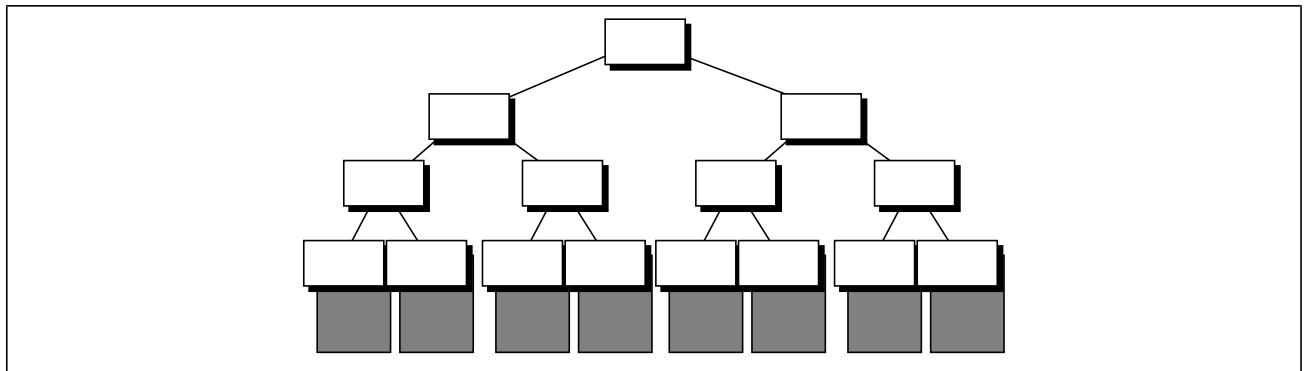


Abbildung 5: Hierarchische Lastbalancierungsstruktur.

koordinieren. Sie sollen in dieser Arbeit mit ‘expliziter’ und ‘impliziter’ dezentraler Lastbalancierung benannt werden.

Im expliziten Ansatz behandelt der Lastbalancierungsagent die Knoten seines Clusters grundsätzlich anders als benachbarte Cluster (seine Sichtweise entspricht Abbildung 4). Die Idee besteht darin, daß jeder Agent sich normalerweise um die Lastsituation und Auftragsverteilung in seinem Cluster kümmert. Von Zeit zu Zeit tauscht er jedoch aggregierte Lastinformationen mit den Nachbaragenten aus. Wenn ein Nachbar-Cluster signifikant weniger belastet ist als das eigene, dann kann er dem Nachbar-Cluster eine Reihe eigener Aufträge abgeben. Lokale Zustandsinformation und lokale Lastbalancierung wird explizit getrennt gegenüber der Zustandsinformation anderer Cluster und der Lastverteilung zwischen Clustern.

Die implizite Technik der dezentralen Balancierung hat im Prinzip nur eine Art von Systeminformationen über Rechenknoten, Aufträge und Cluster und sie verwendet einen einzigen Algorithmus zur Auftragszuweisung bzw. -Migration innerhalb des lokalen Clusters und zwischen Clustern. Ein Nachbar-Cluster wird als weiterer ‘lokal’ Rechenknoten betrachtet, der evtl. etwas unterschiedliche Charakteristik aufweist (die Sichtweise eines Lastbalancierungsagenten ist in Abbildung 6 angedeutet). Dieser „Nachbar-Cluster-Rechenknoten“ hat genau wie lokale Rechenknoten laufende Aufträge und eine Ressourcenbelastung. Die Informationen bekommt der Lastbalancierungsagent in Wirklichkeit als aggregierte Informationen vom Lastbalancierungsagenten des Nachbar-Clusters. Nachbar-Cluster werden genau wie lokale Rechenknoten in Zuweisungs- oder Migrationsentscheidungen ausgewählt.

Der Hauptvorteil impliziter Verteilung ist die Einfachheit. Man benötigt nur eine einzige Lastbalancierungsstrategie und keine getrennten Algorithmen und Informationsgrößen. Es gibt keinen Unterschied zwischen zentralisierter Lastbalancierung kleiner Systeme und einer dezentralen Balancierungsstruktur eines beliebig in Cluster gegliederten großen Systems. Die grundsätzlichen Vorteile zentraler Balancierung können teilweise auf die dezentrale Struktur übertragen werden, je nachdem wie akkurat die

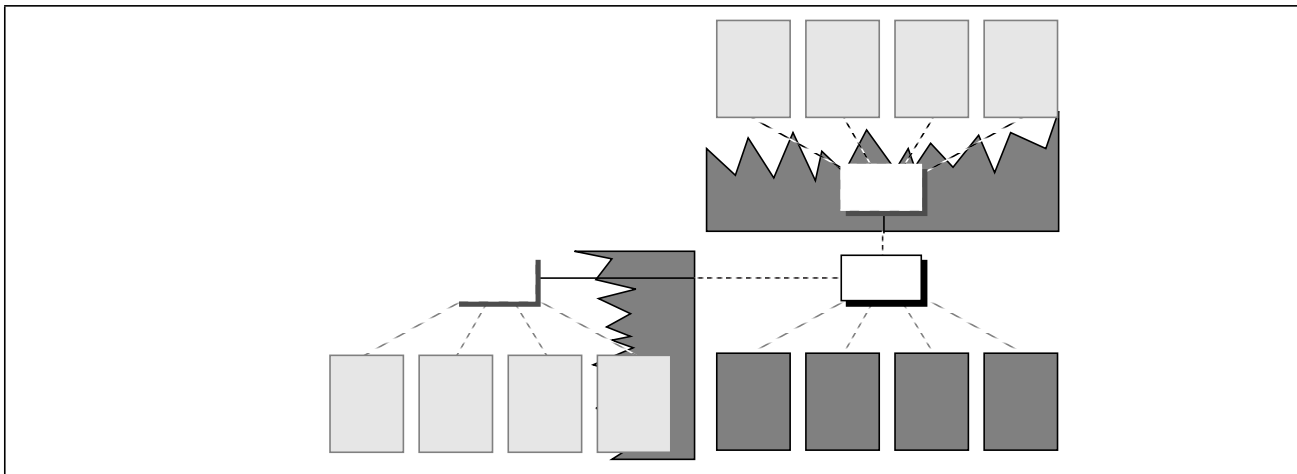


Abbildung 6: Implizit verteilte Lastbalancierungsstruktur.

zwischen Nachbar-Clustern ausgetauschten Informationen sind. Es ist klar, daß man im Extremfall, wenn jeder Lastbalancierungsagent die exakten Informationen über die Rechenknoten und Aufträge der anderen Cluster hat, zwar viele Vorteile, aber auch alle Nachteile der zentralen Balancierung erhält.

Die explizit dezentrale Struktur weist gewöhnlich eine große Lücke auf zwischen einer sehr ausgefeilten, genauen zentralen Strategie, die durch den ‘globalen’ Überblick und komplexe Lastwerte und Profilabschätzungen sehr ausgeglichene Last erzeugt, und der vergleichsweise stupiden, groben dezentralen Strategie zur Kooperation mit Nachbar-Clustern.

Die implizite Verteilung wirft im Detail einige Probleme auf, da sich manche Eigenschaften benachbarter Cluster nicht äquivalent auf Kenngrößen für Rechenknoten reduzieren lassen. Im Rahmen dieser Arbeit wurden beide Ansätze entwickelt und evaluiert.

2.5.4 Statische Ablaufplanung

Das Ziel statischer Ablaufplanung (oft auch „statische Lastbalancierung“ oder „Scheduling“ genannt) ist es, für eine Gruppe fest vorgegebener Aufträge einen Fahrplan zu erstellen, wo und wann die einzelnen Aufträge ausgeführt werden sollen, so daß die Antwortzeit minimal wird. Lastbalancierung hat also hier, wie bereits in Abschnitt 2.4.2 erwähnt, nicht die Aufgabe, den Durchsatz eines endlosen Flusses von Aufträgen zu optimieren, sondern eine endliche Menge von Aufträgen möglichst schnell abzuwickeln. Da es sich hier meist um eine große Anwendung handelt, wird die Antwortzeit durch das Ende des letzten Auftrags bestimmt. Die Ressourcenbedürfnisse der einzelnen Aufträge werden meist als bekannt angenommen und die Rechenknoten stehen allein zur Ausführung dieser Auftragsgruppe zur Verfügung.

Statische Lastbalancierung findet komplett vor der Bearbeitung der Aufträge statt. Sie berechnet für jeden Auftrag einen Startzeitpunkt und einen Rechenknoten, auf dem er bearbeitet werden soll. Bei preemptiven Ablaufmodellen kann sie für jeden Auftrag mehrere Bearbeitungsphasen anordnen. Dann wird der Auftrag suspendiert und später evtl. auf einem anderen Prozessor weiter bearbeitet. Das Betriebssystem versucht, sich an den Ablaufplan der statischen Balancierung zu halten. Wenn die Auftragslaufzeiten unterschätzt wurden, so entstehen Verspätungen oder Phasen, in denen mehrere Aufträge quasi-parallel auf einem Rechenknoten ablaufen. Wurden Auftragsgrößen überschätzt, so entstehen nutzlose Leerlaufzeiten.

Strategien zur statischen Lastbalancierung versuchen, möglichst wenig Leerlaufzeiten auf den Prozessoren zu bekommen und dadurch die Kapazität des parallelen Systems voll auszunutzen. Anspruchsvollere Verfahren, die Rechenknoten mit unterschiedlichen Leistungen einbeziehen, achten darauf, daß vorrangig die schnellen Rechenknoten genutzt werden und langsamere Knoten mit weniger kritischen Aufträgen aufgefüllt werden. Wie kritisch bzw. wichtig ein Auftrag im Rahmen der gesamten Auftragsgruppe ist, kann durch seine Größe oder durch Abhängigkeiten zwischen Aufträgen bestimmt werden (siehe unten). Weiter verfeinerte Verfahren berücksichtigen auch die durch Kommunikation entstehenden Verzögerungen bei der Erstellung des Ablaufplans. Manche Ansätze berücksichtigen weitere Randbedingungen, etwa daß Aufträge nur auf bestimmten Rechenknoten ablaufen dürfen oder daß Aufträge feste Zeitschranken besitzen, bis wann sie fertig bearbeitet sein müssen (solche Bedingungen treten oft in der Echtzeitdatenverarbeitung auf).

Der Rechenaufwand, um einen optimalen statischen Ablaufplan für eine Gruppe von Aufträgen auf einem parallelen und verteilten Rechnersystem zu generieren, wächst im allgemeinen Fall exponentiell mit der Anzahl der Aufträge und der Anzahl der Rechenknoten (als Basis). Es handelt sich um ein NP-hartes Problem. Die Komplexität soll hier nicht nachgewiesen, sondern nur plausibel gemacht werden: allgemein gibt es k^a mögliche Kombinationen, um a Aufträge auf k Rechenknoten zu verteilen (wenn es für $a-1$ Aufträge k^{a-1} Kombinationen gibt, dann kann ein weiterer Auftrag zu jeder dieser Kombinationen auf k verschiedene Knoten gelegt werden; das ergibt $k^{a-1} * k$ Kombinationen). Darunter können r Aufträge, die auf demselben Rechenknoten ablaufen, dort in $r!$ verschiedenen Reihenfolgen ablaufen (ohne Betrachtung quasi-paralleler Abläufe). Die Komplexität kann durch verschiedene - leider meist unrealistische - Einschränkungen reduziert werden. So reduzieren Reihenfolgeabhängigkeiten zwischen Aufträgen die Kombinationsmöglichkeiten, während unter der Annahme gleichartiger Rechenknoten oder homogener Aufträge viele Kombinationsmöglichkeiten bezüglich der Antwortzeit äquivalent sind.

Quasi-statische Ansätze [Blaz88], [Bono88], [Bono90], [Kim92], [Ross91], [Tant85] berechnen statische Wahrscheinlichkeiten, gemäß denen die Knoten zur Laufzeit ankommende Aufträge an andere Knoten abgeben.

Der Ansatz der statischen Lastbalancierung enthält zwei große Probleme: das erste Problem ist, daß die tatsächlichen Ressourcenbedürfnisse der Aufträge in der Realität nicht genau vorhersagbar sind. Oft sind sie genauer absehbar, wenn ein Auftrag tatsächlich ausführbereit wird, was für dynamische Lastbalancierung genutzt werden kann. Der Nutzen statischer Ablaufplanung mit den bisher bekannten Algorithmen geht aber bereits bei geringen Schwankungen der Auftragsgrößen gegenüber den Vorhersagen verloren. Dazu kommt, daß viele Anwendungen auch die Anzahl ihrer Aufträge sowie die Beziehungen zwischen ihnen nicht exakt im voraus angeben können, weil diese zur Laufzeit erzeugt werden. Das zweite Problem liegt darin, daß große parallele Rechnersysteme meist nicht nur für eine Auftragsgruppe alleine reserviert sind, sondern andere Anwendungen konkurrierend ablaufen. In jedem Falle schließen sich unmittelbar davor und danach weitere Anwendungen an. Die Realität stellt daher meist die Anforderung, Aufträge in einen kontinuierlichen Fluß konkurrierender Aktivitäten gut einzupassen.

Statisches Scheduling wird bereits seit langer Zeit entwickelt, wobei die Problemstellungen meist aus eher theoretischem Blickwinkel untersucht wurden. Im folgenden sollen die Grundkonzepte für die vier wesentlichen Problemklassen erläutert werden. Man beachte, daß diese Strategien allesamt Heuristiken sind, die bei akzeptablem Planungsaufwand recht gute Resultate aufweisen. Nachweislich optimale Verfahren mit polynomielltem Aufwand existieren nur für Spezialfälle.

2.5.4.1 Statische Lastbalancierung unabhängiger Aufträge

Für eine Menge unabhängiger Aufträge ist ein preemptiver bzw. nicht-preemptiver Ablaufplan zu erstellen. Das Hauptaugenmerk der Verfahren liegt darin, bis zuletzt möglichst viel Parallelität (und damit Rechenkapazität) im System zu nutzen, und nicht am Ende noch auf einzelne langlaufende Aufträge warten zu müssen. Für nicht-preemptive Ablaufpläne [Li90] wird im Prinzip folgendermaßen verfahren: wähle den größten (noch übrigen) Auftrag und platziere ihn auf dem Ablaufplan, so daß er möglichst früh beendet wird. Bei homogenen Rechenknoten genügt es, den Rechenknoten zu wählen, der bisher als erster wieder frei ist. Bei heterogenen Rechenknoten muß man berücksichtigen, daß die Aufträge auf den Knoten unterschiedlich lange laufen.

Preemptive Ablaufpläne [Blaz86] sind schwieriger zu erzeugen, weil die einzelnen Aufträge zerstückelt werden können. Man schätzt daher zuerst ab, wie groß die Antwortzeit bei einer guten Lastbalancierung ausfallen wird. Nun füllt man den schnell-

sten Rechenknoten bis zu diesem Zeitpunkt mit Aufträgen auf (wie oben beginnend mit dem größten Auftrag), füllt danach den zweitschnellsten Knoten u.s.w., bis alle Aufträge zugewiesen sind. Aufträge, die hinten über die Zeitachse eines Rechenknotens herausragen würden, werden (preemptiv) auf dem nächsten Rechenknoten am Anfang fortgesetzt.

2.5.4.2 Statische Lastbalancierung Reihenfolge-abhängiger Aufträge

Reihenfolgebeziehungen zwischen Aufträgen schränken die Zuweisungsmöglichkeiten ein. Eine effiziente Nutzung des Systems erfordert aber komplexere Planungsalgorithmen. Zunächst wird anhand dreier Beispiele die Problemstellung verdeutlicht, dann werden die beiden wichtigsten Verfahren zur Generierung statischer Ablaufpläne vorgestellt.

Auftragsgraphen sind Mengen von Aufträgen, die durch gerichtete Kanten verbunden sind. Eine Kante vom Auftrag A zum Auftrag B stellt eine Reihenfolgebeziehung dar; sie legt fest, daß zuerst Auftrag A beendet werden muß bevor Auftrag B gestartet werden darf. Auftragsgraphen dürfen selbstverständlich keine Zyklen enthalten. Jeder Auftragsgraph enthält daher Aufträge, die sofort starten können (Startaufträge) und Aufträge, auf die keine weiteren Aufträge mehr warten (Endaufträge). Als Pfad durch einen Auftragsgraphen wird eine Reihe von Aufträgen bezeichnet, die durch Reihenfolgebeziehungen sequentiell nacheinander ablaufen müssen. Die Länge eines Pfades ist die Summe der Auftragsgrößen dieser Aufträge. Als kritisch bezeichnet man die längsten Pfade durch den gesamten Auftragsgraphen, d.h. von einem Startauftrag zu einem Endauftrag.

Die drei Beispiele sollen zeigen, worin die Herausforderung bei der statischen Lastbalancierung Reihenfolge-abhängiger Aufträge liegt und welches Potential an Durchsatzsteigerung vorhanden ist. Die Beispiele sind auf das wesentlichste reduziert; sie verwenden ein System homogener Rechenknoten. Die Abbildungen zeigen jeweils links den Abhängigkeitsgraphen der Aufträge und rechts zwei mögliche Ablaufpläne. Dabei werden keine spezifischen Lastbalancierungsalgorithmen verwendet sondern intuitiv gute Pläne angegeben, die die Reihenfolgebeziehungen zwischen den Aufträgen ignorieren (a) bzw. sie berücksichtigen (b).

Im ersten Szenarium (Abbildung 7) beginnt der kritische Pfad D-E-F nicht mit dem größten Auftrag. Der Ablauf (b) ist schneller, weil er nicht nur die Auftragsgröße als Priorität für die Zuweisung verwendet, sondern die gesamte Pfadlänge vom Auftrag bis zum Endauftrag. Derselbe Effekt tritt ein, wenn anstelle des Auftrages E eine Sequenz zweier kleiner Aufträge E1 und E2 stünde.

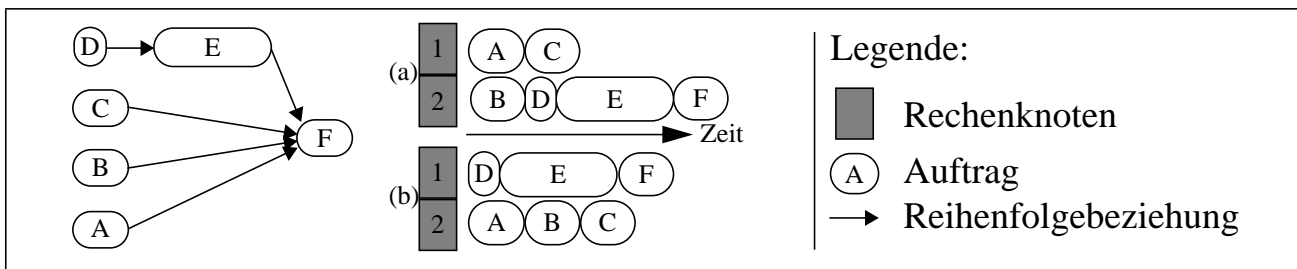


Abbildung 7: Berücksichtigung des kritischen Pfades.

Das zweite Szenarium (Abbildung 8) ist eine ähnliche Konstellation. Der kritische Teil besteht hier nicht nur aus einer sequentiellen Reihe von Aufträgen. Das Problem besteht darin, mehrere kritische Aufträge einzuplanen. Die Ablaufplanung zu (b) erkennt die Aufträge A und C als besonders wichtig und erzeugt so die schnellere Ausführung.

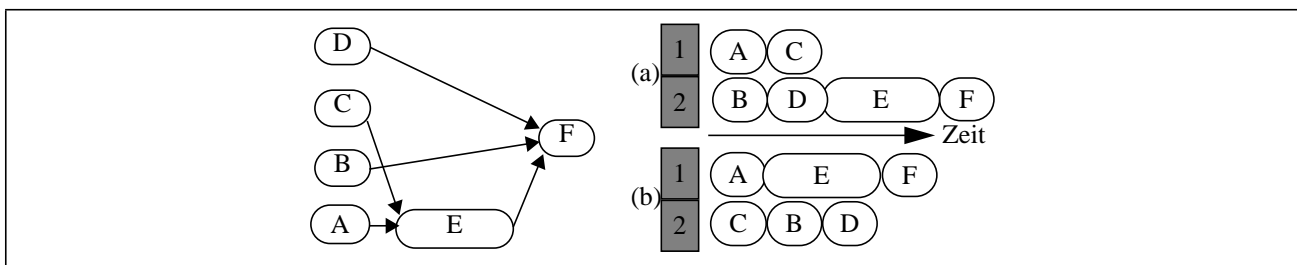


Abbildung 8: Berücksichtigung mehrerer kritischer Pfade.

Das dritte Beispiel (Abbildung 9) enthält überhaupt keinen kritischen Pfad, da alle Pfade durch den Graphen gleichlang sind. Es ist jedoch ein kritischer Teil (DFGHI) erkennbar, der mit höherer Priorität ausgeführt werden sollte, da er insgesamt mehr parallele Rechenleistung erfordert als andere. Diese potentielle Parallelität kann man den linksstehenden Aufträgen noch nicht ansehen, sie entsteht erst in der rechts folgenden Ebene. Der Ablaufplan (b) ist günstiger, weil er den Auftrag D wegen seines hohen Grades an nachfolgender Parallelität bevorzugt.

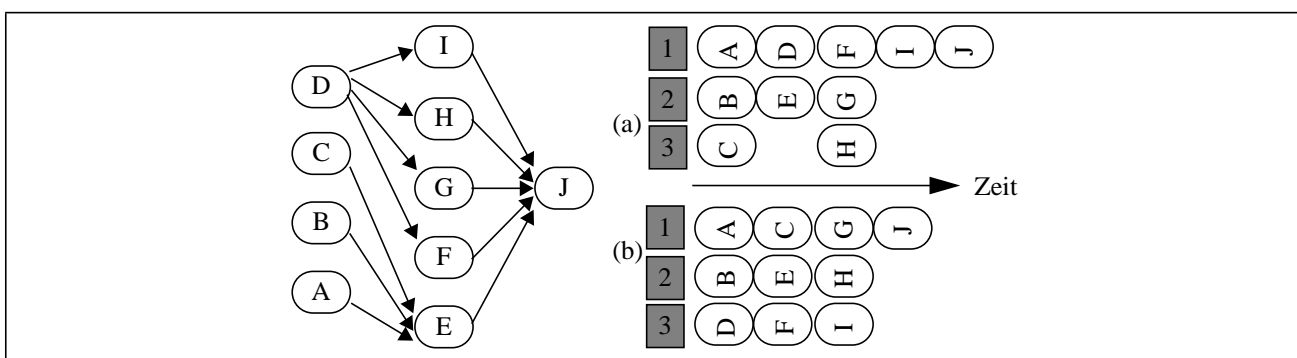


Abbildung 9: Reservierung nachfolgender Parallelität im kritischen Bereich.

Die Beispiele zeigten die Herausforderungen nur für den einfachen Fall homogener Rechenknoten. Die Berücksichtigung von Auftrags-Abhängigkeiten sollte bei Rechenknoten mit unterschiedlichen Leistungen die Aufträge mit höherer Priorität auf schnellere Rechenknoten plazieren.

Nach der Einführung in die Problemstellung sollen die beiden klassischen Verfahren zur statischen Lastbalancierung von Auftragsgruppen unter Berücksichtigung von Reihenfolgebeziehungen vorgestellt werden.

Ein einfaches Verfahren, oft *highest level first scheduling* genannt, teilt den Auftragsgraphen in Ebenen auf [Lam75], [Chan75], [Brun85], [Papa87]. Die erste Ebene enthält die Startaufträge, die zweite Ebene enthält Aufträge, die nur Startaufträge als Vorgänger haben und so fort. Abbildung 10 zeigt links die Ebenen-Einteilung der Aufträge aus obigem zweiten Beispiel. Die statische Lastbalancierung weist zuerst die Aufträge der ersten Ebene zu, dann die Aufträge der zweiten Ebene und so fort. Innerhalb einer Ebene werden die Aufträge der Größe nach priorisiert, was als *heaviest node first* Verfahren bekannt ist.

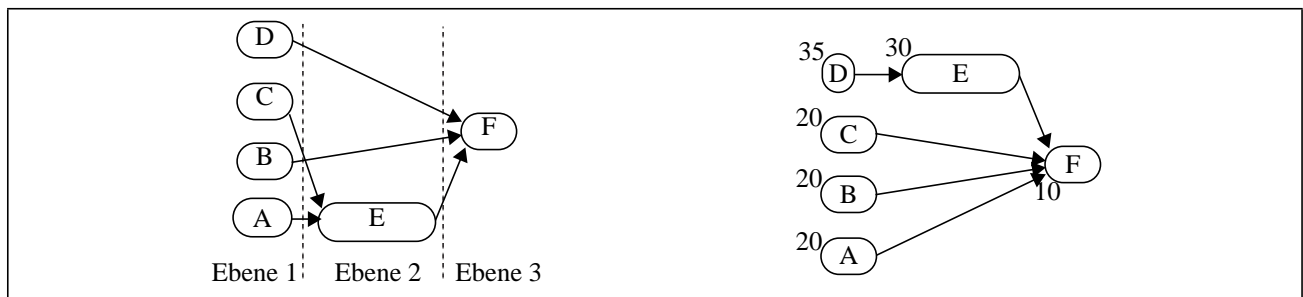


Abbildung 10: Einteilung des Auftragsgraphen in Ebenen (links) bzw. Auftragsprioritäten gemäß der Pfadlänge nachfolgender Aufträge (rechts).

Ein anspruchsvolleres Verfahren, das sogenannte *priority scheduling*, vergibt Prioritäten entsprechend der Pfadlänge der Aufträge zum letzten Endauftrag (exit path). Dahinter steht die Beobachtung, daß die Aufträge entlang des kritischen Pfades sequentiell abzuarbeiten sind und so die Antwortzeit der gesamten Auftragsgruppe bestimmen. Die Prioritäten werden folgendermaßen berechnet: Endaufträge bekommen Prioritäten entsprechend ihrer Auftragsgröße. Danach können die anderen Prioritäten sukzessiv bestimmt werden. Die Priorität eines Auftrags ist seine Auftragsgröße zuzüglich der höchsten Priorität seiner Folgeaufträge. Abbildung 10 zeigt rechts die Prioritätszuweisung an die Aufträge aus obigem ersten Beispiel, wobei Auftragsgrößen von 5, 10 und 20 angenommen wurden. Die Aufträge werden nun nach ihrer Priorität eingeplant: der ausführbereite Auftrag mit der höchsten Priorität wird dem besten Rechenknoten zugewiesen, d.h. dem Knoten, der ihn zum frühesten Zeitpunkt beendet haben wird.

Die Prioritätsberechnung kann um eine gewichtete Pfadlänge erweitert werden. Das beruht auf der Idee, Aufträge, die hohe Parallelität nach sich ziehen, höher zu priorisieren. Man kann zu der wie oben bestimmten Priorität eines Auftrags die Prioritätensumme der Folgeaufträge (mit Ausnahme der größten), dividiert durch die Anzahl der Rechenknoten zuaddieren. Man beachte, daß dies ein im Rahmen der vorliegenden Arbeit entwickeltes Verfahren ist. Alle in der Literatur veröffentlichten Vorschläge normalisieren die Summe, indem sie durch die höchste Priorität dividieren, was jedoch nur sinnvoll ist, wenn die Auftragsgrößen in der Größenordnung von Eins liegen. Die hier gegebene Formel gewichtet gemäß der Relevanz der benötigten Parallelität (das Verhältnis zur verfügbaren Parallelität im System). Abbildung 11 zeigt die Prioritäten der Aufträge aus obigem dritten Beispiel, rechts um die Gewichtungen verfeinert.

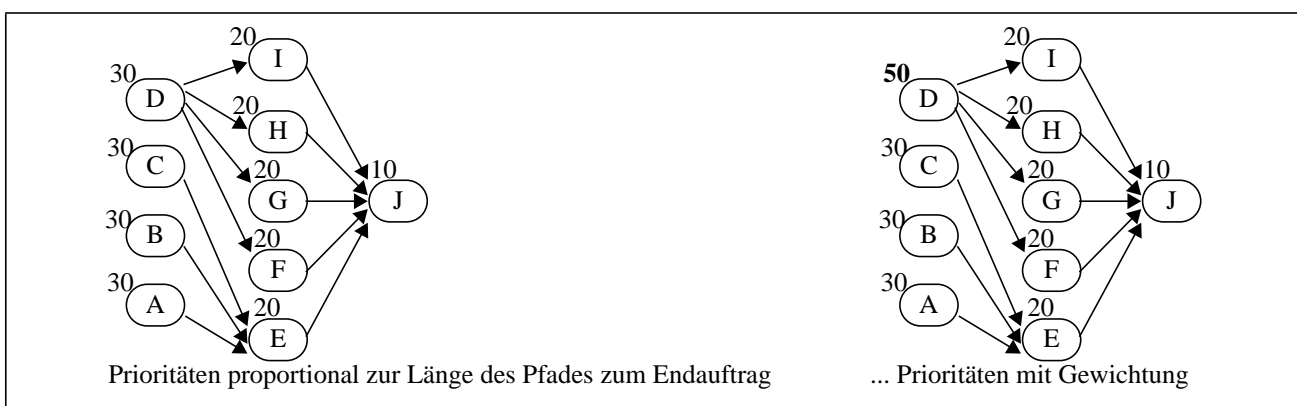


Abbildung 11: Vergleich normaler und gewichteter Berechnung von Prioritäten.

Die Beachtung von Reihenfolge-Abhängigkeiten wird wegen ihres Aufwandes und des genauen Informationsbedarfs fast ausschließlich in statischer Lastbalancierung eingesetzt. In dieser Arbeit wird ein neues Verfahren vorgestellt, das solche Überlegungen sinnvoll in dynamische Lastbalancierung integriert.

2.5.4.3 Statische Lastbalancierung kommunizierender Aufträge

In der Realität laufen die Aufträge einer Anwendung im System nicht völlig entkoppelt ab, sondern arbeiten in bestimmten Formen zusammen. Der Nachrichtenaustausch erzeugt Wartezeiten und belastet die Kanäle des Systems. In nahezu allen Ansätzen zur statischen Lastbalancierung unter Berücksichtigung der Kommunikation zwischen Aufträgen wird angenommen, daß Kommunikation innerhalb eines Rechenknotens keine Verzögerungen und keine Netzlast mit sich bringt, während bei Kommunikation zwischen Aufträgen auf unterschiedlichen Knoten beides signifikant auftritt. Die üblichen Vorgehensweisen, um Kommunikation bei der statischen Ablaufplanung einzubeziehen, hängen vom zugrundeliegenden Ablaufmodell ab und können im wesentlichen in drei Kategorien eingeteilt werden:

Das erste Ablaufmodell ist vor allem auf komplexe parallelisierte Anwendungen zugeschnitten. Eine parallele Anwendung besteht dabei aus einer relativ feststehenden Gruppe von Aufträgen, die zu Beginn gestartet werden und die dann von Zeit zu Zeit untereinander Synchronisationsnachrichten oder Daten austauschen. Zu jedem Paar von Aufträgen wird nun die voraussichtliche Kommunikationsintensität angegeben. Die Aufträge bilden damit einen ungerichteten Graphen, dessen Kantengewichte die Kommunikationsintensitäten zwischen den Auftragsknoten beziffern. Das parallele Rechnersystem wird ebenfalls als ungerichteter Graph angesehen, wobei die Kommunikationskanäle als Kanten mit ihrem Durchsatz bzw. ihrer Nachrichtenverzögerung gewichtet sind. Ob der Durchsatz oder die Verzögerungszeit (Abschnitt 2.4.5) relevant sind, hängt davon ab, ob sehr lange oder viele kurze Nachrichten ausgetauscht werden. Der Ablaufplan wird nun entweder so erstellt, daß auf allen Kanälen in der Summe die Durchsatzanforderungen nicht überschritten werden, oder die Gesamtsumme aller Kommunikationswartezeiten durch Nachrichtenaustausch zwischen Rechenknoten minimiert wird. Die eigentliche Verteilung der Aufträge an die Rechenknoten - um die reine Rechenzeit zu minimieren (vergleiche Abschnitt 2.5.4.1) - ist hier vereinfacht, da ja alle Aufträge zugleich am Anfang auf dem System gestartet werden: man muß nur die Gesamtanforderungen der Aufträge gleichmäßig auf die Rechenknoten verteilen. Die Ablaufplanung wird üblicherweise durch rekursive Aufspaltung des Auftragsgraphen [Bokh81], [Bowe88], [Ma82] oder sogenannte *graph matching* [Lo88], [Shen85], *integer linear programming* oder ähnliches [Bowe92], [Stra88], [Lee92] oder genetische [Kane91] Algorithmen realisiert.

Das zweite Ablaufmodell bezieht sich auf Datenfluß-orientierte Strukturen oder auch Pipeline-Verarbeitung. Jeder Auftrag bekommt beim Start Zwischenergebnisse seiner Vorgänger-Aufträge als Eingabedaten und berechnet daraus weitere (Zwischen-) Ergebnisse, die evtl. von Folgeaufträgen weiterverarbeitet werden. Das Grundprinzip ist hier, jeden Folgeauftrag möglichst auf denselben Rechenknoten zu legen wie seine Vorgängeraufträge, um die Kommunikationswartezeit beim Auftragsstart zu minimieren. Zumindest sollte der Rechenknoten für den Folgeauftrag schnelle Netzverbindungen zu den Rechenknoten der Vorgänger besitzen. Die hier verwendeten Algorithmen zur Generierung von Ablaufplänen sind meist Erweiterungen von in Abschnitt 2.5.4.2 vorgestellten Verfahren, die Reihenfolgebeziehungen zwischen Aufträgen einbeziehen [Ange90], [Chou82], [Coli91], [Indu86], [Lewi93]. Die durch Weitergabe der Zwischenergebnisse entstehenden Verzögerungen werden bei der Auswahl des Rechenknotens für einen Auftrag zur reinen Berechnungsdauer auf diesem Rechenknoten zuaddiert.

Das dritte Ablaufmodell betrachtet Aufträge, die auf gemeinsamen Daten operieren. Das können kooperierende Aufträge innerhalb einer komplexen parallelen Anwendung sein oder auch unabhängige Aufträge, die auf globalen Daten (etwa Dateien oder

Objekten einer Datenbank) arbeiten. Hier ist das Bestreben der statischen Lastbalancierung, die Aufträge dort zu platzieren, wo der größte Teil der benötigten gemeinsamen Daten liegt [Bara85], [Yu86], [Yu91]. Als zusätzlicher Faktor ist neben der Rechenleistung und Belastung der Rechenknoten auch die Summe der Wartezeiten auf Datensätze, die nicht lokal auf dem Rechenknoten verfügbar sind, bestimmend für die Laufzeit eines Auftrags. In Ablaufmodellen, die keine Migration oder Replikation gemeinsamer Daten zulassen (Abschnitte 2.4.6 und 2.4.8), muß anstelle eines direkten Zugriffs auf nicht-lokale Datensätze eine Zugriffsanforderung an den Besitzerknoten der Daten gesandt werden, der dort die Zugriffsoperation ausführt. Die Berücksichtigung von Datenaffinitäten in statischer Lastbalancierung setzt natürlich voraus, daß die Lokationen der Datensätze statisch sind.

2.5.5 Dynamische Ablaufregelung

Zu Beginn des vorigen Abschnitts wurde bereits das Problem der statischen Lastbalancierung angesprochen, daß nämlich die Auftragslast nicht oder nicht genügend genau im voraus abgeschätzt werden kann. Außerdem ist gewöhnlich ein stetiger Fluß von kooperierenden und unabhängigen Aufträgen zu balancieren und nicht eine fixe endliche Menge von Aufträgen. Dynamische Lastbalancierung soll die Durchsatzoptimierung während der Laufzeit des Systems durchführen und dabei auf die tatsächlich auftretenden Lastsituationen reagieren.

Die Grundidee dynamischer Lastbalancierung ist es, die Auslastung der Rechenknoten während der Laufzeit periodisch zu messen. Neue Aufträge werden möglichst auf momentan relativ gering belastete Knoten geschickt [Ezza86], [Graf93], [Grim91], [Hac86], [Hsu86], [Kuch90], [Mutk87], [Osse92], [Zhou87]. Bei starken Lastdifferenzen der Knoten werden - in preemptiven Ablaufmodellen - laufende Aufträge von überlasteten zu gering belasteten Rechenknoten migriert [Bemm90], [Doug91], [Eage86], [Krem92], [Lüli91], [Mirc89]. Dies ist eine globale Sichtweise. Da dynamische Lastbalancierung häufig dezentral strukturiert ist, kann man das Grundkonzept auch folgendermaßen beschreiben: jeder Rechenknoten tauscht seine momentane Lastsituation periodisch mit einigen Nachbarknoten aus. Wenn er starke Lastdifferenzen feststellt, so verschiebt er einige seiner Aufträge an die minderbelasteten Knoten bzw. fordert von höherbelasteten Aufträge an. Wenn bei ihm neue Aufträge entstehen, so entscheidet er, ob er sie lokal startet oder sofort an einen momentan minderbelasteten Nachbarknoten abgibt.

Der Ansatz der dynamischen Lastbalancierung wird noch nicht so lange verfolgt wie die statische Lastbalancierung. Bisher wurden fast ausschließlich sehr simple, dezentral strukturierte Verfahren veröffentlicht, die obige Grundidee mit sehr einfachen Auftragsmodellen und Lastkenngrößen realisieren.

Häufig liegt der Forschungsschwerpunkt noch in der Implementierung von preemptiven Ablaufmodellen, die in heutigen Betriebssystemen nicht adäquat unterstützt werden. In den heute verfügbaren, monolithischen Betriebssystemen ist Prozeßmigration technisch schwer zu realisieren, da Zugriffe auf lokale Ressourcen oder Speicherstrukturen wie etwa Bildschirmspeicher auf anderen Prozessoren nicht dieselben Auswirkungen zeigen. In heterogenen Systemen wird die allgemeine Prozeßmigration noch längere Zeit unmöglich sein. Stattdessen müßten Anwendungen einen sogenannten Checkpoint / Restart Mechanismus unterstützen, um von der Lastbalancierung migriert werden zu können, d.h. sie speichern ihren Bearbeitungszustand, so daß sie auf einem anderen Prozessor dort fortfahren können. Weitere Probleme des preemptiven Ansatzes sind die Einschätzung der restlichen Auftragslaufzeit um zu entscheiden, ob sich eine Migration noch lohnt, die hohen Migrationskosten, und die Gefahr der unnötig häufigen Hin- und Rückmigration von Aufträgen.

Insgesamt bestehen weiterhin noch ungeklärte Fragen zum Konzept der dynamischen Lastbalancierung:

- Ist der vollständig dezentrale Ansatz im allgemeinen Fall der einzig erfolversprechende [Thei88], [Zhou92]?
- Ist es überhaupt gewinnbringend, komplexere Strategien einzusetzen [Efe89], [Ferg88], [Ferr86], [Smit80]?
- Wieviel Potential zur Durchsatzsteigerung enthält der Ansatz der Migration laufender Aufträge [Eage88], [Krue88]?

Wie im Gebiet der statischen Lastbalancierung findet man auch in dynamischen Ansätzen vorwiegend unrealistische Rechenmodelle oder sehr simple, anwendungsspezifische Implementierungen. In der vorliegenden Arbeit wird ein Konzept zur dynamischen Lastbalancierung vorgestellt, das sehr ausgereifte, komplexe Strategien enthält, für ein breites Anwendungsfeld einsetzbar ist, realisiert und durch Messungen validiert wurde. Es gibt derzeit keine anderen Konzepte von vergleichbarer Mächtigkeit und Flexibilität.

Der Ansatz der dynamischen Lastbalancierung bringt einige Grundprobleme mit sich. Da Lastbalancierung als Teil oder Erweiterung des (verteilten) Betriebssystems selbst Rechenzeit verbraucht, muß der Aufwand für die Lastbalancierungsaufgabe möglichst gering gehalten werden. Die für Lastbalancierung verbrauchte Rechenleistung mindert ja Gesamtdurchsatz der Anwendungen im System. Es muß also ein Kompromiß zwischen dem Aufwand und dem Nutzen gefunden werden.

Ein analoges Problem sind die Verzögerungen, die durch die Lastbalancierung verursacht werden. Wenn ein neu entstandener Auftrag lange Zeit in Entscheidungsalgorithm-

men und Warteschlangen zur Verzögerung der Zuweisungsentscheidung zubringt, so verlängert das die Antwortzeit, weil er nicht unverzüglich bearbeitet wird. Auch die Zeitdauer für eine Auftragsmigration verlängert die Antwortzeit des Auftrags. Man beachte dabei, daß in dieser Arbeit der Schwerpunkt der Lastbalancierung auf die Durchsatzsteigerung und weniger auf die Minimierung einzelner Antwortzeiten gelegt wird. Daher sind Wartezeiten für Aufträge kein Verlust, solange die Rechenkapazitäten des Systems anderweitig ausgenutzt werden können, und auch in Zukunft noch genutzt werden können, denn der Auftrag kann ja eine große Menge evtl. paralleler Folgeaufträge haben, die auch erst später starten können, wenn der Auftrag verzögert wird (Abschnitt 2.5.4.2).

Das dritte Problem des dynamischen Ansatzes ist die mangelnde, ungenaue und veraltete Information. Die aktuelle Auslastung eines Rechenknotens beruht auf Messungen über kurze Zeitintervalle. Meist wird eine exponentielle Glättung eingesetzt (frühere Meßwerte werden mit geringerer Gewichtung einbezogen), um kurzfristige Schwankungen abzuschwächen. Die aktuelle Auslastung beschreibt daher lediglich die Lastsituation der Vergangenheit. Die Auftragszuweisungen und -Migrationen auf Basis dieser Information extrapolieren somit den Lastzustand für die nahe Zukunft mit einer Ansatzfunktion Null-ten Grades, d.h. sie nehmen an, daß er konstant bleibt. Relevant für Balancierungsentscheidungen ist aber die zukünftige Lastsituation, wie sie ja auch in statischen Verfahren (Abschnitt 2.5.4) eingesetzt wird. Vorabinformationen über die entstehende Auftragslast wird in den meisten Ansätzen gar nicht eingesetzt. Das bedeutet aber, daß die dynamische Lastbalancierung einerseits stets annimmt, daß das System stabil ist und keine weiteren aufträge entstehen werden, andererseits alle Aufträge als homogen (d.h. mit identischen Ressourcenanforderungen) betrachtet. Für die Abschätzung der verbleibenden Laufzeit eines laufenden Auftrags (für die Überlegung, ob sich eine Migration noch lohnt), wird meist eine exponentialverteilte Laufzeit angenommen, d.h. man vermutet, daß „der Auftrag nochmal solange läuft wie bisher“. Reihenfolgebeziehungen oder Kommunikation zwischen Aufträgen wird ebenso selten im voraus einbezogen [John93], [Winc92]. Entstehender Kommunikationsaufwand wird allenfalls gemessen, woraufhin kommunizierende Aufträge evtl. auf denselben Rechenknoten verlagert werden.

Dynamische Lastbalancierung kann auch die Datenverteilung im System so regeln, daß die Aufträge möglichst viel auf lokale Daten zugreifen [Vara88]. Doch auch hier gilt, wie eben bei der dynamischen Berücksichtigung von Kommunikation und Reihenfolgebeziehungen erwähnt, daß solche Strategien auf Vorabschätzungen basieren. Die meisten dynamischen Verfahren sind bislang nur reaktiv und nutzen kein Vorwissen über Aufträge oder Auftragsgruppen.

Rein dynamische Lastbalancierung hat durch die drei Grundprobleme im Vergleich zu statischen Lastbalancierungsansätzen viel weniger Möglichkeiten, den Systemdurchsatz zu verbessern. Dafür ist sie in der Lage, auf unerwartete Lastsituationen, die katastrophales Systemverhalten zur Folge haben, zu reagieren und wieder einigermaßen günstige Lastverteilung einzustellen. Es gibt bereits Ansätze, die dynamische Lastbalancierung durch vorgelagerte statische Planungsalgorithmen verbessern [Iqba86]. In dieser Arbeit wird ein Ansatz vorgestellt, der Auftragsvorabschätzungen (wie Auftragsgröße oder Datenreferenzmuster) und vermutete Beziehungen zwischen Aufträgen in ein dynamisches Lastbalancierungsverfahren integriert.

2.5.6 Adaptive Ablaufregelung

Das Ziel der adaptiven Lastbalancierung ist die weitere Flexibilisierung und Anpassungsfähigkeit der dynamischen Lastbalancierung. Die Begriffe *dynamische* und *adaptive* Lastbalancierung sind in der Literatur noch nicht klar getrennt und es gibt keinen prinzipiellen Unterschied, denn beide Ansätze verwenden zur Laufzeit ermittelte Werte für Entscheidungen. Ein Ansatz zur Adaption besteht darin, dynamisch zwischen Strategien zu wechseln. Bekannte Arbeiten unterscheiden leichte, mittlere und hohe Systemlast und schalten jeweils auf eine dafür geeignete Strategie um [Eage86], [Eage88]. Andere Ansätze zur Adaption lernen aus früherem Verhalten oder den Ergebnissen früherer Entscheidungen, d.h. setzen frühere Meßwerte im Entscheidungsprozeß ein. Typischerweise werden Entscheidungsparameter aufgrund längerfristiger Beobachtungen geregelt. Beispielsweise können die beobachteten Ressourcenbedürfnisse von Aufträgen oder Anwendungen protokolliert und als Vorabschätzungen für die spätere Zuweisung ähnlicher Aufträge verwendet werden. Da dynamische Lastbalancierung zur Laufzeit durchgeführt wird, sind aufwendigere Lernverfahren nicht tragbar. Diese generellen Ansätze finden sich in der Literatur in verschiedensten Formen, motiviert durch verschiedene Mängel in Balancierungsstrategien, die sich in drei Problemfelder einteilen lassen:

1. Adaptive Anpassung und Korrektur der Vorabschätzungen für Auftragsprofile und das Systemlastverhalten

Dynamische Lastbalancierung entscheidet aufgrund von Informationen über Anwendungsanforderungen und der Systemauslastung, die oft ungenau oder nicht verfügbar sind. Auch wenn oft explizit keine Annahmen über das künftige Auftrags- oder Systemverhalten gemacht werden, sondern nur auf aktuelle Messungen reagiert wird, liegt implizit eine Extrapolation nullten oder ersten Grades zugrunde. Weiter ausgebauten Ansätze können Lastprofilangaben einsetzen, die aus eigenen Beobachtungen oder von den Anwendungen stammen [Deva89], [Gosw93], [Kunz91], [Osse92], [Rahm86]. Durch Vergleich der früheren Vorhersagen mit dem tatsächlichen Systemverhalten kann Lastbalancierung adaptiv die Genauigkeit und den Wert

der Vorabinformationen beurteilen und sie entsprechend korrigieren. Beispielsweise wird in [Yu86], [Yu91] periodisch durch Regressionsanalyse die Korrelation zwischen den angenommenen und den beobachteten Antwortzeiten ermittelt und die Abweichungen durch Korrekturfaktoren für weitere Routing-Entscheidungen verringert.

2. Regelung schwieriger Entscheidungsgrößen durch Rückkopplung

Eine weitere Schwäche dynamischer Balancierungsverfahren ist das ungenaue Ausführungsmodell, auf dem der Entscheidungsalgorithmus basiert. In realen parallelen und verteilten Systemen und großen, realen Anwendungen ist es äußerst schwierig, alle Effekte und Abhängigkeiten in einem kompakten Modell zu erfassen. Weil Entscheidungen zur Laufzeit getroffen werden, müssen sie sich auf simple Rechenmodelle beschränken. Somit ist die Korrelation zwischen den Größen, die die Lastbalancierung einbezieht (z.B. die *run queue length* der Prozessoren) und dem Gesamtdurchsatz, der optimiert werden soll, nicht in allen Situationen stark genug. Beispielsweise hängen die Kosten für entfernte Datenzugriffe von der Größe und Komplexität der Daten, von der Auslastung des Netzwerks, von Sperrwartezeiten und von der Auslastung des Datenbesitzers ab, was sich schlecht kompakt ausdrücken läßt.

3. Adaptive Optimierung des Kosten - Nutzen Verhältnisses der Lastbalancierung

Der Nutzen dynamischer Lastbalancierung wird durch ihren Aufwand geschmälert., da sie zur Informationssammlung und Entscheidungsfindung Kommunikations-, Rechenlast und Verzögerungen erzeugt. Ohne Adaption nimmt Lastbalancierung an, daß ihr Aufwand stets angemessen ist, was nicht für alle Situationen und Lastprofile zutrifft. Lastbalancierung sollte also ihren Aufwand minimieren oder sogar das Kosten - Nutzen Verhältnis beobachten und regeln [Sale90]. Beispielsweise sollte dezentrale Lastbalancierung in Situationen global hoher Systemlast weniger Lastinformationen austauschen, weil wenig migriert werden sollte und sich die Zusatzkommunikationslast nicht lohnt, und sie sollte anstatt aufwendiger Sender-Initiierung auf Empfänger-Initiierung wechseln, weil die Suche nach gering belasteten Knoten aufwendig und wenig aussichtsreich ist.

2.5.7 Optimierungskriterien für dynamische Lastbalancierung

Um die Vorstellung der Konzepte im Bereich der Lastbalancierung abzuschließen, sollen die Ansätze für dynamische Lastverteilung auf ihre Ziele hin unterschieden werden. Das durch Scheduling oder Lastbalancierung mögliche Optimierungspotential sowie die notwendige Komplexität hängt von dem zugrundeliegenden Optimierungskriterium ab. Wie in Abschnitt 2.5.5 erwähnt, ist dynamische Lastbalancierung für den Dauerbetrieb ausgelegt. Sie versucht nicht, eine feste endliche Menge von Aufträgen in insgesamt kürzester Zeit abzuarbeiten, sondern den Systemdurchsatz bei der Abarbei-

tung eines unendlichen Stroms von Aufträgen zu maximieren. Dieses Ziel kann jedoch nicht unmittelbar als Optimierungskriterium in Lastbalancierungsstrategien, d.h. in Form eines Entscheidungsalgorithmus, realisiert werden.

Im Folgenden werden vier Optimierungskriterien herausgearbeitet, die in dynamischen Verfahren verwendet werden. Dabei steigt das theoretische Optimierungspotential mit dem Entscheidungsaufwand ebenso wie die erforderliche Anzahl und Güte der Vorab- und Mess-Informationen über das System- und Anwendungsverhalten vom ersten bis zum dritten Kriterium an; die Komplexität sinkt wieder beim vierten Kriterium.

1. Vermeidung arbeitsloser Rechenknoten.

Dieses sehr häufig verwendete Kriterium beruht auf der Überlegung, daß unbeschäftigte Rechenknoten im System den maximal möglichen Durchsatz verringern. Lastbalancierung versucht daher, unbeschäftigte Knoten durch neue Aufträge oder durch Verlagerung von Aufträgen überlasteter Knoten wieder zu nutzen. Diese Technik kann ohne Vorwissen über Auftragsverhalten auskommen, enthält aber die impliziten Annahmen homogener Rechenkapazitäten der Knoten, homogener Auftragsprofile und völliger Isolation und Ortsunabhängigkeit (z.B. bzgl. Datenzugriffen) der Aufträge.

2. Minimale Antwortzeit für Einzelaufträge.

Dieses Optimierungskriterium wird angestrebt, indem die Lastbalancierung jeden Auftrag dem Rechenknoten zuweist, der ihn am Schnellsten abzuarbeiten verspricht. Wenn Auftragsmigration möglich ist, können außerdem laufende Aufträge auf Knoten verlagert werden, wenn sie dort aufgrund neuer Informationen schneller fertiggestellt werden können. Solche Verfahren sind in der Lage, unterschiedlich leistungsfähige Rechenknoten zu berücksichtigen, da es vorteilhaft sein kann, stärkere Rechenknoten höher zu belasten und sogar schwache Knoten unbeschäftigt zu lassen. Weiterhin können Vorabschätzungen über Auftragsprofile berücksichtigt werden, um die Antwortzeiten der Aufträge zu erwägen. Das Kriterium ermöglicht es weiterhin, Orts- und Kommunikationsabhängigkeiten von und zwischen Aufträgen für die Abschätzung von Antwortzeiten auf bestimmten Knoten einzubeziehen. Verfahren, die dieses Kriterium anstreben, sind bisher selten [Yu86].

3. Minimale Gesamtlaufzeit des Systems.

Das Kriterium entspricht dem der statischen Lastbalancierung, denn es wird angenommen, daß außer den momentan bekannten, d.h. den angekündigten, den ausführbaren und den bereits zugewiesenen Aufträgen keine weiteren Aufträge mehr kommen. Lastbalancierung weist daher Aufträge so den Rechenknoten zu, daß die Gesamtlaufzeit, d.h. die Zeit, bis der letzte Auftrag beendet ist, möglichst klein wird. Wichtige Grundprinzipien sind dabei die Vermeidung von Lücken in der Auslastung der Knoten, die Berücksichtigung von Reihenfolgebeziehungen und des Kommunikationsaufwands zwischen Aufträgen, die Beachtung unterschiedlicher Antwortzei-

ten auf verschiedenen schnellen Rechenknoten und vor allem die volle Nutzung aller Knoten bis zum Ende aller Bearbeitungen. Letzteres ist wichtig zur Vermeidung einer langen Auslaufphase, in der nur noch wenige Rechenknoten beschäftigt sind. Dynamische Lastbalancierungsansätze mit diesem Optimierungsziel sind noch äußerst rar [Chow79].

4. Maximaler Gesamtdurchsatz des Systems bis zum Eintreffen des nächsten Auftrags. Da dynamische Lastbalancierung eigentlich für den Dauerbetrieb ausgelegt ist, bleibt das im dritten Punkt beschriebene Kriterium suboptimal. Die Schwierigkeit besteht darin, daß die Lastbalancierung um das Eintreffen weiterer Aufträge weiß, aber nicht beliebig weit im voraus ahnt, wieviele, wo, wann und welche Aufträge folgen ankommen werden. Unter diesen Voraussetzungen ist es am besten, nur für den Zeitraum bis zum (vermutlichen) Eintreffen weiterer Aufträge das System optimal zu nutzen. Die Planung für die weitere Zukunft wird erst nach dem Eintreffen weiterer Aufträge unter den dadurch geänderten Umständen vorgenommen. Zusätzlich zu den für das dritte Kriterium erforderlichen Informationen ist hier eine Abschätzung notwendig, wie lange nach Eintreffen eines Auftrags oder Auftragschubs mit einer weiteren Auftragsankunft zu rechnen ist. Die Zuweisungs- bzw. Migrationsstrategie wird weniger aufwendig, da sie nur begrenzte Zeit vorausblicken muß, was bei der Ungenauigkeit und Unsicherheit der Informationen in dynamischer Lastbalancierung ohnehin ratsam ist.

Wie beim dritten Punkt besteht auch hier die Optimierungsmöglichkeit, längere Antwortzeiten einzelner Aufträge zugunsten einer gleichmäßigen Vollauslastung der Ressourcen in Kauf zu nehmen und die Möglichkeit, die Bearbeitung unkritischer Aufträge zeitlich nach hinten zu verschieben. Gegenüber dem dritten Punkt besteht hier außerdem die Möglichkeit, die Einplanung von über das Zeitintervall hinausgehenden Bearbeitungen noch aufzuschieben; möglichst späte Entscheidungen sind wichtig im Bereich der dynamischen Lastbalancierung. Die Lastbalancierung kann hier ein ‚ausfransen‘ der Bearbeitungen auf den Prozessoren hinter dem Zeitintervall in Kauf nehmen, denn bis das Intervall verstrichen ist, sind weitere Aufträge zur Verteilung eingetroffen oder die momentan noch nicht zugewiesenen Aufträge können dann geeignet verteilt werden.

Allgemein sind Durchsatzmaximierung und Antwortzeitminimierung orthogonale Kriterien. Minimierung einzelner Antwortzeiten ist wichtig, wenn das System nahezu unbelastet ist, oder wenn die Antwortzeit eines Auftrags kritisch ist, weil ein Benutzer darauf wartet oder weitere Aufträge innerhalb einer Anwendung von diesem Auftrag abhängen. In stärker belasteten Systemen, wo meist mehrere ausführbare Aufträge zur Verfügung stehen, die Ressourcen konkurrierend nutzen können, ist es global wichtig, den Gesamtdurchsatz des Systems zu erhöhen. Der Durchsatz ist die Summe aller sinnvoll verwendeten Ressourcen des Systems, wobei meist die nur die Summe der ver-

brauchten Prozessorzeiten betrachtet wird. Somit kann man einen maximalen Durchsatz schon erreichen, indem man lediglich Prozessorleerlaufzeiten vermeidet (1. Kriterium). Daß Aufträge auf stark belasteten oder langsameren Knoten deutlich länger laufen, ändert am Gesamtdurchsatz nichts. Unausgewogene Lastverteilung verschlechtert jedoch den Systemdurchsatz, wenn durch lange Laufzeit eines Auftrags Leerlaufzeiten entstehen, weil abhängige Aufträge nicht rechtzeitig starten können, oder wenn durch Datenkommunikation Leerlaufzeiten in Form synchronen Wartens entstehen.

Das in der vorliegenden Arbeit entwickelte Verfahren zur dynamischen Lastbalancierung basiert auf dem vierten Optimierungskriterium. Bisher sind noch keine vergleichbar allgemeinen, dynamischen Ansätze dieser Klasse veröffentlicht worden.

3 Das *HiCon* Konzept zur dynamischen Lastbalancierung

In diesem Kapitel werden die Konzepte vorgestellt, die im Rahmen der vorliegenden Arbeit entwickelt wurden. Dabei werden die Begriffe und Grundlagen aus Kapitel 2 verwendet. *HiCon* ist ein älteres Projekt-Akronym mit der Bedeutung *Hierarchical Controlled Network computing*.

3.1 Motivation

Basierend auf einem realistischen parallelen Rechnersystem soll durch automatische Lastbalancierung der Ablauf verschiedener großer Anwendungen optimiert werden.

Dabei sollen realistische Rechnerstrukturen unterstützt werden: Die Rechenkapazitäten in der Industrie bestehen zunehmend aus einer Menge vernetzter Workstations und wenigen Parallelrechnern. Die Rechenknoten sind dabei relativ lose gekoppelt, d.h. einer sehr hohen Prozessor-Rechenleistung steht ein langsames Netzwerk gegenüber.

Weiterhin sollen im *HiCon*-Modell realistische Anwendungsprofile unterstützt werden: Die Rechenlast auf diesem System setzt sich zum Großteil aus komplexen, teilweise bereits parallelisierten Anwendungen und einigen kleineren Anwendungen zusammen. Große Anwendungen waren bisher wissenschaftliche und kommerzielle Produktionsläufe, während kleinere Aufträge meist interaktive Kommunikations- und Entwicklungsvorgänge der Benutzer waren. Neuere Softwaretechnologie bewirkt eine Vermischung der Anwendungsklassen: graphische Aufbereitung und Benutzeroberflächen konsumieren viel Rechenleistung, die Funktionalität größerer Anwendungspakete wird auf das Rechnernetz verteilt und erzeugt nichttriviale Lastprofile; schließlich basieren verschiedene größere Anwendungen zunehmend auf Datenverwaltungssystemen, auf die sowohl konkurrierende interaktive Zugriffe als auch langlaufende Aufträge abgebildet werden.

Betriebssystem-Plattformen befinden sich auf dem Wege zu einer einheitlichen Funktionalität gegenüber den Anwendungen. Das ermöglicht es, Anwendungen mit geringem Aufwand für verschiedene Rechnertypen zur Verfügung zu stellen und große Anwendungen sogar auf heterogene Systeme zu verteilen und zu parallelisieren. Diese Möglichkeit zusammen mit den stark gemischten und wechselnden Lastprofilen benötigen eine automatische Lastbalancierung, die solche Lasten dynamisch geeignet auf heterogene Rechnersysteme verteilt. Dabei sind nicht nur die reinen Rechenzeitanforderungen zu betrachten, sondern auch die Kommunikation in parallelen Anwendungen und die Arbeit auf gemeinsamen Datenbasen. Wie unten hervorgehoben wird, ist es nicht leicht, verschiedene komplexe Anwendungen gut zu balancieren. Heute verfü-

bare Ansätze stellen nur einen Anfang dar. Beliebige starke Verteilung paralleler Aufträge innerhalb einer Anwendung bringt keine optimale Effizienz, sondern ein vernünftiges Maß muß gefunden werden. Sehr verschiedene Lastprofile und Granulate können nicht mit einem einfachen Verfahren, erst recht nicht gleichzeitig zusammen balanciert werden. Bestehende Ansätze sind in bezug auf verschiedenartige Anwendungstypen, Rechnersysteme und Lastsituationen noch recht unflexibel. Das *HiCon*-Konzept soll durch drei Ansätze flexible Lastbalancierung ermöglichen:

- Die Balancierungsstrategie berücksichtigt verschiedene Informationstypen über Anwendungen und Rechnersystem, die sowohl Vorabschätzungen als auch gemessene Werte enthalten. Beispiele für Laufzeitinformationen sind die Prozessorauslastung, Hauptspeicherausnutzung, die Datenverteilung im System, Auftragsgrößen, Datenreferenzprofile und Reihenfolgebeziehungen zwischen Aufträgen.
- Lastbalancierung kann durch eine skalierbare, gemischt zentrale und dezentrale Struktur auf beliebige Systemkonfigurationen angepaßt werden, und kann vor allem die Vorteile zentraler Verfahren nutzen. Innerhalb von Clustern kann die Last zentral durch komplexe Strategien koordiniert werden, während zwischen benachbarten Clustern ein grober Lastausgleich erfolgt.
- Die Lastbalancierung ist in der Lage, ihre Entscheidungsparameter durch Beobachtung des tatsächlichen Systemverhaltens dynamisch anzupassen. So können etwa die tatsächlichen Datenkommunikationskosten gemessen und für weitere Entscheidungen verwendet werden; Der Lastbalancierungsaufwand kann durch Wahl einer geeigneten Strategie in sinnvollem Rahmen gehalten werden.

Das Konzept ist auf datenintensive Anwendungen ausgerichtet. Üblicherweise werden die Anwendungen aus dem Bereich der Datenverwaltung als datenintensiv bezeichnet. Allgemein sind jedoch solche Auftragsarten datenintensiv, deren Anteil an reinen Datenzugriffen im Verhältnis zur Rechenarbeit signifikant ist. Im *HiCon*-Modell können natürlich auch rechenintensive Anwendungen berücksichtigt werden, da sie einfacher zu balancieren sind. Das Ablaufmodell kennt explizit globale, flüchtige und persistente Datensätze, auf denen Aufträge einer oder mehrerer Anwendungen gemeinsam arbeiten. Es ist somit für typische Datenbankanwendungen sehr gut geeignet. In dieser Arbeit wird jedoch darauf geachtet, die breite Anwendbarkeit des Ablaufmodells zu demonstrieren (Abschnitt 5.2), indem auch andere Anwendungstypen für die betrachteten grobgranularen parallelen Systemen so strukturiert und parallelisiert wurden, daß sie über gemeinsame Daten kooperieren (Ablaufmodell Abschnitt 3.4).

Die Motivation schließt mit einer kurzen Betrachtung, warum es sinnvoll ist, komplexe zentrale Verfahren zur Lastbalancierung zu entwickeln, obwohl bisher die meisten Veröffentlichungen völlig dezentrale (und einfache) Verfahren favorisieren. Beginnend

mit der Grundidee der Lastbalancierung werden schrittweise reale Anforderungen einbezogen.

- Lose gekoppelte Parallelrechner und Rechnernetze sind erfahrungsgemäß selten voll und gleichmäßig ausgelastet; einige Rechenknoten sind überlastet, während der Großteil der Ressourcen kaum genutzt wird [Mutk92]. Daher werden automatische Lastverteilungsmechanismen entwickelt, die eine bessere Systemnutzung ermöglichen.
- Der Grundansatz zur Lastbalancierung besteht darin, jeden Rechenknoten gleich stark auszulasten. Keiner der Rechenknoten soll leer laufen, keiner überlastet sein. Je nachdem, ob das Ablaufmodell *Multitasking* enthält oder nicht, kann man die mittlere Zahl laufbereiter Aufträge (Prozesse) oder die Längen der Auftragswarteschlangen der Rechenknoten als einfaches Lastmaß verwenden.
- Parallele und verteilte Rechnersysteme sind heterogen, sobald nicht nur ein einzelner Parallelrechner oder ein völlig homogenes Cluster isoliert betrieben wird. Manche Rechenknoten haben höhere Rechenleistung oder mehrere Prozessoren, die parallel arbeiten. Solche Rechenknoten vertragen mehr Last; wenn man ihnen dieselbe Anzahl von Aufträgen anvertraut wie langsameren, so wird die Gesamtsystemleistung nicht optimal genutzt. Es kann günstiger sein, schwache Rechenknoten leer laufen zu lassen, während andere stark belastet sind.
- Aufträge sind nicht homogen. Wenn verschiedene Anwendungen konkurrierend ablaufen, sind unterschiedliche Auftragsprofile zu berücksichtigen. Aber auch innerhalb einer Anwendung erscheinen Aufträge mit unterschiedlichen Profilen, die meist von Laufzeitparametern abhängen. Die bearbeitenden Knoten müssen den Aufträgen entsprechend als höher belastet bzw. länger belegt angesehen werden, denn andere, weitere Aufträge würden dort stärker gebremst bzw. müßten länger in der Warteschlange verbleiben.
- In der Realität ist nicht nur eine Vielzahl unabhängiger, sequentieller Aufträge zu balancieren, sondern ein heterogenes Gemisch mehrerer in sich paralleler Anwendungen. Lastbalancierung muß daher mit wechselnden, unvorhersehbaren Lastsituationen fertig werden.
- Die Einzelaufträge in komplexen Anwendungen sind durch Reihenfolgebeziehungen verknüpft. Aufträge können große bzw. viele, sequentielle oder parallele Folgeaufträge haben. Damit solche Auftragsgruppen möglichst schnell zu ihrem nächsten Synchronisationspunkt gelangen (Ende der Anwendung oder Punkt mit geringer Parallelität, z.B. am Ende einer parallelen Schleife), sollten alle Aufträge etwa zugleich enden. Das ergibt neben kurzen Antwortzeiten auch die beste Nutzung der Parallelität im System. Daher sind manche Aufträge mit höherer Priorität zu bearbeiten.

- Aufträge innerhalb einer Anwendung und auch konkurrierende Aufträge aus verschiedenen Anwendungen operieren auf gemeinsamen Daten, z.B. Zwischenergebnissen, Dateisätzen oder Datenbankobjekten. Die Antwortzeit solcher Aufträge hängt stark davon ab, wieviele der benötigten gemeinsamen Daten lokal auf dem Rechenknoten verfügbar sind. Datenkommunikation erzeugt Netzbelastung und Wartezeiten und sollte daher gering gehalten werden.
- Weitere Randbedingungen sind von großer Bedeutung für die Effizienz des parallelen Systems. Rechenknoten laufen nicht in jedem Arbeitspunkt effizient; zu viele Prozeßwechsel oder Hauptspeicherüberlastung durch hohe quasi-Parallelarbeit verursachen erhebliche Zusatzkosten. Auch Überlastung von Kommunikationskanälen oder der Lastbalancierungskomponenten erzeugt Zusatzkosten und Wartezeiten. Es muß also ein vernünftiges Maß an Parallelität im System und ein sinnvolles Maß an Lastbalancierungsunterstützung gefunden werden.
- Einige in Balancierungsstrategien verwendeten Größen haben je nach Lastsituation und Anwendungsprofilen unterschiedliche große Bedeutung. Ihre Gewichtung läßt sich selten statisch vorab ermitteln. Solche Größen müssen von der Lastbalancierung selbst beobachtet und bewertet werden, um die Entscheidungsparameter optimal zu justieren und die Betrachtung auf die relevanten Größen zu beschränken.

Manche der oben genannten Herausforderungen an reale dynamische Lastbalancierungsverfahren können bereits mit einfachen Optimierungskriterien (Abschnitt 2.5.7) erfaßt werden. Der *HiCon*-Ansatz integriert erstmalig alle genannten Anforderungen in einem allgemein verwendbaren dynamischen Lastbalancierungskonzept.

3.2 Klassifikation des Ansatzes

Kapitel 2 gab einen Überblick über diverse Techniken zur Lastbalancierung. Bevor der *HiCon*-Ansatz detailliert vorgestellt wird, soll er kurz charakterisiert werden. Die Klassifikation richtet sich dabei nach bekannten Übersichtsartikeln zur dynamischen Lastbalancierung [Bern93], [Casa88], [He89], [Scha92], [Shir92].

Als Systemplattform werden Workstations und lose gekoppelte Parallelrechner angenommen. Ein Rechenknoten besteht aus einem Prozessor oder mehreren speichergekoppelten Prozessoren sowie Sekundärspeicher, Ein- / Ausgabegeräten und Netzwerkanschlüssen. Jede Workstation stellt also einen Rechenknoten dar, ebenso ein SMP (*shared memory processor*) Parallelrechner, während die Knoten eines MPP (*massively parallel processor, shared nothing*) Parallelrechners als einzelne Rechenknoten balanciert werden können.

Mehrbenutzerbetrieb durch konkurrierende Anwendungen im System ist möglich. Jede der Anwendungen kann aber auch aus diversen, teilweise parallel laufenden und kooperierenden Teilaufträgen bestehen. Anwendungen sind Client - Server strukturiert (Abschnitt 3.4): Aufträge sind Serverklassenaufrufe; Aufträge können dabei komplette Anwendungen oder Teilaufgaben einer Anwendung repräsentieren. Jeder Auftrag kann auf einem beliebigen Rechenknoten ausgeführt werden.

Die Ausführung eines Auftrags erfolgt im *Multitasking* (Prioritäten- oder Zeitscheibenverfahren, je nach Betriebssystem) auf einem Knoten. Einmal gestartet, kann er nicht mehr migriert werden. Es besteht auch die Möglichkeit, Aufträge in einer lokalen Warteschlange je Knoten zu puffern, so daß nicht notwendigerweise alle dort platzierten Aufträge parallel oder quasi-parallel ablaufen müssen. Die Ausführung eines Auftrags beansprucht Prozessorrechenzeit, Ein- / Ausgabeleistungen und Kommunikation. Das genaue Ablaufmodell folgt in Abschnitt 3.4.

Das Netzwerk verbindet Rechenknoten untereinander und hat eine beliebige Struktur. Kooperation zwischen Aufträgen geschieht über synchronisiertes Arbeiten auf gemeinsamen Daten. Daten können migriert und kopiert werden. Es gibt keine entfernten Operationen auf Daten im Sinne eines *remote memory access*, sondern die Daten müssen zu dem Rechenknoten transportiert werden, auf dem der zugreifende Auftrag abläuft - oder es wird ein Auftrag verschickt, der einen Server bei den Daten aktiviert.

Die Lastbalancierung setzt, wie in Abbildung 12 angedeutet, nur anwendungsunabhängige Informationen ein. Da die Lastverteilung Teil eines verteilten Betriebssystems sein sollte, können keine Eigenschaften und Kenntnisse einer speziellen Anwendung ausgenutzt werden, wie etwa in [Bog192], [Cap92], [Kreu89], [Sinh93], [Will91]. *HiCon*-Lastbalancierung mißt nicht nur die Ressourcenauslastung auf Systemebene und setzt sie für reaktive Entscheidungen ein, sondern verwendet auch Vorabschätzungen von Anwendungen über Auftragsprofile und Auftragsgruppenprofile.

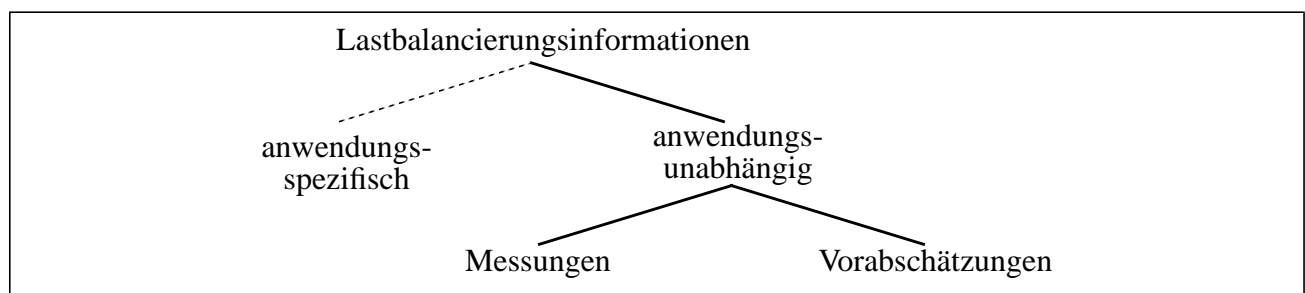


Abbildung 12: Klassifikation der genutzten Lastbalancierungsinformationen.

In Abbildung 13 sind die Aufgaben und damit Einflußmöglichkeiten der Lastbalancierung klassifiziert. Der hier vorgestellte Ansatz beschränkt sich in der Auftragsverwaltung auf die Zuweisung von Aufträgen, da die Migration laufender Prozesse in

heterogenen Systemen noch nicht mit vernünftigem Aufwand realisierbar ist. Der Zuweisungszeitpunkt für Aufträge ist beliebig, d.h. nicht unbedingt sofort bei Entstehung des Auftrags und nicht unbedingt erst, wenn ein Knoten oder Server leer läuft. Die Möglichkeit der Auftragsduplikation ist nur für kontextfreie Aufträge, keinesfalls für Berechnungen auf globalen Daten anwendbar: gibt man einen Auftrag mehrfach aus, um das Ergebnis des schnellsten Bearbeiters weiter zu verwenden und die anderen Bearbeitungen zu verwerfen, so darf der Auftrag keine globalen Daten verändern (oder die Modifikationen der anderen Bearbeiter müssen rückgängig gemacht werden).

Im Bereich der Datenverwaltung kann Lastbalancierung sowohl die Verlagerung von Daten als auch die Verteilung von Datenkopien beeinflussen. Die realisierten Strategien nutzen dieses Potential jedoch nur indirekt durch Auftragsplatzierungen, woraufhin Datenverschiebungen automatisch durch eine Datenverwaltungskomponente des verteilten Betriebssystems erfolgen (Abschnitt 3.5).

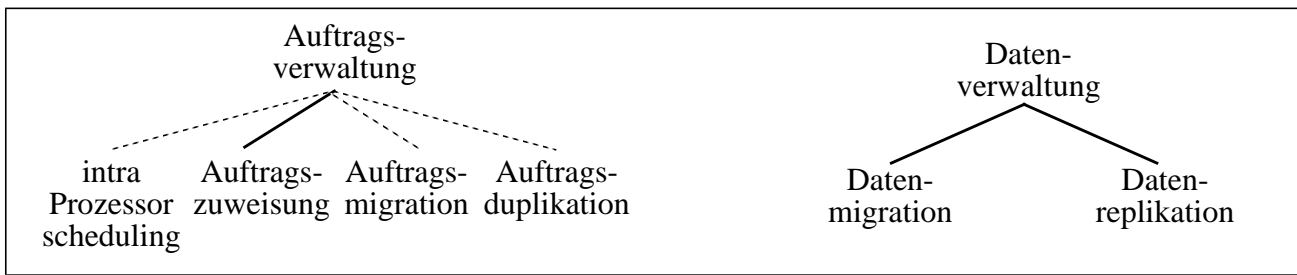


Abbildung 13: Klassifikation der Lastbalancierungsaufgaben.

Die Struktur der Lastbalancierung kann rein zentral sein, kooperierende Cluster unterstützen, oder völlig dezentral konfiguriert werden. Dezentrale Konfigurationen können mit explizit oder implizit verteilter Lastbalancierung betrieben werden. Die zentrale Struktur kennt keine Unterscheidung zwischen Sender- und Empfänger-Initiierung. Die dezentrale Erweiterung verläuft Sender-initiiert, da Lastbalancierungsagenten Aufträge an weniger belastete Nachbar-Cluster abgeben.

Der Lastbalancierungsansatz ist dynamisch, da Entscheidungen zur Laufzeit aufgrund aktueller Informationen getroffen werden. Er ist adaptiv, weil Entscheidungsparameter zur Laufzeit durch Auswertung längerfristiger Beobachtungen angepaßt werden. Obwohl der Ansatz dynamisch ist, werden zusätzlich Techniken statischer Balancierungsverfahren eingesetzt.

Im Sinne der gebräuchlichen Klassifikationen [Casa88] ist der Ansatz suboptimal und heuristisch. Er garantiert keine Bestverteilung der Last, sondern erreicht durch Ausnutzung grober und vereinfachter Zusammenhänge eine Verbesserung des Systemdurchsatzes. Weiterhin ist sowohl isolierte Lastbalancierung einzelner Aufträge möglich als auch sogenannte soziale Lastbalancierung, d.h. Interaktionen, Abhängigkeiten und

gegenseitige Behinderungen zwischen Aufträgen werden berücksichtigt. Diverse Lastkenngrößen können zur Entscheidungsfindung genutzt werden (Abschnitt 3.7.1, 3.7.2).

3.3 Kurzvorstellung des Gesamtsystems

Abbildung 14 gibt einen Gesamtüberblick eines Clusters im Gesamtsystem: das verteilte Betriebssystem, die Komponenten zur Abwicklung der Lastbalancierung, und die Komponenten der Anwendungen, d.h. Clients, Server und Serverklassen (Abschnitt 3.4). Außerdem ist das Konzept der gemeinsamen Daten skizziert. Das gesamte Laufzeitsystem zur Abwicklung der Anwendungen sollte Teil eines geeigneten verteilten Betriebssystems sein, wobei manche Teile heutzutage als Datenbankkomponente oder *Transaction-Processing Monitor* realisiert sind. Charakteristisch sind in diesem Systemmodell - im Vergleich zu vielen anderen Projekten - zum einen die logisch zentralisierten Funktionen der Lastbalancierung, die Verwaltung der Systemzustandinformation, das Treffen der Balancierungsentscheidungen und die Adaption der Strategie, zum anderen die Auftrennung der Lastbalancierung in eine unmittelbar agierende Komponente zur Bewertung und Auftragszuweisung, und eine Anpassungskomponente, die dynamische Adaption der Lastbalancierungsstrategie ermöglicht. Es wird keine spezifische Prozessor- oder Netzwerktopologie vorausgesetzt. Das Bild gibt die Struktur eines Clusters wieder. Zur Skalierung auf sehr große Systeme können beliebig viele Cluster dieser Art vernetzt werden (Abschnitt 3.6), die dezentral kooperieren.

Die unten in der Abbildung verlaufenden Querbalken sollen die Rechenknoten eines Clusters symbolisieren. Auf diese Knoten sind die Anwendungen verteilt. Eine Anwendung besteht jeweils aus einem Client (als weiße Ellipsen angedeutet), der Aufträge zur Bearbeitung generiert. Über das System sind Server verteilt (ebenfalls durch weiße Ellipsen dargestellt). Server sind nach Funktionsgruppen in Serverklassen (schwarze Hinterlegung) eingeteilt und können die Aufträge der Anwendungen bearbeiten. Die Funktionalität des verteilten Betriebssystems ist in einem Kasten darüber skizziert. Sie besteht in der Realität aus zentralen Teilen und solchen, die auf die Knoten verteilt sind. Auch die Messung der Ressourcenbelastung ist auf das System verteilt. Die Lastbalancierung ist in der Abbildung auf oberster Ebene angesiedelt, da sie vom Laufzeitsystem Informationen erhält und über das Laufzeitsystem die Verteilung der Anwendungsaufträge vornimmt. Sowohl die Lastbalancierung als auch das verteilte Betriebssystem kooperieren mit den entsprechenden Komponenten benachbarter Cluster. Im folgenden werden die Komponenten und ihre Funktion genauer beschrieben.

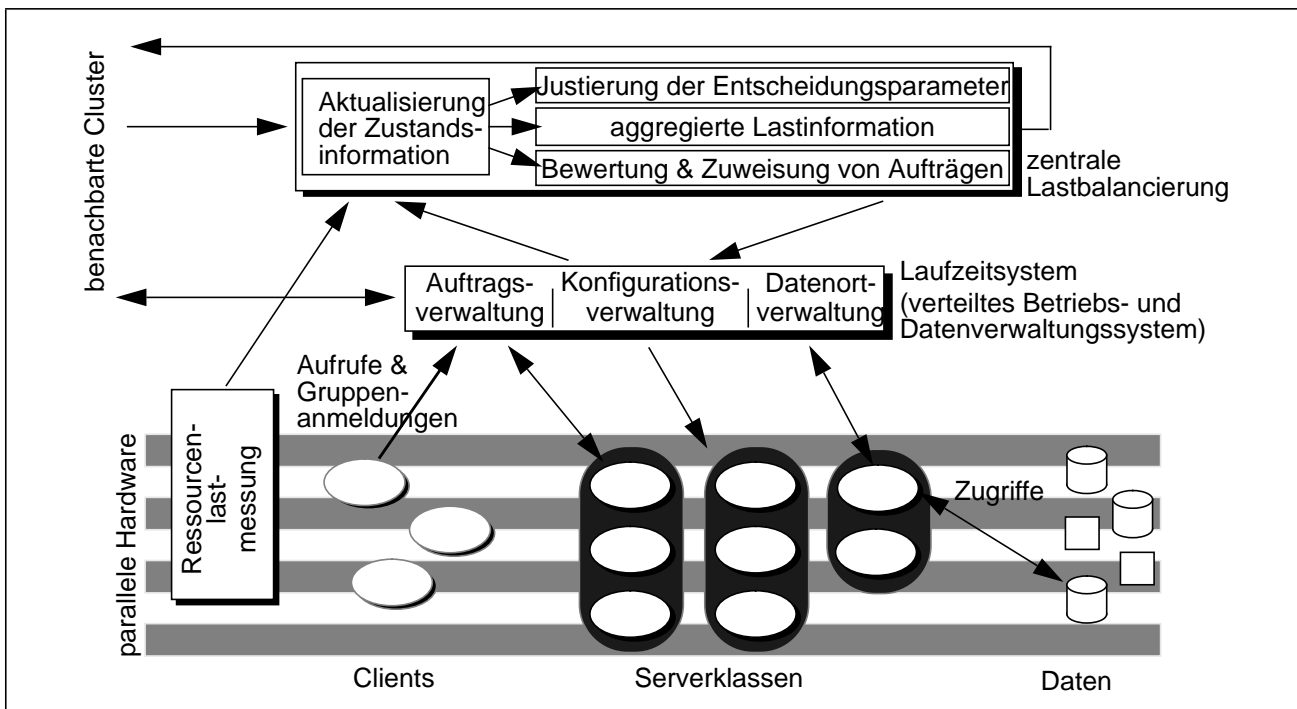


Abbildung 14: Komponenten eines Clusters im Gesamtsystem - Anwendungen, Betriebssystem und Lastbalancierung.

3.4 Verarbeitungsmodell für Anwendungen

Je allgemeiner die Ablaufstrukturen, die Kooperation und die Synchronisation in parallelierten Anwendungen und zwischen konkurrierenden Aufträgen sind, desto schwieriger ist es, die Auslastung des Systems zu interpretieren, um geeignete Maßnahmen zur günstigeren Lastverteilung treffen zu können. Es finden sich zahlreiche Arbeiten, in denen eine beliebige Menge kommunizierender und unabhängiger Prozesse betrachtet wird. Automatische Lastverteilung ist hier aufgrund der mangelnden Informationen über die laufenden Anwendungen nur begrenzt möglich. Extreme Leistungssteigerungen wurden dagegen in Projekten erreicht, die für spezielle parallelisierte Anwendungen dedizierte Lastausgleichsverfahren entwickelt haben. Der hier vorgestellte Ansatz benutzt ein eingeschränktes Ablaufmodell, das der Lastbalancierung eine Interpretation des Systemverhaltens ermöglicht, und er erlaubt den Anwendungen, die an sich anwendungsunabhängige Lastbalancierung durch Hinweise zu unterstützen.

Client - Server Modell. Als Verarbeitungsmodell wird das Client - Server Modell verwendet. Dieses Modell hat sich in Datenbank-Umgebungen und bei nahezu allen größeren, parallelen und verteilten Anwendungen seit längerem bewährt. Das Client - Server Modell ist ein einfaches Wechselspiel zwischen Funktionsaufruf, Bearbeitung und Ergebnisrückgabe, erlaubt aber durch die Konzepte der Serverklassen eine sehr

flexible Verteilung, Parallelisierung und Lastbalancierung innerhalb und zwischen Anwendungen.

Serverklassen. Anwendungstypen oder auch Teilfunktionen von Anwendungstypen werden als Serverklassen bezeichnet. Eine Serverklasse stellt eine bestimmte Funktionalität als Dienst zur Verfügung. Die Funktion einer Serverklasse besitzt Eingabeparameter, die beim Funktionsaufruf belegt werden. Da eine Serverklasse üblicherweise mehrere Funktionen zur Verfügung stellt, z.B. verschiedene Zugriffsoperationen auf einem bestimmten abstrakten Datentyp, teilt man die Funktionalität einer Serverklasse in Subklassen (Auftragstypen) ein.

Server. Zur Ausführung einer solchen Funktion benötigt man einen Server, der die Funktionalität der Serverklasse anbietet. Das ist in der Regel ein Prozeß, der entsprechenden Programmcode ausführt. Prinzipiell kann ein Server auch durch mehrere kooperierende Prozesse realisiert werden, aber im *HiCon*-Modell ist jeder Server durch einen sequentiellen Prozeß realisiert. Ein Server durchläuft eine endlose Schleife, in der er jeweils auf Aufrufparameter wartet, die Funktion durchführt und dann die Ergebnisse zurückschickt. Es wird also nicht für jede Funktionsausführung ein Server kreiert, sondern es wird eine gewisse Anzahl an Servern auf dem System konfiguriert, die für Funktionsausführungen bereitstehen. Nach Ausführung eines Funktionsaufrufs steht der Server sofort wieder zur Ausführung des nächsten Auftrags bereit. Pro Serverklasse können im System beliebig viele Server gleichzeitig aktiv sein. Server verbrauchen keine Ressourcen, solange sie keinen Auftrag zu bearbeiten haben.

Clients. Die Ausführung einer Anwendung wird durch einen Client repräsentiert. Ein Client ist ein beliebiger Prozeß, der die Verarbeitung einer Anwendung durch Absenden verschiedener Aufrufe (Aufträge) an Serverklassen koordiniert. Clients können genau einen Serverklassenaufruf tätigen, der die gesamte Anwendungsarbeit ausführt, sie können jedoch auch beliebige Aufrufsequenzen und parallele Aufrufe an dieselbe und unterschiedliche Serverklassen durchführen. Server können auch selbst Unteraufrufe an andere Serverklassen absetzen und nehmen dabei die Rolle eines Clients an.

Auftragsbearbeitung. Aufträge sind Serverklassenaufrufe, die von Clients abgesetzt werden. Beim Absetzen eines Auftrags spezifiziert der Client die gewünschte Funktion (Serverklasse) und gibt die Aufrufparameter an. Die Struktur der Parameter ist anwendungsabhängig. Der Auftrag wird vom Betriebssystem entgegengenommen. Es garantiert, daß der Auftrag irgendwann ausgeführt wird und ein Ergebnis zurückliefert. Jeder Auftrag muß also sofort ausführbar sein, wenn er vom Client abgesandt wird (d.h. er ist sofort zur Bearbeitung freigegeben). Nach Absendung eines Auftrags kann der Client weiterlaufen; Er ist nicht blockiert bis das Ergebnis eintrifft (*asynchronous remote procedure call*). Auf diese Art kann er verschiedene Aufträge (auch derselben Server-

klasse) parallel bearbeiten lassen. Das Ablaufmodell garantiert nicht, daß die Aufträge in derselben Reihenfolge gestartet oder bearbeitet werden, wie der Client sie abgeschickt hat. Nicht einmal die Ergebnisse müssen in derselben Reihenfolge eintreffen. Die Server geben am Ende der Bearbeitung eines Auftrags Ergebniswerte ab, deren Struktur ebenfalls anwendungsabhängig gewählt werden kann. Das Betriebssystem sendet die Resultate zurück an den Aufrufer (Client), wo sie ggfs. gepuffert werden. Sobald ein Client die Ergebnisse eines oder mehrerer Aufträge benötigt, um fortfahren zu können, so kann er warten bis er Ergebnisse früherer Aufrufe erhält. Die häufige Einschränkung, daß asynchrone Aufrufe kein Resultat haben dürfen, existiert im *HiCon*-Ablaufmodell nicht.

Abbildung 15 skizziert die Entstehung, Verwaltung und Bearbeitung von Aufträgen innerhalb eines Clusters. Ein Client erzeugt einen Auftrag mit Parametern und einer Profil-Vorabschätzung zur Unterstützung der Lastbalancierung (1). Die zentrale Lastbalancierung ordnet den Auftrag in der zentralen Warteschlange ein (2). Dabei werden evtl. Prioritäten berücksichtigt. Die Lastbalancierung trifft sofort oder später eine Zuweisungsentscheidung (3), woraufhin der Auftrag an einen Server gesandt wird. Wenn der betreffende Server bereits arbeitet, wird der Auftrag dort in die lokale Warteschlange eingereiht (4). Der Server bearbeitet den Auftrag vollständig (5). Wenn auf dem Prozessor mehrere Server arbeiten, so teilen sie sich die Rechenzeit gemäß der Scheduling-Strategie des lokalen Betriebssystems. Die Rücksendung des Resultats ist ein Ereignis, das die Lastbalancierung aktiviert (6). Sie kann auf die neue Situation hin Aufträge neu bewerten und evtl. Aufträge an Server oder Nachbar-Cluster zuweisen.

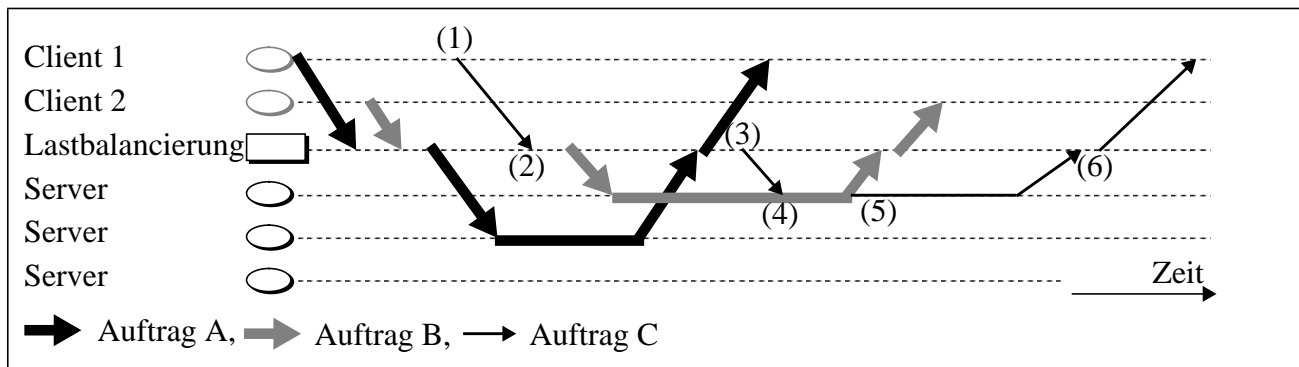


Abbildung 15: Entstehung, Verwaltung und Bearbeitung eines Auftrags.

Daten. In realen Anwendungen arbeiten Server nicht kontextfrei (rein von Aufrufparametern abhängig), sondern verwenden und modifizieren bei der Auftragsbearbeitung globale Daten, seien es gemeinsame Daten in einer Anwendung oder globale, evtl. persistente Datenbestände. Persistente Daten existieren über den Ablauf einer Anwendung hinweg, wofür Datenbankobjekte das wichtigste Beispiel darstellen. Gemeinsame Daten einer Anwendung, z.B. große Matrizen, existieren hingegen nur für den Verlauf

der Anwendung existieren. Das *HiCon*-Modell erlaubt daher kontextsensitive Bearbeitungen. Kontextsensitive Serverklassen bieten eine Funktionalität, bei der die Resultate vom aktuellen Anwendungs- oder globalen Kontext abhängen. Weiterhin haben diese Serverklassen auch Seiteneffekte, d.h. geben nicht nur Ergebnisse zurück sondern modifizieren auch den Kontext.

Dazu werden Daten explizit in das Verarbeitungsmodell für die *HiCon*-Lastbalancierung integriert: Anwendungen arbeiten auf globalen Datensätzen. Datensätze werden von Anwendungen kreiert und benannt. Die Strukturen und Speicherungsformen der Datensätze sind anwendungsspezifisch, d.h. jeder Datensatz kann beliebig aus Haupt- und Sekundärspeicherstrukturen bestehen. Datensätze können global innerhalb eines Anwendungslaufs, global innerhalb einer Serverklasse oder systemglobal sein. Die Datensätze können für die Dauer einer Anwendung existieren oder persistent sein. Jede Anwendung kann überall auf gemeinsame Datensätze zugreifen, da die Synchronisation, Datenlokalisierung und Replikationsverwaltung vom verteilten Betriebssystem übernommen wird. Für nichtdeterministische parallele Abläufe oder unabhängig parallel arbeitende Aufträge können Zugriffe auf gemeinsame Daten durch geeignete Sperren synchronisiert werden. Den Anwendungen wird also durch die Datenverwaltungs-komponente des verteilten Betriebssystems ein virtueller gemeinsamer Speicher zur Verfügung gestellt. Alle Synchronisation und Datenkommunikation innerhalb und zwischen Anwendungen wird im *HiCon*-Modell durch die Clients (Aufrufverhalten, Parameter) und durch die Server (Arbeit auf gemeinsamen Daten) realisiert.

Das *HiCon*-Verarbeitungsmodell ist so allgemein gehalten, daß die genaue Funktionsweise der Datenverwaltungs-komponente unerheblich ist. Das Verfahren zur Realisierung, Lokalisierung, Synchronisation und Verteilung der virtuell gemeinsamen Daten wird im *HiCon*-Modell nicht vorgeschrieben. Für die Lastbalancierung ist lediglich wichtig, daß gemeinsame Daten existieren, die physisch auf das System verteilt sind, daß Server auf ihnen operieren, und daß Zugriffe auf nicht-lokal vorliegende Daten Zusatzaufwand verursachen. Auftragsbearbeitungen der Server können also durch Zugriffe auf Daten, die momentan nicht lokal vorhanden sind, unterbrochen werden, weil die Server auf die Daten warten müssen.

Zum besseren Verständnis der folgenden Kapitel soll hier kurz ein allgemeines, flexibles Schema zur Verwaltung logisch gemeinsamer, physisch verteilter Daten skizziert werden, das auch in der prototypischen Implementierung realisiert wurde (Abschnitt 5.1). Heute verfügbare Verfahren in *virtual shared memory* Systemen und Datenverwaltungssystemen sind ähnlich, jedoch meist weniger flexibel. Als Zuständiger (Besitzer) eines Datensatzes gilt jeweils der Server, der zuletzt eine Änderungsoperation darauf durchgeführt hat. Dort werden die zentralen Verwaltungsinformationen und -

Operationen für diesen Datensatz gespeichert / durchgeführt. Datensatzkopien werden auf Zugriffsanforderungen hin automatisch im System verteilt und vor Änderungsoperationen wieder invalidiert (d.h. nicht nach Änderungsoperationen aktualisiert). Eine derartige Datenverwaltung optimiert den Zugriffsaufwand, sofern die Server eine gewisse Lokalität im Datenreferenzverhalten aufweisen: Wiederholte Lesezugriffe auf lokale Datenkopien sind sehr günstig, und die Anzahl und Verteilung von Kopien paßt sich automatisch an das Lese-/Schreibverhältnis der Datenzugriffe an. Die *HiCon*-Lastbalancierung versucht, diese Zugriffslokalität zu erhöhen. Konsistenz wird gesichert, indem Server Zugriffe auf Datensätze durch entsprechende Schreib-/Lesesperren kapseln. Abbildung 16 skizziert den Ablauf eines Datenzugriffs beispielhaft für den Fall, daß ein Server einen Datensatz exklusiv benötigt und außer dem derzeitigen Besitzer zwei weitere Server gültige Kopien besitzen. Wichtig für die Lastbalancierung ist außerdem, daß solche durch Datenkommunikation entstehenden Wartezeiten durch Berechnungen anderer Server auf dem Knoten genutzt werden können.

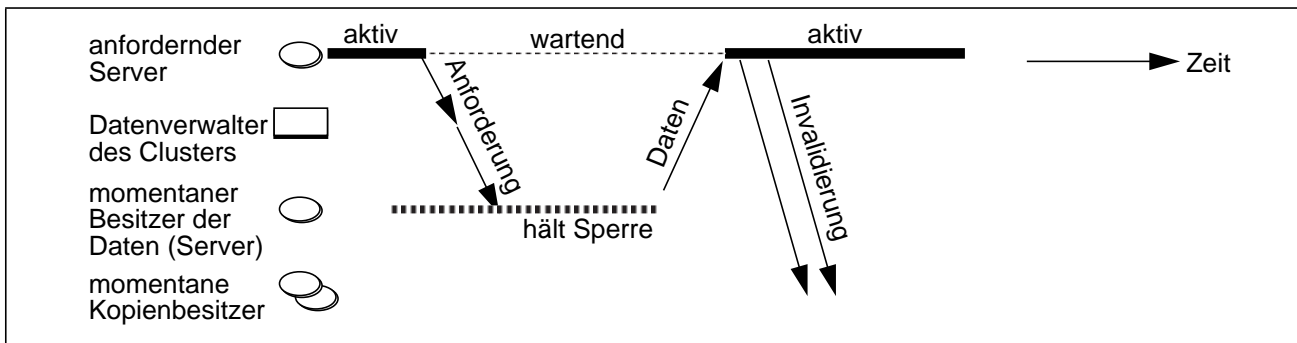


Abbildung 16: Kommunikationsablauf für exklusiven Zugriff auf nicht-lokale Daten.

Konsequenzen des Modells für Lastbalancierung und Anwendungen. Die Wahl des Ablaufmodells hat starken Einfluß auf die Existenz und Relevanz von Lastkenngrößen und auf das grundsätzliche Verbesserungspotential der Lastbalancierung. Außerdem sollte das Ablaufmodell effiziente und einfache Entwicklung, Strukturierung und Ausführung von Anwendungen ermöglichen.

Das Client - Server Verarbeitungsmodell hat sich in nahezu allen Datenverwaltungssystemen und in allen großen, weiträumig verteilten Anwendungen als brauchbar erwiesen und durchgesetzt. Andererseits sind viele existierende parallele Anwendungen, insbesondere auf Großrechnern bzw. enggekoppelten Systemen und im ingenieurwissenschaftlichen Bereich, nach dem SPMD-Modell (single program multiple data) bzw. nach dem CSP-Modell (communicating sequential processes) strukturiert. Im SPMD-Modell werden beim Start der Anwendung gleichartige Prozesse auf die Rechenknoten geladen, die dann rechnen und miteinander kommunizieren; Die Synchronisation und der Datenaustausch erfolgt allein durch Nachrichten, die beliebig zwischen diesen Pro-

zessen ausgetauscht werden. Im CSP-Modell bestehen Anwendungen aus beliebigen Prozessen, die beliebig kommunizieren.

Das Konzept der Serverklassenaufrufe erlaubt eine sehr flexible Verteilung der Aufträge im parallelen System. Jeder Serverklassenaufruf kann auf einem beliebigen Rechenknoten ausgeführt werden. Lastbalancierung kann Profilabschätzungen und Beobachtungen für bestimmte Serverklassen machen. Die Bearbeitungen von Aufträgen durch Server ist ein sehr einfaches Modell, das auch einfache Vorabschätzungen der entstehenden Last erlaubt, während das Verhalten beliebig kommunizierender Prozesse schwer geeignet zu modellieren ist. Das Konzept der Kooperation über gemeinsame Daten ermöglicht eine flexible Verteilung der Aufträge im System auch für Serverklassen, die nicht kontextfrei sind. Die gemeinsamen Daten sind für die Lastbalancierung Objekte, anhand derer sie sowohl die Kommunikation zwischen Aufträgen, die Weitergabe von Zwischenergebnissen zwischen Aufträgen als auch die Datenaffinität bei Zugriffen auf globale, persistente Daten in ihr Kostenmodell einbeziehen kann.

Das Client - Server Modell mit asynchronen Serverklassenaufrufen und Kooperation über gemeinsame Daten ist nicht für alle Anwendungen das eleganteste oder effizienteste Programmiermodell. In dieser Arbeit sollen die verschiedenen Programmierparadigmen nicht detailliert diskutiert werden; Es sei lediglich darauf hingewiesen, daß das Modell sehr mächtig und flexibel, einfach zu programmieren und nachvollziehbar ist. In Abschnitt 5.2 werden auch Anwendungen betrachtet und erfolgreich Client - Server strukturiert, die typische Vertreter des SPMD-Modells sind. Die Praxis hat jedoch klar erwiesen, daß die in SPMD und CSP verwendete Synchronisation bzw. Datenflußsteuerung durch direkte Nachrichten zwischen Aufträgen für bestimmte Anwendungsklassen - im Vergleich zum Client - Server Modell und dem Konzept der Arbeit auf gemeinsamen Daten - feinere Auftragsgranulate und höhere Parallelisierung ermöglichen.

3.5 Das Betriebssystem zur Verwaltung der Anwendungsläufe

Anwendungsunabhängige dynamische Lastbalancierung sollte in ein verteiltes Betriebssystem eingebettet sein. Ein verteiltes Betriebssystem übernimmt die Prozeß-Server-Verwaltung, die Datenverwaltung und die Verwaltung von Kommunikationsvorgängen zwischen Prozessen. Das sind lediglich die für Lastbalancierung unmittelbar relevanten Funktionen. Die Prozeßverwaltung lädt auszuführende Programme als Prozesse in den Hauptspeicher der Knoten, startet sie und wechselt die Ausführung zwischen mehreren ausführbereiten Prozessen auf einem Prozessor im Zeitscheiben- oder Prioritätenverfahren. Die Datenverwaltung stellt den Programmen Hauptspeicherdaten durch lineare Adressierung und Sekundärspeicherdaten durch Dateinamen und

stückweise Einlagerung in den Hauptspeicher zur Verfügung. Idealerweise sind Dateien über ihren Namen im gesamten parallelen und verteilten System für die Anwendungen gleichermaßen sichtbar. Kommunikation zwischen Prozessen wird durch logische Verbindungen und gepufferten Transfer von Hauptspeicherdaten ermöglicht. Die Realisierung der virtuellen Speicherverwaltung (Abschnitt 2.4) ist eine weitere Funktionalität des Betriebssystems, die für Lastbalancierungsbemühungen wichtig ist.

Im Rahmen dieser Arbeit soll die Funktionalität verteilter Datenverwaltungssysteme ebenfalls als Teil des verteilten Betriebssystems betrachtet werden, obwohl Datenverwaltungssysteme heute meist noch zentraler Struktur sind und in vielen Betriebssystemen als einzelne, unabhängige Schicht aufgesetzt werden. Ein solches Datenverwaltungssystem bietet zum einen den Prozessen eine einfache Funktionalität zum Arbeiten auf gemeinsamen Sekundärspeicherdaten: Einzelne Datensätze von feinem Granulat können global adressiert werden und die Anwendungen können sehr effizient darauf zugreifen, wobei bestimmte Synchronisationsregeln automatisch eingehalten werden. Weiterhin stellen viele Datenverwaltungssysteme durch sogenannte *Transaction Processing Monitore* eine Prozeß- und Auftragsverwaltung ähnlich dem Client - Server Konzept zur Verfügung (Abschnitt 3.4). Das ermöglicht eine flexible Verteilung und Parallelisierung von Anwendungen bei relativ feinem Auftragsgranulat.

Im folgenden sollen die drei Komponenten des Betriebssystems näher spezifiziert werden, die für die Einbettung und Einflußnahme der *HiCon*-Lastbalancierung wesentlich sind. Das Betriebssystem verwaltet die Konfiguration der Server auf den Rechenknoten, die Zuweisung von Aufträgen an die Server und die Verteilung der gemeinsamen Daten auf die Rechenknoten bzw. Server.

Die *Konfigurationsverwaltung* betreibt auf jedem Rechenknoten eine gewisse Anzahl von Prozessen, die die Funktionalität der Serverklassen realisieren. Die Anzahl der verfügbaren Server pro Knoten kann durch die Lastbalancierung zur Laufzeit geregelt werden. Jeder Server wird als sequentieller Betriebssystemprozeß realisiert. Neuere Betriebssysteme ermöglichen sogenannte *multi threaded* Server. Dabei kann für jede Auftragsbearbeitung ein neuer Thread im Serverprozeß kreiert werden, was eine flexible und kostengünstige Handhabung der Parallelbearbeitung erlaubt. Threads können innerhalb eines Prozesses konkurrierend laufen. Thread-basierte Server haben außerdem den Vorteil, daß Wartezeiten in einer Bearbeitung durch den schnellen Tread-Wechsel effizient für andere Bearbeitungen genutzt werden können. Im vorgestellten Lastbalancierungsmodell werden *multi threaded* Server nicht explizit unterstützt (Abschnitt 6.2). Solange der virtuelle Speicher ausreicht, können beliebig viele Server pro Knoten konfiguriert werden, ohne daß die Rechenknoten dadurch zusätzlich belastet werden, denn Server verbrauchen keine Ressourcen, solange sie keine Aufträge

bearbeiten. Dennoch ist es wichtig, die Anzahl der konfigurierten Server nicht wesentlich größer zu wählen als die tatsächlich genutzte Parallelität der Serverklasse. Ein Grund ist, daß Aufträge durch die Lastbalancierung nicht an Rechenknoten, sondern an einzelne Server zugewiesen werden. Dadurch steigt der Verwaltungs- und Entscheidungsaufwand für die Lastbalancierung mit der Anzahl der verfügbaren Server. Außerdem steigt gewöhnlich bei jeder Implementierung der Aufwand für die Lastbalancierung und das Betriebssystem mit der Anzahl der Server, und meist auch der Speicherbedarf durch mehrfach replizierte Daten auf Knoten.

Die *Auftragsverwaltung* ist für die Entgegennahme und Abwicklung von Aufträgen (Serverklassenaufrufen) zuständig. Wie in Abbildung 17 skizziert, können Aufträge zuerst in einer globalen Warteschlange je Cluster aufbewahrt werden. Zu einem beliebigen Zeitpunkt, der von der Lastbalancierung bestimmt wird, werden Aufträge an Server zugewiesen, d.h. in deren lokale Warteschlange eingereiht. Dies muß nicht in der Reihenfolge geschehen, in der die Aufträge ins System kamen. Die Zuweisung eines Auftrags an einen Server ist endgültig, d.h. danach ist keine Migration mehr möglich. Die Server arbeiten die Aufträge in ihrer lokalen Warteschlange sequentiell in der Reihenfolge ab, in der sie in der Warteschlange eintrafen. Parallelarbeit auf einem Rechenknoten, d.h. Multitasking bzw. echte Parallelität auf Multiprozessorknoten, kann durch mehrere aktive Server pro Knoten erreicht werden. Auftragsergebnisse werden vom Betriebssystem an die zugehörigen Clients zurückgeschickt. Das Betriebssystem garantiert also weder, daß Aufträge in derselben Reihenfolge bearbeitet werden, in der sie von Clients generiert wurden, noch, daß sie unmittelbar gestartet werden. Das Warteschlangen ist wichtig, weil im *HiCon*-Modell keine Migration von Aufträgen möglich ist. So kann die Lastbalancierung die Auftragszuweisung beliebig handhaben, d.h. Aufträge früh auf die Server verteilen, wenn die Strategie sehr einfach agieren muß (etwa weil sie überlastet ist) oder sie einen weiten Vorausblick auf das zukünftige Lastverhalten hat, oder wenn die Aufträge sehr klein sind. Ansonsten kann sie Aufträge möglichst spät (etwa wenn der momentan beste Server frei ist) zuweisen, um bei der Zuweisung die aktuelle Situation zu berücksichtigen.

Aufgabe der *Datenverwaltung* ist die Bereitstellung eines virtuellen gemeinsamen Datenspeichers für die Anwendungen. Die Anwendungen können beliebige Datenstrukturen aus Haupt- und Sekundärspeicher als gemeinsam deklarieren und melden lesende oder exklusive Zugriffe jeweils beim Betriebssystem an. Da die Rechenknoten keine gemeinsamen Speichermedien haben, muß das Betriebssystem bei Zugriffsanforderungen die Daten über das Netzwerk zum anfordernden Server schicken. Für Lastbalancierungserwägungen ist es wichtig zu wissen, daß Operationen auf gemeinsamen Daten teuer sein können, d.h. Wartezeiten bei der Bearbeitung und Last auf dem Netzwerk erzeugen, wenn die Daten nicht lokal beim Server vorliegen (Abschnitt 3.4).

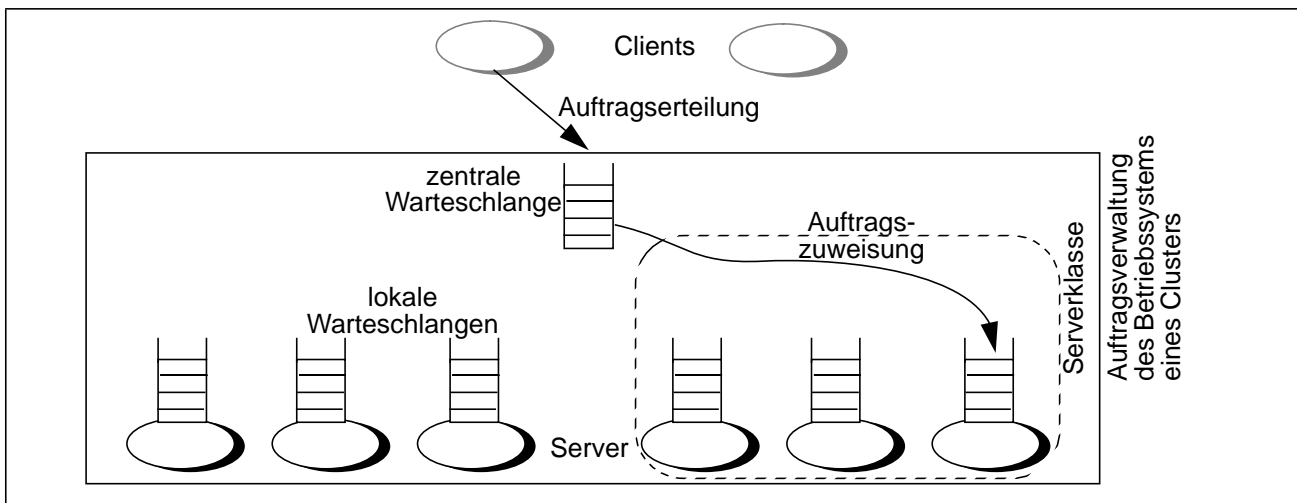


Abbildung 17: Auftragsverwaltung mit zentralen und lokalen Warteschlangen

3.6 Ein Netzwerk kooperierender Lastbalancierungsagenten

Für das verteilte Betriebssystem wird angenommen, daß es beliebig große Rechnersysteme überspannen kann. Um diese Skalierbarkeit zu gewährleisten, darf es keine zentralisierten Funktionen für das Gesamtsystem enthalten. Kritische Komponenten sind dabei sowohl die Konfigurations- als auch die Auftrags- und Datenverwaltung. Die Konfigurationsverwaltung sollte jeweils für Cluster autonom realisiert sein, um die Wartung und den Betriebsablauf zu vereinfachen. Die Auftragsverwaltung hat pro Serverklasse eine zentrale Warteschlange für Aufträge, die noch keinem Server zugewiesen sind. Derartiges kann lediglich für Cluster begrenzter Größe realisiert werden. Skalierbarkeit verlangt auch, daß die Datenverwaltung keinen globalen Überblick des Gesamtsystems haben kann, sondern nur je ein Cluster verwaltet. Aufträge und Informationen über Daten-Aufenthaltsorte müssen daher zwischen den jeweiligen Auftrags- und Datenverwaltungs-komponenten ausgetauscht werden, wenn sie Cluster-Grenzen überschreiten.

Im *HiCon*-Modell wird die Lastbalancierungsstruktur an die Struktur des verteilten Betriebssystems angepaßt, d.h. es wird die in Abschnitt 2.5.3.2 eingeführte dezentrale Struktur verwendet. Zahlreiche Studien haben aufgezeigt, daß eine zentralisierte Lastbalancierung nicht unbegrenzt skalierbar ist. Wird das zu regelnde System oder die Anzahl und Ankunftsrate von Aufträgen sehr groß, so verbraucht die Lastbalancierung selbst viel Speicher und Rechenleistung, und die Verzögerung zwischen Absendung eines Auftrages und dessen Bearbeitungsbeginn wächst an. Zentrale Lastbalancierung wird zum Engpaß, wenn ihr Ressourcenbedarf in derselben Größenordnung liegt wie der Ressourcenbedarf der laufenden Anwendungen oder die Verzögerung von Aufträgen in der Größenordnung der eigentlichen Auftragsbearbeitungszeiten liegt. Wie in

Abschnitt 2.5.3.1 diskutiert, hat aber zentrale Lastbalancierung entscheidende Vorteile, so daß jeweils möglichst große Cluster durch zentrale Verfahren balanciert werden sollten.

Das *HiCon*-Modell sieht vor, daß jeder Lastbalancierungsagent einen möglichst großen Teil (Cluster) des gesamten Systems zur lokal-zentralen Balancierung zugewiesen bekommt. Was über die Kapazität eines Agenten hinaus geht, wird in mehrere Cluster aufgeteilt. Zwischen den Clustern können, völlig transparent für die Anwendungen, Lastinformationen, Aufträge und Daten ausgetauscht werden. Dahinter steht die Idee, daß, solange Anwendungen innerhalb eines Clusters vernünftig ablaufen können und zwischen verschiedenen Clustern keine allzu großen Lastdifferenzen auftreten, alles zentral und somit effizient abgewickelt wird. Cluster-übergreifende Aktionen sind mit zusätzlichem Aufwand verbunden und treten nur bei groben Auslastungsdifferenzen oder sehr großen Aufträgen bzw. Auftragsgruppen auf.

Die Cluster bilden eine beliebig vernetzte Struktur. Die Lastbalancierungsagenten direkt benachbarter Cluster tauschen periodisch Lastinformationen und Aufträge untereinander aus. An keiner Stelle im System liegen globale Lastinformationen über das System vor, und nirgendwo werden globale Lastbalancierungsentscheidungen für das Gesamtsystem getroffen. Es wird keine funktional hierarchische Struktur verwendet, sondern die Agenten der Cluster kooperieren auf derselben Ebene, d.h. haben dieselbe Blickweite und dieselben Aufgaben.

Sowohl explizite als auch implizite Verfahren der dezentralen Lastbalancierung (Abschnitt 2.5.3.4) können eingesetzt werden. Bei expliziter Verteilung werden zwischen Clustern dezentrale Lastbalancierungsverfahren eingesetzt; Lastbalancierer tauschen periodisch Informationen aus und verschieben Aufträge an Nachbar-Cluster, die weniger belastet sind. Bei impliziter Verteilung kann jeder Nachbar-Cluster wie ein eigener lokaler Server mit Aufträgen bedacht werden und wird durch ähnliche Lastgrößen charakterisiert wie ein lokaler Server. Die im *HiCon*-Modell aufgrund von Erfahrungen mit der Balancierungsumgebung unter verschiedenartigen Anwendungstypen und Systemkonfigurationen favorisierte Struktur wird in Abschnitt 3.7.3 beschrieben; Vergleiche zwischen expliziten und impliziten Ansätzen wurden in [Beck94c] durchgeführt.

3.7 Aufbau und Ablauf der Lastbalancierung

Die hier vorgestellte Lastbalancierung besteht logisch aus drei Komponenten (vergleiche Abschnitt 2.5.2). Die erste dient der Informationssammlung, die zweite trifft Entscheidungen und die dritte paßt die Lastbalancierungsstrategie an. Es wird hier nicht explizit zwischen Transfer- und Lokationsstrategie unterschieden. Der Entscheidungs-

algorithmus geht stets in der Reihenfolge vor, daß zuerst bestimmt wird, welche Aufträge wann zuzuweisen sind (Transferstrategie), und dann entschieden wird, wohin sie zuzuweisen sind (Lokationsstrategie). Abbildung 18 skizziert, wie eine Lastbalancierungskomponente durch Ereignisse über das Laufzeitsystem angestoßen wird und reagiert und welche Abläufe in welchen Komponenten der Balancierung angesiedelt sind. Links wird symbolisiert, wie ein Client Vorankündigungen über kommende Auftragsgruppen abgeben kann, woraufhin die Informationssammlung der Lastbalancierung Prioritäten für Aufträge zwischenspeichern kann. Wenn ein Client oder ein benachbartes Cluster einen Auftrag zur Bearbeitung schickt, wird dieser von der Entscheidungskomponente zunächst in eine zentrale Auftragswarteschlange eingereiht. Zugleich wird ein Bewertungs- und Zuweisungsvorgang angestoßen, infolgedessen Aufträge zugewiesen werden können, wobei die betroffenen Zustände in der Informationssammlung aktualisiert werden. Resultate bearbeiteter Aufträge, Lastmessdaten und Auslastungsdaten von Nachbarclustern werden in der Informationssammlungskomponente ausgewertet, und je nach Situation kann jeweils wiederum ein Bewertungs- und Zuweisungsvorgang angestoßen werden. Zusätzlich werden die Zustandsinformationen in der Adaptionskomponente verwendet, um Regelgrößen und Vorabschätzungsgrößen für zukünftige Entscheidungen zu erzeugen bzw. zu justieren. Rechts im Bild ist schließlich die Involvierung der Lastbalancierung in die Datenverwaltung skizziert, wobei die Informationssammlung grobe Zustandsdaten über Aufenthaltsorte, Zugriffscharakteristik und Zugriffsaufwand erhält. Auch hier kann die Adaptionskomponente längerfristige Auswertungen der Informationen durchführen.

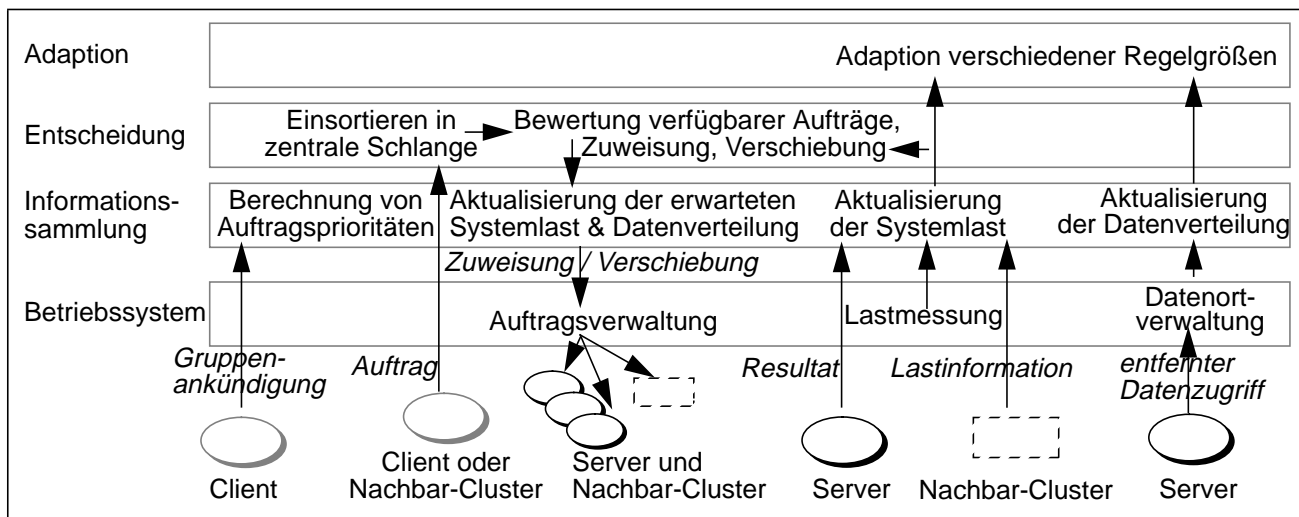


Abbildung 18: Aktivierung und interner Ablauf von Betriebssystem und Lastbalancierung.

3.7.1 Informationssammlung

Die Informationssammlungs-Komponente besteht aus Rückrufprozeduren, die durch verschiedene Ereignisse vom Betriebssystem aktiviert werden und neue Informationen in die Datenstrukturen der Lastbalancierung eintragen. Am Ende des Abschnitts sind alle Informationen und Parameter für die Lastbalancierung noch einmal zusammengestellt. Zur Lastbalancierung relevante Ereignisse sind im Folgenden aufgelistet:

- *Vorankündigung von Auftragsgruppen.* Um die Beziehungen zwischen Aufträgen innerhalb einer Anwendung zur Lastbalancierung nutzen zu können, können Clients Vorabschätzungen über die Gesamtstruktur einer anstehenden Gruppe von Aufträgen angeben. Auf eine Vorankündigung hin berechnet die *HiCon*-Lastbalancierung Prioritäten für die Aufträge der Gruppe (Abschnitt 3.7.4). Die Vorabschätzungen beschränken sich im vorliegenden Modell auf die Angabe von Reihenfolgebeziehungen zwischen Aufträgen und des Ressourcenbedarfs der einzelnen Aufträge. Kommunikationsbeziehungen bzw. Zugriffsmuster auf gemeinsame Daten werden zu diesem Zeitpunkt noch nicht benötigt, denn die *HiCon*-Lastbalancierung erhält und nutzt sie erst bei Eingang der einzelnen Aufträge. Die Aufträge werden einfach durch anwendungsdefinierte Namen innerhalb einer Anwendung spezifiziert. Die Lastbalancierung erkennt vorangekündigte Aufträge später anhand des Namens wieder, wenn sie vom Client abgeschickt werden. Die Vorabinformationen müssen weder vollständig noch korrekt sein, und die Clients sind für die Einhaltung der Reihenfolgebeziehungen selbst zuständig: abgesandte Aufträge sind sofort ausführbar.
- *Eintreffen neuer Aufträge.* Ein solches Ereignis tritt jedesmal ein, wenn ein Client oder ein benachbartes Cluster einen Auftrag (Serverklassenaufruf) gesandt hat. Die Lastbalancierung registriert, daß ein ausführbarer Auftrag mehr im System ist, der noch beliebig an einen Server der angegebenen Klasse zugewiesen werden kann, und sie notiert die Ankunftszeit des Auftrags. Weiterhin werden eventuelle Vorabschätzungen des Client über das Auftragsprofil ausgewertet. Wenn der Client den Auftrag mit einem Namen versehen hat, so wird in einer Tabelle nachgesehen, ob für diesen Auftrag bereits Informationen vorliegen, die etwa im Rahmen einer Auftragsgruppenvorankündigung abgespeichert wurden. Vorabschätzungen des Clients über das Auftragsprofil können folgendes enthalten: den Integer- und Fließkomma-Rechenaufwand des Auftrags, den Ein-/Ausgabe-Ressourcenbedarf, den Speicherbedarf und schließlich Datenreferenzmuster. Abschätzungen über Datenreferenzen sind Aufzählungen bestimmter Datensatznamen oder -Namensbereiche. Dabei kann zu jedem Datensatzname bzw. Namensbereich eine Wahrscheinlichkeit für exklusiven Zugriff angegeben werden.

Der eingetroffene Auftrag wird gemäß seiner Priorität in die zentrale Auftragswarteschlange einsortiert. Die Priorität wird entweder aufgrund einer vorhergehenden Gruppeneinplanung (Abschnitt 3.7.4) bestimmt und ist ansonsten bei Einzelaufträ-

gen proportional zu ihrer geschätzten Auftragsgröße. Priorität wird in Instruktionen gemessen. Die Prioritäten der Aufträge wachsen linear mit der Zeit, die sie in der zentralen Warteschlange stehen. Wird daher ein neuer Auftrag a einsortiert, dann wird die Priorität der anderen Aufträge beim Vergleich um die Rechenzeit erhöht, die sie in der Zwischenzeit in der Warteschlange ‘verpaßt’ haben. Dazu wird eine mittlere Systemleistung zugrundegelegt:

Falls a innerhalb einer Gruppe vorangekündigt war, gilt

$$priority_a = announcedPriority_a \times computeTimeAdapt_{cs} ,$$

sonst

$$priority_a = instructions_a \times computeTimeAdapt_a .$$

Stammt a von einem Nachbarcluster, so wird die dort ‘verpaßte’ Rechenzeit

$$t_{spentInCentralQueues}(a) \times avg_k(MFLOPS_k)$$

addiert. Schließlich wird a hinter dem ersten Auftrag b eingefügt, der - inklusive seiner durch Wartezeit gestiegenen Priorität - eine höhere Priorität hat:

$$priority_a \leq priority_b + t_{spentInCentralQueues}(b) \times avg_k(MFLOPS_k) .$$

Der Korrekturfaktor $computeTimeAdapt_{cs}$ der Serverklasse c und des Auftragstyps s dieser Klasse wird adaptiv geregelt (Abschnitt 3.7.5).

- *Änderungen im Arbeitszustand von Servern.* Jedesmal, wenn Server einen Auftrag abgeschlossen haben, schicken sie ein Resultat zum Client zurück. Dabei geben sie der Lastbalancierung mit dem Ergebnis automatisch einige Informationen über den Bearbeitungsverlauf des Auftrages und die verbleibende Last (Bearbeitungszeit) an Aufträgen in ihrer lokalen Auftragswarteschlange $t_{remainingWork}(S)$ mit. Informationen über den Verarbeitungsverlauf enthalten die Laufzeit im Server, Wartezeiten durch Datenkommunikation und eventuelle Leerlaufzeiten des Servers zwischen Aufträgen. Die Adaption regelt anhand der vom Server mitgelieferten Informationen die Größen $cpuUtilAdapt_{cs}$, $computeTimeAdapt_{cs}$ und $dataTimeAdapt_{cs}$ nach (Abschnitt 3.7.5).
- *Änderungen in der Ressourcen-Auslastung.* Das Betriebssystem mißt periodisch die Auslastung der Ressourcen auf den Rechenknoten. Wenn sich die Auslastung einer Ressource (d.h. einer der Kenngrößen gegenüber der vorherigen Meldung) signifikant geändert hat, so wird es der Lastbalancierung als Ereignis gemeldet. Als Ressourcen werden hier die Prozessorauslastung (CPU-Nutzung und mittlere Anzahl laufbereiter Prozesse), der Hauptspeicherbedarf (Seiteneinlagerungsrate durch Speicherüberlastung), die Plattennutzung und die Nachrichtenlast auf Netzverbindungen zwischen den Knoten angesehen.
- *Bewegung von Datensätzen und Verteilung von Datenkopien.* Gemeinsame Datensätze werden vom Betriebssystem jeweils zu dem Server transportiert, der auf sie

zugreifen möchte (Abschnitt 3.4). Die Entstehung neuer Datensätze, die Migration und die Kopienverteilung von Daten sind jeweils Ereignisse, woraufhin die Informationssammlung ihre Tabellen *ownerS*, *ownerCl* und *hasCopy* über die vermutliche Datenverteilung im System aktualisiert sowie die Adaptionen *dataCommCost_{cd}*, *dataCommCostRemote_{cd}* und *dataReadWrite_{cd}* justiert (Abschnitt 3.7.5).

- *Zustandsänderungen in einem Nachbar-Cluster.* Im *HiCon*-Modell kooperieren jeweils die Balancierungsagenten benachbarter Cluster (Abschnitt 3.6). Ein Cluster ist ein Teilsystem, für das ein zentraler Lastbalancierungsagent konfiguriert wurde. Ebenso kann die Nachbarschaftstopologie zwischen Clustern konfiguriert werden. Die Informationssammlung eines Clusters verwaltet daher aggregierte Lastinformationen über ihr Cluster. Bei signifikanten Änderungen im Lastzustand des Clusters benachrichtigt sie die benachbarten Lastbalancierungsagenten. Beim Eintreffen von Information über ein Nachbar-Cluster *cl* wird die geschätzte Anwendungslast *applLoad_{cl}* und die verfügbare Gesamtrechenleistung $\Sigma MFLOPS_k$ des Nachbar-Clusters aktualisiert (Abschnitt 3.7.3).
- *Änderung in der Auslastung der Lastbalancierung selbst.* Wie in Abschnitt 2.4.7 erklärt, sieht die Lastbalancierung im *HiCon*-Modell sich selbst ebenfalls als Teil des Systems. Die Informationssammlung verfolgt daher auch die Rechenlast und die Verzögerungen, die durch die Lastbalancierungsfunktion entstehen. Daraufhin kann die Lastbalancierung ihre Strategie anpassen (Abschnitt 3.7.5).

Die Informationssammlung erhält also durch jedes Ereignis neue Vorab- oder Meßinformationen über die Lastsituation im System und den Verarbeitungsverlauf der Anwendungen. Sie kann auf jedes Ereignis hin die Entscheidungskomponente aktivieren. Erfahrungen haben gezeigt, daß es genügt, bei Ankunft und Fertigstellung von Aufträgen den Entscheidungsalgorithmus zu aktivieren. Die Informationssammlung aktualisiert zumindest stets die Informationsstrukturen, auf denen die Balancierungsentscheidungen basieren. Die Informationstypen, die im *HiCon*-Modell Verwendung finden, werden im folgenden zusammengestellt. Die Vielzahl an Informationstypen erweckt zunächst den Eindruck, daß die Lastbalancierung sehr großen Aufwand treibt, um die Lastsituation und den Verlauf der Anwendungen akkurat zu verfolgen. Bei genauerer Betrachtung müssen jedoch fast keine Informationstypen separat gewonnen oder gespeichert werden, sondern sie sind im verteilten Betriebssystem ohnehin verfügbar und verursachen auch keine zusätzlichen Aktualisierungsnachrichten.

- *Informationen über Rechenknoten.* Zu jedem Rechenknoten *k* werden die Anzahl der Prozessoren *numberOfProcessors_k* samt ihren theoretischen Rechenleistungen für Integer- und Fließkomma-Operationen *MIPS_k*, *MFLOPS_k* sowie die Leistungen der sonstigen Ressourcen (Hauptspeicher, E/A-Geschwindigkeit) gespeichert. Das sind statische Werte, die durch die Lastbalancierung beim Start mittels kurzer Benchmarks bestimmt werden. Dynamische Laufzeitinformationen sind die Anzahl

der auf den Knoten konfigurierten Server und die aktuelle Belastung der Prozessoren in Form der *run queue length* (Abschnitt 2.4). Die Speicherbelastung des Knotens wird durch die Speichereinlagerungsrate gemessen (Abschnitt 2.4). Die Informationen über Rechenknoten dienen zur Abschätzung, wie schnell ein weiterer Auftrag bearbeitet würde, wenn er auf dem Knoten plazierte würde; sie werden nicht - wie in den meisten Ansätzen zur dynamischen Lastbalancierung - dazu verwendet, um die Last zwischen Knoten zu vergleichen und Ausgleich zu schaffen, da dies im vorliegenden Ansatz nicht primäres Ziel ist (Abschnitt 3.2).

- *Informationen über Netzverbindungen.* Zwischen den Knoten könnte statisch pro Kanal die Kapazität und die Verzögerungszeit gespeichert werden. Dynamisch könnte die Anzahl der Nachrichten pro Zeiteinheit verfolgt werden. Je nach Betriebsprotokoll des Kanals könnte auch die relative Nutzungsdauer oder Anzahl der kollidierenden Nachrichtenpakete wichtig sein. Informationen über Netzverbindungen dienen dazu, die bei Datenkommunikation zu erwartenden Verzögerungen abzuschätzen. Die Minimierung der Kommunikation ist jedoch kein primäres Ziel der Lastbalancierung; das Ziel ist der größtmögliche Durchsatz der Anwendungen. Im *HiCon*-Modell werden Leistung und Auslastung der Netzverbindungen derzeit nur indirekt durch Beobachtung der Datenkommunikationskosten auf Datentyp-Ebene berücksichtigt (siehe Informationen über Aufträge und Auftragsgruppen).
- *Informationen über Serverklassen und Auftragstypen.* Pro Serverklasse wird die Anzahl und Verteilung der konfigurierten Server gespeichert. Die adaptive Lastbalancierung verwaltet Informationen über Serverklassen c und deren Subklassen s (Auftragstypen): den mittleren CPU-Nutzungsanteil $cpuUtilAdapt_{cs}$, den Auftragsrechenzeit-Korrekturfaktor $computeTimeAdapt_{cs}$ und den Auftragsdatenkommunikations-Korrekturfaktor $dataTimeAdapt_{cs}$.
- *Informationen über Server.* Zu jedem Server s wird als statische Information verwaltet, zu welcher Serverklasse er gehört und auf welchem Rechenknoten er konfiguriert ist. Dynamische Informationen sind die Anzahl der Aufträge in der lokalen Warteschlange sowie die Summen der Ressourcenbedürfnisse über die Aufträge in der lokalen Warteschlange. Diese Informationen dienen einerseits zur Abschätzung, welche Last der Server auf seinem Rechenknoten in Zukunft noch erzeugen wird, andererseits der Abschätzung, wieviel Zeit $t_{remainingWork}(s)$ noch vergeht, bis ein weiterer Auftrag zur Bearbeitung an die Reihe käme, wenn er nun dem Server zugewiesen würde. Weiterhin wird die Zeit gespeichert, seit wann der Server an seinem derzeitigen Auftrag arbeitet, um abzuschätzen, wieviel Zeit er noch dafür benötigt. Hinterher kann dabei auch beurteilt werden, um wieviel die tatsächliche Bearbeitungszeit von der vorabgeschätzten abgewichen ist. Außerdem wird für jeden Server die für einen weiteren Auftrag a momentan bzw. zum Zeitpunkt t verfügbare Rechenkapazität $procPower_k(a,s,t)$ auf dem Knoten verwaltet.

Bei implizit verteilter Lastbalancierung (Abschnitt 2.5.3.4) betrachtet die Lastbalancierung zwar die Nachbar-Cluster als besondere Server auf besonderen Rechenknoten, aber die Informationen über diese Server bzw. Knoten sind aggregierte Informationen über ganze Serverklassen bzw. über die Rechenknoten eines ganzen Clusters.

- *Informationen über Anwendungen.* Zwischen Clustern werden nur jeweils die Aufträge ganzer Anwendungen ausgetauscht. Es wird also entschieden, alle noch nicht zugewiesenen und alle noch folgenden Aufträge einer Anwendung bis auf weiteres an ein anderes Cluster abzugeben. Dazu sind Informationen über Anwendungen A notwendig. So werden die vermutlichen Datenaustauschkosten $migDataCost_A$ zwischen Clustern verwaltet. Diese Größe schätzt die Summe der Wartezeiten zum Verschieben aller momentan in der Anwendung aktiven globalen Daten ($activeGlobalData_{Ad}$) ab. Pro Anwendung wird auch die verbleibende Restlaufzeit der Anwendungen $remainingProcessingTime_A$, abgeleitet aus den Vorabschätzungen des Client, verwaltet. Schließlich werden die seit der letzten Verschiebung der Anwendung ins Cluster abgeleisteten Auftragsbearbeitungszeiten $t_{localProcessing}(A)$ protokolliert.
- *Informationen über Aufträge und Auftragsgruppen.* Für einzelne Aufträge a werden Vorabinformationen gespeichert, die als Abschätzungen des Client beim Aufruf mitgegeben werden bzw. daraus abgeleitet werden. Darunter zählen der vermutliche Rechenaufwand $instructions_a$ und der Bedarf an sonstigen Ressourcen, wie Hauptspeicher oder Plattenein- / -ausgabe. Der Rechenaufwand wird in der Einheit 'Instruktionen' abgeschätzt, da die resultierende Rechenzeit von der jeweiligen Prozessorleistung und -belastung abhängig ist. Der Anteil an Fließkomma- und Integer-Rechnung $floatPortion_a$, $intPortion_a$ kann ebenfalls angegeben werden. Weiter können Datenzugriffsmuster durch Auflistung von Namensbereichen $dataRefRange_{1..N}(a)$ angegeben werden. Zu jedem Datenbereich kann eine Wahrscheinlichkeit $dataRangeWriteProb_i(a)$ dafür spezifiziert werden, daß im Rahmen des Auftrags auf diese Daten exklusiv zugegriffen wird. Zu jedem Auftrag wird weiterhin die Zeit $t_{spentInCentralQueues}(a)$ gespeichert, die er seit seiner Entstehung in zentralen Warteschlangen verbracht hat. Schließlich kann die Lastbalancierung erwartete Ausführungszeiten $t_{compute}(s,a)$ und Datenwartezeiten $t_{dataComm}(s,a)$ auf bestimmten Servern unter den aktuellen Bedingungen speichern, die sie ausgerechnet hat. Über vollendete Aufträge sind die tatsächlich benötigten Zeiten $t_{computeReal}$, $t_{dataReal}$ verfügbar.

Clients können innerhalb einer Anwendung Abhängigkeiten zwischen zukünftigen Aufträgen einer Gruppe angeben. Die Angabe der Nachfolgeaufträge spezifiziert einen Graph der vermuteten Reihenfolge-Beziehungen zwischen den Aufträgen einer Gruppe. Für vorangekündigte Aufträge, die noch nicht als ausführbare Aufträge eingetroffen sind, werden Informationen über die vermutete Auftragsgröße

(Instruktionen) und eine Liste von Nachfolgeaufträgen gespeichert. Durch den Planungsalgorithmus der Lastbalancierung können dazu noch Prioritäten *announced-Priority_a* beigefügt werden (Abschnitt 3.7.4). Aus diesen kann, zusammen mit der bisherigen Wartezeit in zentralen Warteschlangen, eine dynamische Priorität *priority_a* für einen Auftrag abgeleitet werden. Kommunikationsaufkommen innerhalb einer Auftragsgruppe wird lediglich durch die Datenreferenzabschätzungen der einzelnen Aufträge berücksichtigt.

- *Informationen über globale Datensätze und Datentypen.* Von gemeinsamen Datensätzen liegen jeweils Informationen vor, welcher Server *ownerS(dataRef)* momentan im “Besitz des Originals” ist (d.h. für den Datensatz *d* zuständig ist), und welche Server *s* über gültige Kopien verfügen, *hasCopy(s,dataRef)*. In dezentralen Strukturen ist die genaue Datenverteilung nur innerhalb des Clusters bekannt, und außerdem die Information, bei welchen Nachbar-Clustern *ownerCl(dataRef)* Originale oder Kopien der Datensätze, *hasCopy(cl,dataRef)*, zu suchen sind. Die Informationen der Lastbalancierung spiegeln dabei nicht den aktuellen, sondern den erwarteten Zustand wieder, da sie bei Auftragszuweisungen bzw. Auftragsverschiebungen an Nachbar-Cluster bereits vorab gemäß der Datenreferenzabschätzungen der Clients aktualisiert werden. Weiterhin wird pro Serverklasse *c* und Datentyp *d* der mittlere Aufwand zum entfernten Zugriff auf Daten, unterschieden nach Austausch innerhalb eines Clusters, *dataCommCost_{cs}*, und zwischen Clustern, *dataCommCostRemote_{cs}*, adaptiv ermittelt. Durch diese Informationen werden die Größe der Datensätze, die Sperrwartezeiten, die Geschwindigkeit und die Auslastung des Netzwerks berücksichtigt. Weiterhin wird die mittlere Anzahl aufeinanderfolgender Lesezugriffe zwischen zwei exklusiven Zugriffen *dataReadWrite_{cd}* je Datentyp adaptiv verwaltet.
- *Informationen über Cluster.* Die Informationssammlung einer Lastbalancierungskomponente verwaltet auch aggregierte Informationen über den globalen Zustand ihres Clusters und dieselben Informationen über die benachbarten Cluster (dezentrale Struktur). Darunter fällt die dortige Auftragslast *applLoad_{cl}*, d.h. die Anzahl der dort aktiven Anwendungen *activeAppls_{cl}* dividiert durch die Summe der Prozessorleistungen der Rechenknoten, die mittlere Prozessorbelastung der Rechenknoten und die Anzahl der insgesamt momentan arbeitenden Server. Weiterhin wird für das eigene Cluster die Anzahl der Aufträge in der zentralen Warteschlange *centralQueueSize* und die Summe der Auftragsgrößen (Ressourcenbedürfnisse) in den zentralen Warteschlangen mitgeführt.
- *Informationen über die Lastbalancierung.* Die Lastbalancierung kann sich selbst beobachten, indem sie die Anzahl der unverarbeiteten anstehenden Ereignisse *eventQueueSize* bzw. *eventQueueSize* (exponentiell geglättet) und die durch Lastbalancierungsberechnungen erzeugte Prozessorbelastung beobachtet. Diese Informationen dienen nicht unmittelbar Balancierungsentscheidungen, sondern der Adaption der

Strategie (Abschnitt 3.7.5), denn sie zeigen an, wann die Lastbalancierung überlastet wird bzw. übermäßig große Verzögerungen und Zusatzlast mit sich bringt.

Alle Informationen und Parameter für die Lastbalancierung sind im folgenden noch einmal zusammengestellt:

Parameter	Bedeutung
$activeAppls_{cl}$	Anzahl von Anwendungen, die im Cluster cl derzeit laufen (und nicht an andere Cluster verschoben sind).
$activeGlobalData_{Ad}$	Tabelle der Anzahl an benutzter Datensätze pro Datentyp d und Anwendung A .
$announcedPriority_a$	Vom Client angekündigte Priorität eines Auftrags. Dient der Lastbalancierung zur Initialisierung von $priority_a$.
$applLoad_{cl}$	Cluster-Belastung durch Anwendungen $activeAppls_{cl}$.
$centralQueueSize$	Anzahl der Aufträge in der zentralen Warteschlange eines Clusters.
$computeTimeAdapt_{cs}$	Adaptionsfaktor pro Serverklasse c und Auftragsstyp (Subklasse) s , der das Verhältnis zwischen den von Clients abgeschätzten Auftragsgrößen ($instructions_a$) und den tatsächlich beobachteten Auftragsgrößen angibt.
$cpuUtilAdapt_{cs}$	Adaptionsfaktor pro Serverklasse c und Auftragsstyp (Subklasse) s , der den mittleren CPU-Bedarf solcher Aufträge abschätzt.
$CPUrunQueueLen$	Momentane <i>run queue length</i> eines Knotens.
$dataCommCost_{cd}$	Adaptionsgröße pro Serverklasse c und Datentyp d , die die Wartezeit auf einen Datensatz angibt, wenn ein Server den Datensatz von einem anderen Server im Cluster besorgen muß.
$dataCommCostRemote_{cd}$	Adaptionsgröße pro Serverklasse c und Datentyp d , die die Wartezeit auf einen Datensatz angibt, wenn ein Server den Datensatz von einem anderen Server aus einem anderen Cluster besorgen muß.
$dataRangeWriteProb_{1..N}(a)$	Abschätzung des Clients, wie hoch die Wahrscheinlichkeit für exklusiven Zugriff auf die in $dataRefRange_{1..N}(a)$ angegebenen Datenbereiche bei Bearbeitung von Auftrag a ist.

$dataReadWrite_{cd}$	Adaptionsgröße pro Serverklasse c und Datentyp d , die abschätzt, wieviel Lesezugriffe im Mittel zwischen zwei Schreibzugriffen auf solche Datensätze erfolgen.
$dataRef_{i,j}(a)$	Einzelne Datenreferenz im Bereich $dataRefRange$
$dataRefRange_{1..N}(a)$	Abschätzung des Clients, welche Liste von Datenbereichen Auftrag a referenzieren wird.
$dataTimeAdapt_{cs}$	Adaptionsfaktor pro Serverklasse c und Auftragsstyp (Subklasse) s , der das Verhältnis zwischen den Datenwartezeiten während der Ausführung solcher Aufträge (aufgrund von Clients angegebenen Datenreferenzmustern und der Datenverteilung abgeschätzt) und den tatsächlich beobachteten Datenwartezeiten je Auftrag angibt.
$d_{overrate}$	Faktor, mit dem die vermuteten Datenwartezeiten für Aufträge auf Servern künstlich überbewertet werden.
$eventQueueSize$	Anzahl der anstehenden, unverarbeiteten Ereignisse einer Latbalancierungskomponente
$\overline{eventQueueSize}$	Anzahl der anstehenden, unverarbeiteten Ereignisse einer Latbalancierungskomponente (exponentiell geglättet).
$floatPortion_a$	Anteil der Fließkommaoperationen in Auftrag a .
$hasCopy$	Tabelle, die vermutliche Kopienbesitzer (Server) von Datensätzen enthält.
$instructions_a$	Vom Client abgeschätzte Auftragsgröße, die sich aus Integer- und Flieskommaoperationen anteilig zusammensetzt ($floatPortion_a$, $intPortion_a$)
$intPortion_a$	Anteil der Integer-Operationen in Auftrag a .
$maxTasksConsidered$	Anzahl betrachteter Aufträge in der zentralen Warteschlange für Zuweisungsentscheidungen.
$MFLOPS_k$	Bauartbedingte Fließkommarechenleistung des Knotens k (Millionen Instruktionen pro Sekunde).
$migDataCost_A$	Vermutlich benötigte Zeit, um die aktiven Daten ($activeGlobalData_A$) der Anwendung A in ein Nachbar-Cluster zu migrieren, falls A dorthin verschoben wird.
$MIPS_k$	Bauartbedingte Integer-Rechenleistung des Knotens k (Millionen Instruktionen pro Sekunde).

$numberOfProcessors_k$	Prozessorzahl des Knotens k .
$ownerCl$	Tabelle, die die vermutlichen derzeitigen Besitzer-Cluster von Datensätzen enthält.
$ownerS$	Tabelle, die die vermutlichen derzeitigen Besitzer (Server im Cluster) von Datensätzen enthält.
$priority_a$	Priorität eines Auftrags a in der zentralen Warteschlange. Gemessen in Instruktionen. Bedeutung: Länge des kritischen Pfades, der nach dem Auftrag folgt (inklusive des Auftrags selbst) zuzüglich der Instruktionen, die mittlerweile hätten abgearbeitet werden können, während der Auftrag stattdessen in der Warteschlange lag.
$procPower_k(a,s,t)$	Zum Zeitpunkt t vermutlich verfügbare Gesamtrechenleistung beim Server s für Auftrag a .
$remainingProcessingTime_A$	Vermutete Restlaufzeit der Anwendung A .
$servers_k$	Liste der Server auf Knoten k .
$t_{advance}$	Zeitspanne, für die die Lastbalancierung im voraus Knoten auszulasten versucht.
$t_{compute}(s,a)$	Erwartete Ausführungszeit von Auftrag a auf Server s .
$t_{computeReal}$	Real gemessene Rechenzeit eines Auftrags.
$t_{dataAccess}(s,a,i,j)$	Vermutete Wartezeit durch Datenzugriff auf $dataRef_{i,j}(a)$ bei Server s für Auftrag a .
$t_{dataComm}(s,a)$	Erwartete Datenwartezeit von Auftrag a auf Server s .
$t_{dataReal}$	Real gemessene Datenwartezeit eines Auftrags.
$t_{localProcessing}(A)$	Seit der letzten Verschiebung der Anwendung A zwischen Clustern im Cluster abgeleistete Auftragsrechenzeit.
$t_{remainingWork}(s)$	Vermutete Restzeit, die Server s momentan benötigt, um seine Auftragswarteschlange abzuarbeiten.
$t_{spentInCentralQueues}(a)$	Zeit, die Auftrag a bisher in zentralen Warteschlangen verbracht
w	Momentane Eignung einer Anwendung zur Verschiebung in ein Nachbar-Cluster.

3.7.2 Entscheidung: Bewertung und Zuweisung

Die Entscheidungskomponente der Lastbalancierung bestimmt, wann und wohin welche Aufträge oder Daten zugewiesen werden.

Auftragsplazierung meint im *HiCon*-Modell die Zuweisung von Aufträgen, die ausführbar in der zentralen Warteschlange liegen, an Server. Nach der Zuweisung an einen Server ist keine Migration von Aufträgen mehr zulässig. An Nachbarsysteme verschobene Aufträge können allerdings dort noch beliebig zugewiesen, weiterverschoben oder zurückverschoben werden (Abschnitt 3.7.3).

Die Plazierung, Migration und Replikation gemeinsamer Daten wird nicht durch separate Lastbalancierungsaktionen angestoßen. Daten liegen zuerst bei dem Server, der sie erzeugt hat. Durch lesende Zugriffe anderer Server verschickt das verteilte Betriebssystem Kopien, durch Änderungszugriffe wandern die Datensätze zu den entsprechenden Servern. Obwohl die Lastbalancierung prinzipiell direkt auf die Datenverteilung Einfluß nehmen könnte, wurden aus Zeitgründen im *HiCon*-Projekt bislang nur Strategien realisiert, die Aufträge unter Berücksichtigung der aktuellen Datenverteilung und des vermuteten Aufwandes für die notwendigen (automatisch folgenden) Datenumverteilungen zuweisen. Die Datenverteilung wird also mittelbar beeinflusst, indem die Aufträge Daten zu dem Server ziehen, auf dem sie bearbeitet werden.

Die Entscheidungskomponente wird durch die Informationssammelkomponente aktiviert. Ereignisse aktivieren ja zuerst die Informationssammelkomponente (Abschnitt 3.7.1), die neue Informationen extrahiert und einordnet und dann eventuell die Entscheidungskomponente aufruft. Wenn die Entscheidungskomponente aktiviert wird, so betrachtet sie die Systemsituation und greift gegebenenfalls durch Zuweisung verfügbarer Aufträge aus der zentralen Warteschlange ein. Die Zuweisung von Aufträgen kann zu beliebigen Zeitpunkten erfolgen, denn die Server haben lokale Auftragswarteschlangen, die sie der Reihe nach abarbeiten (Abschnitt 3.5).

Der im folgenden beschriebene Zuweisungsalgorithmus ist nur vom groben Ablauf her für den *HiCon*-Lastbalancierungsansatz bindend; die Detailberechnungen sind lediglich als eine Möglichkeit zu verstehen, der sich bewährt hat. In verschiedenen Veröffentlichungen über das *HiCon*-Modell wurden unterschiedliche Algorithmen untersucht. Keiner der Algorithmen setzt alle Lastinformationen ein, die das Lastbalancierungskonzept zur Verfügung stellt. Je nach Größe und Struktur des Rechnersystems und abhängig vom Typ der betrachteten Anwendungen sind im Entscheidungsalgorithmus unterschiedliche Informationen relevant. Durch automatische Adaption (Abschnitt 3.7.5) soll stets ein passender effizienter Entscheidungsalgorithmus eingestellt werden.

Die verfügbaren Aufträge in der zentralen Warteschlange nach ihrer Priorität sortiert. Ohne Anmeldungen von Auftragsgruppen entspricht das der zeitlicher Ankunftsreihenfolge (Abschnitt 3.7.1).

Der Entscheidungsalgorithmus bestimmt zunächst, welche Strategie (siehe unten) momentan zu verwenden ist: er schaltet auf eine einfache Überlaststrategie um, sobald $eventQueueSize > 4$ wird, und zurück auf die komplexe Strategie, sobald $eventQueueSize < 2$ wird (Abschnitt 3.7.5). Danach wird bestimmt, wie groß momentan die Zeitspanne $t_{advance}$ zu wählen ist, für die Knoten im voraus mit Aufträgen zu beladen sind (Abschnitt 3.7.5). Weiterhin wird eingestellt, ob derzeit mehr auf Durchsatz oder auf Antwortzeit hin optimiert werden soll: wenn die Knoten gut ausgelastet sind, d.h. z.B. höchstens einer unterbelastet ist, dann werden die Datenkommunikationszeiten durch einen Faktor, proportional zur Systemlast und zur Belastung der Balancierungskomponente, überbewertet:

$$d_{overrate} = 1 + centralQueueSize + eventQueueSize .$$

Dieser Faktor wird weiter unten verwendet. Während bei normaler Berücksichtigung der Datenwartezeiten vorrangig die Antwortzeiten der einzelnen Aufträge minimiert werden, bewirkt eine Überbewertung der Datenkommunikation, daß unproduktive Netzlast und Leerlaufzeiten durch Datenkommunikation reduziert werden, was den Gesamtdurchsatz im System steigert, aber für einzelne Aufträge nachteilig sein kann.

Nun betrachtet der Algorithmus nacheinander die zentral wartenden Aufträge, beginnend vom höchstpriorisierten, bis eine Maximalzahl $maxTasksConsidered$ von Aufträgen betrachtet wurde oder die zentrale Warteschlange leer ist. Aufträge a von derzeit an Nachbar-Cluster verschobenen Anwendungen A (Abschnitt 3.7.3) werden sofort an das Nachbar-Cluster cl geschickt, wobei ihre Verweilzeit in der zentralen Warteschlange $t_{spentInCentralQueues}(a)$ erhöht wird, und der bisher zum Nachbarn gesandte Auftragsumfang aktualisiert wird:

$$t_{localProcessing}(A) += instructions_a \times computeTimeAdapt_{cs} / avg_{k \text{ in Cluster}} MFLOPS_k ,$$

und die Tabellen der vermutlichen Datenorte gemäß der im Auftrag angegebenen Datenreferenzen (N Bereiche mit je N_i Datensätzen) aktualisiert werden:

$$ownerCl(dataRef_{i,j}(a)) = cl \text{ für alle } i=1..N, j=1..N_i .$$

Die übrigen Aufträge werden, falls noch Prozessoren vorhanden sind, die in naher Zukunft nicht mehr genügend ausgelastet sind, daraufhin untersucht, ob sie nun einem Server zugewiesen werden können. Durch diese Struktur des Entscheidungsalgorithmus verfolgt die Lastbalancierung das in Abschnitt 2.5.7 vorgestellte vierte Optimierungskriterium, wobei $t_{advance}$ anstelle der vermuteten Zeit bis zum Eintreffen des nächsten Auftrags verwendet wird. Jeder Auftrag wird folgendermaßen bewertet:

1. Bestimmung des momentan bestgeeigneten Servers S für den Auftrag a : Wenn die Überlaststrategie aktiv ist, so wird der Server gewählt, der bei der ersten Bewertung des Auftrags, d.h. bei seiner Ankunft, ermittelt wurde. Es wird also dann keine Neubewertung wartender Aufträge in der zentralen Warteschlange durchgeführt.

Wenn die komplexe Strategie aktiv ist, werden alle im Cluster konfigurierten Server s der betreffenden Klasse verglichen. Auch Server, die momentan beschäftigt sind, werden mitbetrachtet. Es wird der Server gewählt, der den Auftrag a (Klasse c , Auftragsstyp s) vermutlich zuerst abschließen könnte, wenn er ihn jetzt zugewiesen bekäme. Dabei werden die Rechenzeit für den Auftrag, die Wartezeit bis zur Beendigung der bereits beim Server wartenden Aufträge und die zu erwartenden Verzögerungen durch Zugriffe auf nicht lokal vorhandene Datensätze berücksichtigt:

$$S = \text{MIN}_s (t_{\text{compute}}(s,a) + t_{\text{dataComm}}(s,a) \times d_{\text{overrate}} + t_{\text{remainingWork}}(s)) \quad .$$

Die bei Server s zu erwartende Rechenzeit wird durch

$$t_{\text{compute}}(s,a) = \text{instructions}_a \times \text{computeTimeAdapt}_{cs} \times \text{procPower}_k(a,s,t_{\text{remainingWork}}(s))$$

abgeschätzt. Grundlage ist dabei die vom Rechenknoten k des Servers s zu erwartende Rechenleistung zu dem Zeitpunkt, an dem der Server vermutlich die Bearbeitung des Auftrags beginnen könnte. Die Rechenleistung wird gemäß

$$\text{procPower}_k(a,s,t) = \frac{\text{floatPortion}_a \times \text{MFLOPS}_k + \text{intPortion}_a \times \text{MIPS}_k}{\left(\sum_{i \in \text{servers}_k \wedge t_{\text{remainingWork}}(i) > t} \text{cpuUtilAdapt}_{c's'} + 1 \right) / \text{numberOfProcessors}_k}$$

abgeschätzt, wobei im Prinzip die theoretische Leistung des Prozessors, bzw. eines Prozessors bei Multiprozessor-Knoten, durch die Anzahl der sonstigen dann aktiven Server auf dem Prozessor - plus eins, da der betrachtete Server dann frei ist - geteilt wird. Bei Multiprozessor-Knoten wird die Anzahl der aktiven Server durch die Anzahl der Prozessoren dividiert. Ein Vierprozessor-Knoten zeigt beispielsweise bei drei arbeitenden Servern für einen weiteren Auftrag immer noch die volle Leistung eines Prozessors. Die theoretische Leistung der Prozessoren wird von der Lastbalancierung beim Start durch *Benchmarks* festgestellt und nach Integer- und Fließkomma-rechenleistung unterschieden. Anwendungen können bei Aufrufen die Anteile an Integer- und Fließkomma-Operationen der Aufträge abschätzen. Die Knotenleistung für den Auftrag wird nach diesen Anteilen gewichtet aufsummiert. In Abschnitt 3.7.5 wird erläutert, wie die Prozessorbelastung anhand des mittleren Prozessornutzungsanteils von Auftragsstypen adaptiv, durch längerfristige Rückkopplung mit der im System gemessenen Prozessorbelastung, justiert wird. c' ist die Serverklasse und s' der Auftragsstyp des bei Server s momentan bearbeiteten Auftrags.

Die Hauptspeicherüberlastung kann mitberücksichtigt werden, indem ab einer gewissen Speicherseiten-Einlagerungsrate die Knotenlast künstlich erhöht wird. Hierzu wurde jedoch keine Evaluierung durchgeführt.

Die vom Client beim Aufruf vermutete Auftragsgröße wird durch einen Adaptionsfaktor $computeTimeAdapt_{cs}$ je Auftragsstyp korrigiert (Abschnitt 3.7.5).

Zur Antwortzeit zählt weiterhin die Wartezeit $t_{dataComm}$ auf vermutlich benötigte Daten, die der Server nicht lokal vorliegen hat. Diese Zeit wird aufgrund der Datenreferenzvorabschätzungen des Client, der momentan vermuteten Datenverteilung und der in der letzten Zeit beobachteten Kommunikationskosten der betroffenen Datentypen d abgeschätzt. Pro vermuteter Datenreferenz $dataRef_{i,j}(a)$ entstehen folgende Kosten $t_{dataAccess}(s,a,i,j) =$

$$\begin{array}{ll} 0 & \text{falls } ownerCl(dataRef_{i,j}(a)) \in \text{Cluster} \\ & \text{und } ownerS(dataRef_{i,j}(a))=s, \\ dataRangeWriteProb_i(a) \times dataCommCost_{cd} & \text{falls } hasCopyS(dataRef_{i,j}(a))=s, \\ dataCommCost_{cd} & \text{falls } ownerCl(dataRef_{i,j}(a)) \in \text{Cluster}, \\ dataCommCostRemote_{cd} & \text{ansonsten.} \end{array}$$

Datenzugriffskosten werden nach Zugriffen innerhalb eines Clusters und Zugriffen zwischen Clustern getrennt. Für vermutete Lesezugriffe genügt es, wenn der Server eine Kopie der Daten besitzt, für Schreibzugriffe benötigt er das Original. Die Werte $dataCommCost$ werden adaptiv bestimmt (Abschnitt 3.7.5). Obige Formel wird zwar für die Abschätzung der Datenzugriffskosten verwendet, aber für den Vergleich der Antwortzeiten zwischen den Servern werden die Zugriffskosten auf Kopien um den adaptiv geregelten Faktor $dataReadWrite_{cs}$ geringer bewertet (Abschnitt 3.7.5), um das Anlegen mehrfach verwendbarer Kopien zu fördern.

Insgesamt ergibt sich für den Auftrag

$$t_{dataComm}(s, a) = dataTimeAdapt_{cs} \times \sum_{i=1}^N \sum_{j=1}^{N_i} t_{dataAccess}(s, a, i, j)$$

an Zeitbedarf für Datenkommunikation durch nicht-lokale Datenreferenzen, wobei $dataTimeAdapt_{cs}$ wiederum eine Adaptionsgröße ist (Abschnitt 3.7.5).

Schließlich zählt auch die Wartezeit $t_{remainingWork}(s)$, bis die schon beim Server wartenden oder in Ausführung befindlichen Aufträge erledigt sind, zur Antwortzeit des Auftrags auf diesem Server. Diese Zeit wird jeweils bei Zuweisung eines Auftrags aktualisiert (siehe unten). Die Zeit, die der Server bereits an seinem aktuellen Auftrag gearbeitet hat, kann abgezogen werden, da bekannt ist, wann er das letzte Resultat zurückgeschickt hatte. Da die Auftragsgrößen nur Abschätzungen seitens der Clients sind und die Ausführungszeiten nur durch heuristische Abschätzungen ermittelt werden, kann es passieren, daß ein Server an einem Auftrag länger rechnet als vermutet. Dann wird - für die Schätzung der Antwortzeit - angenommen, daß der Auftrag nun unverzüglich fertig wird: *if* $t_{remainingWork}(s) < 0$ *then* $t_{remainingWork}(s) = 0$.

2. Der Auftrag a wird nun dem Server S zugewiesen, sofern sein Knoten k für die nächste Zeit dadurch nicht überlastet wird:

$$\left(\sum_{(i \in servers_k \wedge t_{remainingWork}(i) > t_{advance})} cpuUtilAdapt_{c,s} + 1 \right) / numberOfProcessors_k + cpuUtilAdapt_{c,s} < 2 .$$

Dabei ist 2 ein Erfahrungswert mit der Bedeutung, daß Workstations bei durchschnittlich 2 laufbereiten Prozessen recht guten Durchsatz bieten. Ansonsten verbleibt der Auftrag in der zentralen Warteschlange und wird bei der nächsten Aktivierung des Entscheidungsalgorithmus erneut betrachtet. Da der Zuweisungsalgorithmus stets mehrere Aufträge betrachtet und jeweils ihrem bestgeeigneten Server zuzuweisen versucht, kann es passieren, daß viele Aufträge einem günstig erscheinenden Server zugeteilt werden sollen, dessen Prozessor aber momentan genügend lange ausgelastet ist. Während eines Laufs des Entscheidungsalgorithmus muß daher die Wartezeit an solchen Servern auch dann temporär erhöht werden durch

$$t_{remainingWork}(s) += t_{dataComm} + t_{compute} ,$$

wenn ein Auftrag dorthin möchte, aber jetzt nicht zugewiesen wird, weil der Prozessor ausgelastet ist. Dadurch wird bei der Betrachtung der nachfolgenden Aufträge berücksichtigt, daß schon andere Aufträge auf den Server kommen sollen.

Bei der Zuweisung wird der (seit der letzten Verschiebung von Anwendung A ins Cluster) lokal zugewiesene Auftragsumfang aktualisiert:

$$t_{localProcessing}(A) += t_{compute}(s,a) ,$$

und die Tabellen der vermutlichen Datenorte werden gemäß der im Auftrag angegebenen Datenreferenzen (N Bereiche mit je N_i Datensätzen) aktualisiert:

$$ownerCl(dataRef_{i,j}(a)) = \text{lokales Cluster und } ownerS(dataRef_{i,j}(a)) = S \text{ für alle } i=1..N, j=1..N_i .$$

Im Zuweisungsalgorithmus wird also die Bestimmung des bestgeeigneten Auftrag - Server- Paares in zwei Schritte zerlegt: Es wird jeweils zuerst der wichtigste Auftrag und danach der für ihn bestgeeignete Server bestimmt. Pro Auftrag entsteht, zur Einsortierung in die zentrale Warteschlange gemäß der Auftragspriorität (Abschnitt 3.7.4), einmaliger Aufwand, der höchstens proportional zur Anzahl der Aufträge in der zentralen Warteschlange ist, gewöhnlich aber vernachlässigbar ist, denn die meisten Aufträge werden direkt bei niedrigster Priorität eingereiht (da die Prioritäten der bereits in der Warteschlange liegenden Aufträge ständig wachsen). So wächst der Entscheidungsaufwand je Aktivierung des Algorithmus lediglich linear mit der Anzahl der Server pro Klasse multipliziert mit der Anzahl der Aufträge in der zentralen Warteschlange, wobei maximal die *maxTasksConsidered* höchst-priorisierten Aufträge in der zentralen Warteschlange betrachtet werden. In Überlastungssituationen, solange die einfache Strategie aktiv ist, sinkt der Balancierungsaufwand, da nur einmal pro Auftrag ein Vergleich aller Server durchgeführt wird.

Im Prinzip könnte zur Zuweisung jeweils das optimale Paar bestimmt werden (Betrachtung und Bewertung aller Zuweisungskombinationen Auftrag - Server). Dabei würde jedoch der Entscheidungsaufwand stark steigen, was sich durch die möglicherweise besseren Entscheidungen nicht auszahlt; Es wurden jedoch im *HiCon*-Modell bislang keine Vergleiche durchgeführt. Die Trennung und Reihenfolge der Paar-Bestimmung hat folgenden Hintergrund: Für die Auftragszuweisung ist es wichtig, daß die alten bzw. sehr wichtigen Aufträge baldmöglichst bearbeitet werden; daher wird zuerst der Auftrag gewählt. Danach kann der passende Server bestimmt werden. Eine umgekehrte Reihenfolge birgt die Gefahr, daß Aufträge sehr lange in der Warteschlange verbleiben, solange die Server andere verfügbare Aufträge als besser geeignet betrachten.

Die Netzwerkleistung und Netzwerkbelastung werden in der Informationssammlung und im Entscheidungsalgorithmus des *HiCon*-Konzepts nicht explizit berücksichtigt. Das Modell enthält explizit nur Verzögerungen durch Austausch von Daten, die in erster Linie, d.h. bei kurzen Synchronisations- und Informationsnachrichten, mit den Latenzzeiten des Netzwerks korrelieren. Für Nachrichten, die größere Datensätze zwischen Servern austauschen, spiegelt die Datenaustausch-Zeitverzögerung jedoch auch die effektiv momentan verfügbare Bandbreite wieder. Durch Einsatz der adaptiv geregelten Entscheidungsparameter $dataCommCost_{cs}$ (Abschätzung der zu erwartenden Datenzugriffszeit je Datensatz innerhalb eines Clusters und dasselbe für Zugriffszeiten zwischen Clustern) berücksichtigt die *HiCon*-Lastbalancierung implizit sowohl den effektiven Durchsatz als auch die Nachrichtenverzögerung der Netzwerke.

3.7.3 Entscheidung: Austausch von Anwendungen zwischen Clustern

Ein separater Entscheidungsalgorithmus erwägt und initiiert die Verschiebung ganzer Anwendungen zu Nachbar-Clustern. Dieser Entscheidungsalgorithmus kann ebenfalls durch verschiedene Ereignisse aktiviert werden, wie z.B. durch den Empfang einer Lastinformation von einem Nachbar-Cluster. Die Verschiebung einer lokalen Anwendung zu einem Nachbarn bedeutet, daß ab diesem Zeitpunkt alle Aufträge der Anwendung direkt an das Nachbar-Cluster geschickt werden. Clients selbst lassen sich ja im *HiCon*-Modell nicht migrieren, da in heterogenen Systemen keine automatische Prozeßmigration möglich ist. Anwendungsaustausch zwischen Clustern ist Sender-initiiert. Das Ziel-Cluster kann die Anwendungen bei Bedarf jederzeit wieder zurückverschieben, und kann ebenso Anwendungen, die von anderen Clustern an ihn geroutet wurden, weiter verschieben. Da die Aufträge einer verschobenen Anwendung weiterhin beim Client im Ursprungs-Cluster entstehen, werden sie ggfs. durch die Balancierungskomponenten mehrerer Cluster geschleust.

Das *HiCon*-Verfahren zum Anwendungsaustausch zwischen Clustern ist dezentral und recht simpel im Vergleich zur komplexen zentralen Intra-Cluster-Balancierung. Da die Netzverbindungen zwischen Clustern oft langsam sind, und weil nur wenig Informationen für feingranulare Balancierungsentscheidungen verfügbar sind, werden lediglich komplette (parallele) Anwendungen zwischen höher und geringer belasteten Clustern verschoben. Lastbalancierungskomponenten tauschen dazu gelegentlich aggregierte Statusinformationen über ihre Cluster aus. Im Gegensatz zu vielen bekannten dezentralen Ansätzen ist die Informationsaustauschfrequenz unkritisch, da nur wenig Information nach signifikanten Änderungen, d.h. wenn sich $applLoad_{cl}$ um einen bestimmten Faktor gegenüber dem zuletzt verschickten Wert änderte, verschickt wird. Bei großen Anwendungen treten diese Lastzustandsänderungen weniger häufig auf. Ansonsten kann die Informationsaustauschfrequenz bei hoher Belastung der Balancierung oder des Netzwerks reduziert werden. Bei hoher Gesamtlast, solange Cluster nicht völlig leer laufen, sorgen ja die Lastkontrollmechanismen der zentralen Balancierungskomponenten für optimale Ressourcennutzung innerhalb der Cluster, so daß ein grober Lastausgleich zwischen Clustern aufgrund weniger häufigen Informationen genügt.

Die Balancierungskomponente eines Clusters verschiebt dann eine Anwendung A zum minimal belasteten Nachbar-Cluster cl , wenn gilt

$$applLoad_{cl} < (activeAppls - (T + 1)) / \sum_k MFLOPS_k ,$$

wobei die Last eines Clusters, das die Knoten k enthält, durch

$$applLoad_{cl} = \frac{activeAppls}{\sum_k MFLOPS_k}$$

bestimmt wird.

Ein fixer Differenzschwellwert T bestimmt, ab welcher Lastdifferenz eine Anwendung an ein geringer belastetes Cluster abgegeben wird. Nun wird die bestgeeignete Anwendung A zum Verschieben ausgewählt: sie hat im Cluster entweder noch gar nicht gerechnet, oder hat den größten Eignungswert w gemäß Abschnitt 3.7.5. Ein weiterer Schwellwert bestimmt, wieviele Ressourcen die bestgeeignete Anwendung im Cluster konsumiert haben muß, bevor sie erneut verschoben werden darf, und welchen verbleibenden Ressourcenbedarf sie noch aufweisen muß. Das verhindert zu häufige Verschiebung von Anwendungen, d.h. vermeidet fruchtlose Zusatzbelastung, Wartezeiten und Datenverschiebungen, und verhindert, daß sehr kurze Anwendungen oder große Anwendungen kurz vor ihrem Abschluß noch verschoben werden. In Abschnitt 3.7.5 wird beschrieben, wie dieser Schwellwert adaptiv geregelt wird.

An benachbarte Cluster abgegebene Aufträge werden dort in die zentrale Warteschlange eingereiht, wobei zu ihrer Priorität die bisherige Wartezeit in zentralen Warteschlangen anderer Cluster addiert wird. Die hier vorgestellten Entscheidungsalgorithmen realisieren eine explizit dezentrale Balancierungsstruktur (Abschnitt 2.5.3.4), es wurden im *HiCon*-Modell jedoch auch implizite Strukturen untersucht und verglichen [Beck94c].

Eine wichtige Anforderung an Balancierungsmechanismen ist die Stabilität. Darunter versteht man im allgemeinen, daß die Wahrscheinlichkeit für krasses Fehlverhalten der Lastbalancierung gering ist, d.h. es wenige Situationen gibt, in denen die Lastbalancierung stark stört oder extrem schlechte Ressourcennutzung bewirkt. Der Stabilitätsbegriff erscheint meist in dezentralen Ansätzen und bewertet einerseits, ob unnötig viele bzw. unnötig häufig Aufträge zwischen Rechenknoten ausgetauscht werden und andererseits, ob unterbelastete Knoten Gefahr laufen, in Kürze von Aufträgen mehrerer höher belasteter Knoten überhäuft zu werden. Der unerwünschte Effekt, daß Aufträge zu oft zwischen Knoten hin und her verschoben werden, tritt auf, wenn alle Knoten gleichmäßig und ausreichend belastet, wenige aber überlastet sind. Außerdem ist der Effekt zu beobachten, wenn die Knotenlasten sehr schnell schwanken, so daß Migrationsentscheidungen schnell wieder obsolet werden. Zentrale oder hierarchisch strukturierte Lastbalancierungsansätze sind weniger anfällig gegenüber solchen Instabilitätserscheinungen. Die Stabilität wird im *HiCon*-Ansatz bei dezentraler Lastbalancierung durch drei Techniken angestrebt: Erstens verhindert die Lastkontrolle in der zentralen Lastbalancierung jedes Clusters, daß die Ressourcenbelastung im System durch viele ankommende Aufträge zu hoch wird, indem sie entsprechend viele Aufträge in der zentralen Warteschlange zurückhält. Zweitens werden nur große Anwendungen verschoben, die noch lange genug laufen, daß sich die Verschiebung lohnen kann. Drittens wird die häufige Verschiebung einer Anwendung auf ein vernünftiges Maß beschränkt, indem zwischen zwei Migrationen gewisse Mindestrechenzeiten der Anwendungen gefordert werden (Abschnitt 3.7.5).

3.7.4 Dynamische Einplanung von Auftragsgruppen

Um den Durchsatz von unkorrelierten konkurrierenden Aufträgen zu optimieren, genügt es, die einzelnen Aufträge isoliert und die Gesamtlasten auf den Rechenknoten zu betrachten. Vor allem in parallelisierten Anwendungen bestehen jedoch Abhängigkeiten zwischen den einzelnen Aufträgen, die bei der Lastbalancierung beachtet werden sollten. Abschnitt 2.5.4.2 begründet die Notwendigkeit und stellt die wichtigsten Planungsverfahren vor, wie sie in statischer Lastbalancierung eingesetzt werden. Die in Abschnitt 2.5.4.3 vorgestellten Ansätze zur Berücksichtigung der Kommunikation benötigen im vorliegenden Ansatz keine separaten Planungsalgorithmen; es genügt,

die Datenkommunikation, die ja durch Zugriffe auf gemeinsame Datensätze abläuft, isoliert für jeden Auftrag bei der Zuweisung zu berücksichtigen.

Vorwissen über Reihenfolgebeziehungen kann genutzt werden, um den Ablauf paralleler Anwendungen vorherzusehen. Gegenüber den statischen Ansätzen ergeben sich jedoch für die dynamische Lastbalancierung drei neue Probleme: Zum ersten sind die Reihenfolgebeziehungen nicht alle beim Start des Systems global vorgegeben, sondern es kommen zur Laufzeit hin und wieder Anwendungen in das System, die in die aktuelle Situation eingebunden werden müssen. Zum zweiten herrscht unkoordinierter Mehrbenutzerbetrieb, d.h. es sind Reihenfolgebeziehungen jeweils innerhalb von Gruppen von Aufträgen bekannt, wobei aber die verschiedenen Gruppen und andere Einzelaufträge unkorreliert und zu beliebigen Zeiten ablaufen. Das dritte Problem ist die Ungenauigkeit der angegebenen Reihenfolgebeziehungen; bei dynamischer Lastbalancierung wird davon ausgegangen, daß das System unvorhersehbar belastet wird und die Lastbalancierung daher durch Beobachtung des Ist-Verhaltens den Durchsatz optimieren muß. Vorabschätzungen von Clients über Auftragsgruppen sind daher ebenso wie Vorabschätzungen über einzelne Aufträge nur als Hinweise zu verstehen und müssen nicht zutreffen. Es können sogar andere, mehr oder weniger Aufträge sein und sie unterliegen womöglich nicht den vermuteten Reihenfolgebeziehungen. Trotz dieser drei Probleme soll der Ablauf korrekt sein und die Lastbalancierung durch flexible Lastverteilung zur Laufzeit Durchsatzsteigerung erbringen.

Im *HiCon*-Projekt wurde aufgrund der obigen Anforderungen ein Ansatz entwickelt, der eine flexible Vorplanung von Aufträgen unter Berücksichtigung von Reihenfolgebeziehungen ermöglicht, der zur Laufzeit agiert, dem Mehrbenutzerbetrieb gerecht wird und Ungenauigkeiten toleriert.

Bei Ankündigung einer Auftragsgruppe in Form eines gerichteten, Auftragsgraphen (Abschnitt 3.7.1) geben Clients die vermuteten Größen von Aufträgen (d.h. die Ressourcenbedürfnisse) und Reihenfolgebeziehungen zwischen ihnen an. Auftragsgruppen sind vom Client gewählte, beliebige Teilausschnitte einer Anwendung. Die Aufträge der Gruppe werden durch Namen identifiziert. Die Lastbalancierung kann nun durch verschiedene Verfahren Prioritäten zu den Aufträgen berechnen. Wenn Aufträge ankommen, die auf diese Weise vorangekündigt waren, so bekommen sie diese Priorität zugewiesen und können (je nach Strategie) entsprechend in die zentrale Warteschlange einsortiert werden. Dabei wird neben der Priorität auch das Alter der Aufträge berücksichtigt. Im *HiCon*-Modell wurden fünf verschiedene Verfahren untersucht, wobei als einzige Ressource die Prozessorrechenzeit (gemessen in Instruktionen) betrachtet wurde. Die Verfahren wurden bereits in Abschnitt 2.5.4.3 vorgestellt:

1. Keine Prioritätsberechnung. Vorankündigungen werden ignoriert. Die fehlende Vorplanung impliziert, daß ankommende Aufträge grundsätzlich nach ihrem Alter in die zentrale Warteschlange eingeordnet werden.
2. Die Priorität jedes Auftrags wird isoliert, proportional zu seiner vermuteten Größe, bestimmt.
3. Die Priorität jedes Auftrags ist proportional zu seiner Ebene, d.h. sie entspricht der mittleren Auftragsgröße der Gruppe multipliziert mit der Anzahl der Folgeaufträge auf dem längsten Pfad zu einem Endauftrag. Man beachte, daß diese Priorisierung nur scheinbar dem *highest level first scheduling* ähnelt. Jenes Verfahren teilt die Ebenen von den Startaufträgen her ein (die erste Ebene enthält die Aufträge ohne Vorgänger) und weist die Aufträge strikt Ebene für Ebene zu.
4. Die Priorität jedes Auftrags ist proportional zu seiner Auftragsgröße zuzüglich der Summe der Auftragsgrößen entlang des längsten Pfades zu einem Endauftrag.
5. Die Prioritäten werden wie oben bestimmt, erweitert um gewichtete Pfadlängen.

Man beachte, daß das vorgestellte dynamische Planungsverfahren im Gegensatz zu den statischen Lastbalancierungsverfahren den Aufträgen hier noch keinen Prozessor und noch keinen Zuweisungszeitpunkt berechnet (und auch keine Ressourcen reserviert), sondern lediglich Prioritäten bestimmt. Die Prioritäten der Aufträge bewirken vorrangige, d.h. schnellere Zuweisung und damit schnellere Abarbeitung durch die Sortierung der Aufträge nach Prioritätenfolge in den zentralen Warteschlangen. Dadurch erlangen die höher-priorisierten Aufträge in der Regel auch die leistungsfähigeren Knoten. Wann die Aufträge wohin zugewiesen oder an Nachbar-Cluster verschoben werden, wird weiterhin durch den Entscheidungsalgorithmus bestimmt, der aktuelle Laufzeitinformationen berücksichtigt. Weiterhin ist zu beachten, daß die Berücksichtigung von Auftragsprioritäten der Durchsatzsteigerung, nicht der Minimierung einzelner Auftrags-Antwortzeiten dient. Dies wird im *HiCon*-Modell automatisch berücksichtigt, da eine Einsortierung der Aufträge natürlich nur in Situationen hoher Last, wenn die zentrale Auftragswarteschlange nicht leer ist, erfolgt. In Hochlastsituationen optimiert die *HiCon*-Balancierung ohnehin verstärkt auf Systemdurchsatz.

Die Priorität von Aufträgen wird in Instruktionen gemessen und stellt jeweils die Größe des Auftrags samt den Größen seiner Folgeaufträge dar. Prioritäten können dadurch auch zwischen Aufträgen unterschiedlicher konkurrierender bzw. früher oder später eingeplanter Anwendungen miteinander verglichen werden, weil das Maß unabhängig von Anwendung oder Knotenleistung ist. Das ermöglicht die dynamische Berücksichtigung von Reihenfolgebeziehungen im heterogenen Mehrbenutzerbetrieb.

Bei der Einsortierung eines Auftrags in die zentrale Warteschlange werden nicht allein die Prioritäten zum Einplanungszeitpunkt betrachtet, sondern auch die bisherige War-

tezeit der Aufträge in der zentralen Warteschlange (Abschnitt 3.7.2): Die Priorität der auf Verschiebung oder Zuweisung wartenden Aufträge wird um den Betrag (an Instruktionen) erhöht, den sie seit ihrer Entstehung auf einem durchschnittlichen Rechenknoten an Bearbeitung erhalten hätten. Dadurch wird verhindert, daß Aufträge mit geringer Priorität beliebig lange in der Warteschlange bleiben, solange höher-priorisierte Aufträge verfügbar sind. Eine Leistungsbewertung der Ansätze findet sich in Abschnitt 5.6.1.

3.7.5 Dynamische Adaption

Das *HiCon*-Konzept enthält Konzepte zur dynamischen Anpassung der Balancierung gemäß Abschnitt 2.5.6. Die Lastbalancierung lernt also selbständig aus früheren Entscheidungen; sie paßt ihre Vorgehensweise in den ungenaueren Teilen ihres Modells dem real gemessenen Verhalten an und optimiert durch Selbstbeobachtung ihr Kosten-Nutzen-Verhältnis. Während die erste der folgenden Techniken die dezentrale Balancierung betrifft, sind die übrigen für zentrale Balancierung entwickelt. Eine Leistungsbewertung der Ansätze findet sich in Abschnitt 5.6.2.

1. Adaptive Regelung des Schwellwerts zur Verschiebung von Anwendungen zwischen Clustern

Da Auftragsverschiebung zwischen Clustern teuer ist und vor allem eine entsprechende Menge an Datenkommunikationskosten nach sich zieht, versucht die *HiCon*-Lastbalancierung, die Auftragslast möglichst innerhalb des Clusters zu verteilen. Wie in Abschnitt 3.7.3 beschrieben, werden dazu fixe Schwellwerte eingesetzt. Es wird aber auch ein Kriterium verwendet, das dynamisch der Netzwerkkapazität, den Datenkommunikationskosten und dem Volumen aktiver gemeinsamer Daten der Anwendungen angepaßt wird: Die Komponente zur Anwendungsverschiebung betrachtet bei der Suche nach der zum Verschieben bestgeeigneten Anwendung nur solche, die die Bedingung

$$w > 1 \wedge remainingProcessingTime_A > migDataCost_A$$

erfüllen. Dabei wird die Eignung w der Anwendung zur Verschiebung durch

$$w = \frac{t_{localProcessing_A}(A)}{migDataCost_A}$$

mit Hilfe der Abschätzung

$$migDataCost_A = \sum_d dataCommCostRemote_d \cdot activeGlobalData_{Ad}$$

bestimmt. Dies sind nur grobe Grenzen, die völlig aussichtslose Verschiebungen verhindern sollen. Die Menge der von der Anwendung benutzten Daten, d.h. die Anzahl an Datensätzen je Datentyp, kann von der Datenverwaltungskomponente des

Laufzeitsystems recht einfach grob abgeschätzt werden. Die Abschätzung *remainingProcessingTime_A* kann aus den kritischen Pfadangaben der Auftragsvorabschätzungen gewonnen werden.

2. Automatische Vermeidung von Überlastungen der Balancierungskomponente

Dieser Ansatz optimiert das Kosten-Nutzen-Verhältnis der zentralen Lastbalancierung innerhalb eines Clusters. Zentrale Lastbalancierung im *HiCon*-Modell bewirkt in der Regel eine deutliche Durchsatzsteigerung für einen breiten Anwendungsbereich. Obwohl man also annehmen kann, daß sich der Balancierungsaufwand lohnt, kann die Balancierung in großen Systemen und in Hochlastsituationen aufgrund der Informationssammlung und -Verarbeitung und der Entscheidungsfindung überlastet werden. Permanente Überlastungen sollten durch Verfeinerung der Cluster-Struktur behoben werden, aber kurzfristige Überlasterscheinungen müssen durch die Lastbalancierung selbst erkannt und vermieden werden, indem der Aufwand zur Informationsverwertung und Entscheidung reduziert wird.

Im *HiCon*-Ansatz reduziert die Adaption sowohl die Komplexität als auch die Häufigkeit der Entscheidungsberechnungen, solange der die Balancierungskomponente beherbergende Prozessor stark belastet ist oder die Entscheidungen nicht mehr zügig genug getroffen werden können. Die Anzahl der zur Bearbeitung anstehenden Ereignisse stellte sich als bestes Maß für die Belastung der Balancierungskomponente heraus. Dazu verwaltet die Balancierung sowohl den aktuellen Wert als auch einen exponentiell geglätteten. Um unmittelbar auf Überlastsituationen reagieren zu können, schaltet die Balancierung von der komplexen Strategie auf eine einfachere um, sobald die momentane Ereigniswarteschlangenlänge einen Schwellwert *S* übersteigt: Die vereinfachte Strategie bewertet Aufträge nur einmal bei ihrer Ankunft, und bestimmt dort den bestgeeigneten Server, anstatt bei jeder Aktivierung des Entscheidungsalgorithmus erneut aufgrund aktuellerer Daten zu bewerten. Erst wenn die geglättete Ereignisschlangenlänge unter *S* zurückfällt, wird wieder auf die komplexe Strategie umgeschaltet, um häufiges Umschalten zu vermeiden.

Weiterhin wird der Zeitraum *t_{advance}*, für den die Lastbalancierung im voraus Aufträge an Server zuweist, adaptiv durch

$$t_{advance} = t_{default} * (eventQueueSize + 1)$$

geregelt. *t_{default}* ist eine Konstante, die recht klein sein sollte, um späte Zuweisung aufgrund möglichst aktueller Informationen zu gewährleisten. Der empirische Wert von 1 sec hat sich hier als brauchbar erwiesen. *t_{advance}* wird also proportional zur Belastung der Balancierungskomponente erhöht, um mehr Aufträge aus der zentralen Warteschlange zu entfernen, die ja wiederholt bewertet werden müssen, und abzusichern, daß die Server trotz hoher Belastung der Balancierungskomponente durchgehend arbeiten können.

Die Erfahrungen ergeben, daß komplexe zentrale Balancierung effektiv ist, solange sie die Informationssammlung und Entscheidungsfindung noch rechtzeitig bewältigt, aber leichte Überlastungen zerstören schnell die Gewinne aufwendiger Strategien und müssen vermieden werden. Vereinfachte Strategien müssen sorgfältig gewählt werden, da die Balancierungsaufgabe nicht einfacher wird: Beispielsweise kann schon das Ignorieren von Datenaffinitäten die Ausführungszeiten mancher Anwendungen verzehnfachen. Der *HiCon*-Ansatz verringert den Entscheidungsaufwand, indem er auf weniger aktuellen Informationen basierende Entscheidungen in Kauf nimmt.

3. Adaptive Justierung von Auftragsgrößen-Vorabschätzungen

Der Entscheidungsalgorithmus verwendet Vorabschätzungen über die Auftragsgrößen, um die bestgeeigneten Server zu ermitteln. Diese von den Clients beim Aufruf mitgegebenen Informationen sind jedoch nur grobe Schätzungen. Sie treffen meist die relativen Größenverhältnisse zwischen Aufträgen eines Typs gut, nicht aber die von den Eingabedaten (Problemstellung) abhängigen Größenveränderungen. Die *HiCon*-Lastbalancierung vergleicht daher die tatsächlich beobachtete Ausführungszeit (deren Rechenanteil) $t_{computeReal}$ mit der aus den Vorabschätzungen des Clients abgeleiteten, vermuteten Rechenzeit $t_{compute}(s,a)$ und regelt einen Korrekturfaktor $computeTimeAdapt_{cs}$ pro Serverklasse und Auftragsstyp entsprechend mit exponentieller Glättung (Gewichtung α) nach:

$$computeTimeAdapt_{cs} = (1 - \alpha) \cdot computeTimeAdapt_{cs} + \alpha \cdot t_{compute}(s,a) / t_{computeReal} ,$$

mit dem weitere Auftragsgrößenabschätzungen der Clients korrigiert werden.

4. Adaptive Bestimmung von Datenkommunikationskosten

Die Kosten (Wartezeiten), um Daten zwischen Servern auszutauschen oder zu kopieren, hängt von sich dynamisch ändernden Faktoren ab, die schwer zu ermitteln und geeignet zu berücksichtigen sind, wie z.B. der momentanen Netzwerkauslastung durch Mehrbenutzerbetrieb oder Anwendungs-interner Parallelität, momentanen Datengrößen und Sperrwartezeiten ab. Daher paßt die *HiCon*-Lastbalancierung die Datenzugriffskosten $dataCommCost_{cd}$ je Serverklasse c und Datentyp d durch exponentielle Glättung aufgrund der tatsächlich beobachteten Zeiten der letzten Zeit an und verwendet diese Werte, um die Datenwartezeiten für Aufträge bei Servern abzuschätzen, die einige der vermutlich benötigten Daten momentan nicht lokal verfügbar haben. Die Verwendung aktueller Wartezeiten ist aufgrund starker Schwankungen sinnlos, aber längerfristige Änderungen der Datenkommunikationskosten sollten unbedingt berücksichtigt werden.

Da der Kommunikationsaufwand zwischen Clustern gewöhnlich deutlich größer ist, werden separate Adaptionfaktoren $dataCommCostRemote_{cd}$ verwaltet.

Um die Verteilung von Kopien zu fördern, werden die Kostenabschätzungen für lesende Datenzugriffe um den Faktor $dataReadWrite_{cd}$ verringert. Der Faktor wird für jeden Datentyp adaptiv geregelt durch Beobachtung, wieviel lesende Zugriffe durchschnittlich zwischen zwei exklusiven Zugriffen erfolgen. Dieser Faktor ist notwendig, da sonst bei Betrachtung einzelner Aufträge nur der große Aufwand zum Anlegen der Kopie berücksichtigt wird, und nicht der Nutzen für nachfolgende lokale Lesezugriffe auf diese Kopie.

5. Adaptive Korrektur der Vorabschätzungen des Datenkommunikationsaufwands von Aufträgen

Bei der Auswahl des bestgeeigneten Servers für einen Auftrag schätzt der Entscheidungsalgorithmus auch jeweils die zu erwartenden Datenkommunikationskosten für den Auftrag ab. Clients können beim Aufruf Datenreferenzmuster angeben, die jedoch oft nur grob und unvollständig sind. Es ist sinnlos, die tatsächlich referenzierten Datensätze eines Auftrags mitzuverfolgen, um sie für weitere Aufträge als Vorabschätzung zu verwenden, weil diese mit hoher Wahrscheinlichkeit wechseln. Die Aufträge einer parallelisierten Schleife arbeiten beispielsweise meist auf der gleichen Menge von Daten, aber auf disjunkten Datensätzen. Jedoch kann der Anteil der fehlenden oder falsch vorgegebenen Datenreferenzen, und damit die relative Abweichung der gesamten realen Datenkommunikationskosten von den vorabgeschätzten pro Auftragsstyp beobachtet werden, um die Datenwartezeit für spätere Aufträge entsprechend korrigieren zu können. Dazu wird ein Faktor $dataTimeAdapt_{cs}$ mit exponentieller Glättung (Gewichtung α) jeweils nach Beendigung eines Auftrags a durch

$$dataTimeAdapt_{cs} = (1 - \alpha) \cdot dataTimeAdapt_{cs} + \alpha \cdot t_{dataComm}(s,a)/t_{dataReal}$$

geregelt. Generell ist exponentielle Glättung ein Verfahren, das dynamische Anpassung bei gewisser Robustheit bewirkt und nicht rechen- oder speicherintensiv ist.

6. Adaptive Einschätzung der CPU-Last durch Auftragsausführungen und Lastbalancierungsaktivität

Der Entscheidungsalgorithmus verwendet zur Vorabschätzung von Auftragslaufzeiten die Lastfaktoren der Prozessoren, die durch konkurrierend laufende Server oder durch die Lastbalancierung entstehen. Einfache Möglichkeiten bestehen darin, die Prozessorleistung durch die Anzahl der momentan aktiven Server oder aber durch die vom Betriebssystem ermittelte CPU *run queue length* zu dividieren. Ersteres birgt die Fehlerquelle, daß die Server nicht alle ständig voll die CPU beanspruchen, sondern auch Ein-/Ausgabe durchführen und kommunizieren, so daß die Prozessorbelastung überschätzt wird. Die Verwendung der CPU *run queue length*, die von den Betriebssystemen als ganzzahliger, über längere Zeit gemittelter Wert zur Verfügung gestellt wird (beispielsweise alle 5 Sekunden in UNIX-Systemen), ist sehr ungenau, enthält kurzfristige Lastspitzen und erlaubt keine Vorabschätzung der Belastung für

die nahe Zukunft. Im *HiCon*-Ansatz wird daher die CPU *run queue length* nur verwendet, um Abschätzungsfaktoren $cpuUtilAdapt_{cs}$ für die mittlere CPU-Nutzung von Auftragsstypen längerfristig zu regeln. Die Belastung des Knotens k zu einem Zeitpunkt t kann dadurch mittels

$$l = \left(\sum_{(i \in servers_k \wedge t_{remainingWork}(i) > t)} cpuUtilAdapt_{cs'} + 1 \right)$$

abgeschätzt werden, was auch für die nahe Zukunft akkurate Vorabschätzungen ermöglicht. Der real (vom Betriebssystem) gemessene mittlere CPU-Nutzungsanteil der Balancierungskomponente der letzten Sekunden wird zuaddiert, falls auf dem Knoten k wie die Balancierungskomponente läuft. Durch Vergleich mit der real vom Betriebssystem gemessenen mittleren CPU *run queue length* der letzten Sekunden können die Abschätzungsfaktoren mit exponentieller Glättung (Gewichtung α) mittels

$$cpuUtilAdapt_{cs} = (1 - \alpha) \times cpuUtilAdapt_{cs} + \alpha \times (CPUrunQueueLen \times numberOfProcessors_k - l)$$

nachgeregelt werden. Dabei werden durch l rekursiv die Abschätzungsfaktoren der sonstigen momentan auf dem Knoten laufenden Auftragsstypen mitverwendet.

4 Vergleich mit relevanten Forschungsarbeiten

Das *HiCon*-Modell soll mit den wichtigsten veröffentlichten Ansätzen verglichen werden, die vergleichbare Problemstellungen betrachten oder verwandte Ansätze entwickeln. Interessant sind zentrale Verfahren, Balancierungsverfahren im Bereich der Datenverwaltung, dezentrale Ansätze und spezielle Verfahren für Workstation-Netze.

4.1 Zentrale adaptive Transaktionsplatzierung im Datenbankbereich

Im Bereich der Datenbankapplikationen wird automatische Lastverteilung seit langem durch *Transaction Processing* Monitore unterstützt [Borr90]. In Bezug auf dynamische Lastbalancierung nehmen diese Systemkomponenten die Aufgaben der Pufferung von einlaufenden Transaktionen, der Zuweisung an geeignete Ausführungseinheiten (Server) sowie der Konfigurationsverwaltung wahr. Charakteristisch für die Systemlastprofile sind im Datenverwaltungsbereich einerseits die einzelnen, sequentiellen Transaktions-Aufträge, die in hoher Parallelität auftreten und aufgrund der ausgereiften, optimierenden Anfrageübersetzer automatisch mit Profilabschätzungen versehen werden können. Andererseits greifen die Transaktionsaufträge zu einem erheblichen Zeitanteil auf große Mengen globaler Daten zu. Dies ergibt für Lastbalancierung ähnliche Anforderungen und Randbedingungen, wie sie im *HiCon*-Modell zugrundegelegt werden. Umverteilung oder Replikation von Daten wurde allerdings bisher wenig betrachtet, was an den Grenzen der zugrundeliegenden Datenbanksystemen liegt.

Als wichtigster Ansatz sollen die Arbeiten von [Yu86], [Yu91] vorgestellt werden. Eine zentrale Lastbalancierungskomponente verteilt einlaufende Transaktions-Aufträge auf ein paralleles System, dessen Knoten jeweils Teile einer großen Datenbank verwalten. Zuweisungsentscheidungen werden dynamisch getroffen, eine spätere Migration laufender Transaktionen ist nicht möglich. Das Ablaufmodell eines Transaktions-Auftrags besteht aus lokalen Berechnungen auf dem zugewiesenen Knoten, die durch die eigentlichen Datenbankankfragen / -Operationen unterbrochen werden. Die Datenbankankfragen müssen jeweils zu dem Knoten geschickt werden, der den betroffenen Teil der Daten verwaltet, da keine Daten kopiert oder migriert werden können (shared disk Modell). Dies erzeugt Kommunikationsaufwand und Rechenaufwand bei den betroffenen Knoten. Das Lastbalancierungskonzept ermöglicht Zuweisungsentscheidungen, die aufgrund von Profilvorabschätzungen der Aufträge und Kenntnis über die Datenverteilung und aktuelle Systemauslastung die Antwortzeiten der Aufträge zu minimieren versuchen. Die Antwortzeit eines Auftrags wird dabei durch die Rechenkapazität des zugewiesenen Knotens für die lokalen Berechnungen, Kommunikationswartezeiten für nicht-lokale Anfragen und Rechenzeit für die Datenbankankfra-

gen auf den zuständigen Knoten aufsummiert. Da die Vorabschätzungen der Transaktionsprofile und der verschiedenen Komponenten des Bearbeitungsaufwands ungenau und fehlerhaft sind, werden durch periodische Regressionsanalysen Korrekturfaktoren für den Zusatzaufwand durch entfernte Datenbankabfragen sowie für die gesamte Antwortzeitschätzung aufgrund des real beobachteten Verhaltens nachgeregelt. Im Vergleich zum *HiCon*-Ansatz ist das Modell lediglich eine Simulationsstudie, die in ihrer Anwendbarkeit auf den Bereich der klassischen Datenbank-Applikationen beschränkt ist.

In [Rahm93] werden Ansätze zur dynamischen Zuweisung von Aufträgen mit transaktionsinterner Parallelität am Beispiel der parallelen Verbundberechnung im Ein- und Mehrbenutzerbetrieb entwickelt. Dabei wird die einzusetzende Parallelität und die Datenzuordnung aufgrund aktueller Größen wie Datenumfang, Rechenaufwand, Puffergrößen, Netzwerkleistung etc. dynamisch bestimmt.

Dynamische Balancierung durch Reorganisation der Datenverteilung in Datenbankverwaltungssystemen wird in [Jian89] entwickelt, um trotz durch häufige Einfüge- und Änderungsoperationen verstreuter Daten teure Retrieval-Operationen durch Clustering und Lokalität effektiv zu erhalten.

4.2 Lastbalancierung in Workstation-Netzen

In den letzten Jahren wurden verstärkt Ansätze zur automatischen Lastverteilung für Workstation-Netze entwickelt [Bern93]. Hier gelten besondere Randbedingungen: Workstations eines Clusters sind gewöhnlich durch ein vergleichsweise langsames gemeinsames Netz verbunden und die Dateien der Dienstprogramme sowie die Benutzerdaten werden zentral auf Fileservern verwaltet. Es gibt kein verteiltes Betriebssystem, die Workstations agieren autonom und sind meist bestimmten Benutzern / Besitzern zugeordnet. Da Workstations zugleich Benutzer-Endgeräte sind und aufwendige graphische Benutzeroberflächen anbieten, wird oft verlangt, daß dem interaktiven Benutzer einer Workstation mehr Rechenleistung zur Verfügung gestellt werden muß als anderen Anwendungen, die den Rechner lediglich als Knoten eines parallelen Systems nutzen. Die Lastcharakteristiken von Workstation-Clustern zeigen bei starker Überlastung einzelner Rechner zu Spitzenzeiten und großen Leerlaufzeitspannen auf den meisten Knoten eine mittlere Gesamtsystemauslastung von unter 10% [Mutk92]. Während Workstations bisher hauptsächlich als Entwicklungs-, Verwaltungs- und Bürokommunikationsmaschinen eingesetzt wurden, werden sie zunehmend für große parallele Anwendungen aus dem Produktions- und Datenverwaltungsbereich genutzt. Die wichtigsten Projekte zur automatischen Lastverteilung in Workstation-Clustern sollen kurz charakterisiert werden.

Das dezentrale Verfahren *NEST* [Ezza86] zur dynamischen Auftragsplazierung von UNIX-Prozessen wurde für Workstations realisiert. Migration laufender Prozesse ist nicht möglich. Als Lastinformationen über Knoten werden die *CPU run queue length* und der Prozessornutzungs-Zeitanteil ausgetauscht. Über die Prozesse werden keine Vorabinformationen verwendet.

Die Umgebung *Utopia* [Zhou92] weist UNIX-Prozesse bei ihrer Entstehung wenn möglich Workstations zu, die momentan frei sind, anstatt sie lokal zu starten. Migration laufender Prozesse ist nicht möglich, da heterogene Architekturen unterstützt werden. Informationsverwaltung und Entscheidung ist innerhalb von Clustern dezentral organisiert, während die Kooperation zwischen Clustern durch jeweils einen Knoten gebündelt abläuft. Zu Prozessen kann als Vorabschätzung mitgeteilt werden, ob sie vorwiegend Rechenzeit-, Hauptspeicher- oder Ein-/Ausgabe gebunden sind. Als Lastgrößen für Zuweisungsentscheidungen können die *CPU run queue length* und CPU-Nutzung, der freie Hauptspeicher, die Disk-Zugriffsrate, sowie die Anzahl der Benutzer betrachtet werden. Der verwandte *Task Broker* Ansatz [Graf93] kann weiterhin geeignete Server für Aufträge mit speziellen Anforderungen wie Fließkomma- oder Grafik-Beschleunigern berücksichtigen. In diesem Ansatz sendet eine zentrale Balancierung pro Auftrag Anfragen an Knoten, die mit Angeboten antworten. Aus diesen wählt die Balancierung den bestgeeigneten Server. Aufträge werden in Warteschlangen aufbewahrt, bis ein geeigneter Server verfügbar ist. Die Umgebung *Load Leveler* [IBM95] bietet verschiedene Warteschlangen für Aufträge mit verschiedenen Charakteristiken (Klassen). Bei Definition eines Auftrags können Ressourcenbedürfnisse wie Anzahl paralleler Prozessoren oder benötigte Dateien spezifiziert werden. Die Lastbalancierung führt Staisitken über den tatsächlichen Ressourcenverbrauch der Klassen und weist Aufträge so zu, daß die Knotenauslastung nicht zu hoch ist und die Ressourcenbedürfnisse erfüllt werden können.

Die Umgebung *Condor* [Litz88] und die verwandte Umgebung *Sprite* [Doug91] bieten dynamische Lastbalancierung von UNIX-Prozessen in homogenen Systemen, indem sie große Prozesse in Knoten-Warteschlangen verwaltet und auf Workstations starten, an der seit einer gewissen Zeit kein interaktiver Benutzer mehr tätig war. Ein zentraler Koordinator überprüft periodisch, welche Knoten verfügbar sind, und ebenso, ob Benutzer ihre Workstations wieder interaktiv nutzen und läßt dort die Prozesse durch einen Checkpoint-Restart Mechanismus (*Condor*) bzw. durch Migration des Prozeßkontrollblocks und Umverlagerung der Speicherseiten bei Bedarf (*Sprite*) wieder auf dem Ursprungsknoten zurückverlegen. Die Umgebung PVM [PVM93] zur einfachen, Plattform-unabhängigen Realisierung und Verwaltung paralleler Anwendungen, die über Nachrichten kommunizieren, verfügt ebenfalls über einen Prozeßmigrationsmechanismus, der Prozesse von überlasteten Workstations oder solchen, die wieder interaktiv genutzt werden, auf freie Workstations migriert [Casa94].

Insgesamt stellen diese Ansätze simple, pragmatische Verbesserungen der Lastverteilung in Workstation-Netzen dar. Im Vergleich zum *HiCon*-Ansatz sind sie für existierende Anwendungen direkt einsetzbar, haben jedoch ein deutlich geringeres Optimierungspotential, sind nicht für große, komplexe und parallele Anwendungen ausgelegt und verfügen nicht über geeignete Modelle zur Berücksichtigung der Datenkommunikation.

4.3 Dezentrale Lastbalancierung für Parallelrechner

Für Parallelrechner mit großer Anzahl von Knoten bzw. für große parallele und verteilte Systeme werden dezentrale Lastausgleichsverfahren entwickelt, die entweder die Aufträge einer massiv parallelen Anwendung oder große Zahlen unabhängiger Einzelaufträge im System balancieren sollen. Wegen der hohen Frequenz von Auftragseingängen und Auftragsbeendungen und der großen Knotenzahl soll Skalierbarkeit durch völlig dezentrale Ansätze garantiert werden, bei denen auf jedem Knoten Balancierungskomponenten laufen, die mit Nachbarn Lastinformationen und Aufträge austauschen.

In [Lüli91] werden die wichtigsten dezentralen Ansätze vergleichend auf einem großen Transputersystem für eine baumstrukturierte parallele Anwendung simuliert. Die *Gradientemethode* [Lin87] verwendet drei Lastzustände pro Knoten, basierend auf der CPU *run queue length*. Knoten informieren bei Lastzustandswechsel ihre Nachbarn, wobei jeder Knoten seine kürzeste Entfernung zu einem unterbelasteten Knoten angibt. So können überlastete Knoten Aufträge entlang des steilsten Gradienten in Richtung des nächsten unterbelasteten Knotens absenden. Im Verfahren *Contracting with Neighborhood* [Sale90] wandert jeder neue Auftrag eine gewisse Mindest- und Höchstzahl an Knoten von seinem Ursprungsknoten weg und bleibt auf dem Knoten mit minimaler Last. Der Suchradius kann adaptiv aufgrund der Gesamtsystemlast geregelt werden. [Kale88] vergleicht obige Verfahren.

Insgesamt sind die dezentralen Verfahren uneingeschränkt skalierbar, da der Balancierungsaufwand proportional zur festen Anzahl von Nachbarn je Knoten ist. Sie verwenden jedoch nur simple Lastmodelle, erreichen auch nur sehr langsam Lastausgleich zwischen entfernten Teilen des Systems und neigen mangels zentraler Koordination zu kontraproduktiven Entscheidungen (Abschnitt 2.5.3.2).

4.4 Dynamische Lastbalancierung unter Verwendung von Vorabinformationen

Dynamische Lastbalancierungsansätze, die Vorabschätzungen nicht für Client-Server Ablaufmodelle (Abschnitt 4.1), sondern für das Modell unabhängiger einzelner Prozesse einsetzen, verwenden entweder beim interaktiven Programmstart vom Benutzer mitgegebene Hinweise wie Laufzeit, Eignung zur Migration oder CPU-, Hauptspeicher-, Ein/Ausgabebedarf [Graf93], oder führen eine Liste der häufiger laufenden Programmdateinamen, deren Ablaufprofile sie beobachten und statistisch registrieren [Gosw93]. Diese Statistiken können später für Vorabschätzungen wiederverwendet werden.

Der *HiCon*-Ansatz kombiniert beide Möglichkeiten und kann durch Verwendung des Client-Server Ablaufmodells akkuratere Auftragsprofilvorabschätzungen speziell für Serverklassen und deren Auftragsarten verwalten. Tatsächlich beobachtete Ausführungsprofile werden zur adaptiven Korrektur weiterer Vorabschätzungen verwendet.

4.5 Komplexe zentrale dynamische Lastbalancierung

Es gibt bisher nur wenige dynamische Lastbalancierungsverfahren, die detaillierte Ablaufmodelle verwenden und komplexe Zuweisungsalgorithmen darauf anwenden. Als wichtigster Ansatz soll das durch Simulation evaluierte Balancierungskonzept von [Chow79] vorgestellt werden. Eine zentrale Lastverteilungskomponente übernimmt die Zuweisung neuer Aufträge. Es werden drei Strategien vorgeschlagen und verglichen, die in etwa den zweiten bis vierten Optimierungskriterien aus Abschnitt 2.5.7 gerecht werden. Die erste Strategie minimiert die Antwortzeit der einzelnen Aufträge, wobei gleiche Auftragsgrößen angenommen werden: Jeder Auftrag wird dem Prozessor zugewiesen, dessen Quotient *Anzahl laufender Prozesse / Prozessorleistung* am kleinsten ist. Die zweite Strategie minimiert die Zeit, bis alle zur Zeit im System befindlichen Aufträge einschließlich des neuen Auftrags beendet sind. Die Auftragsgrößen sind dabei vorab bekannt. Jeder Auftrag wird so zugewiesen, daß die Restlaufzeit $\sum \text{Auftragslängen seiner Aufträge} / \text{Prozessorleistung}$ des Prozessors mit der längsten Restlaufzeit minimal ist. Die dritte Strategie benötigt zusätzlich Wissen über die Ankunftszeit des nächsten Auftrags, und weist Aufträge so zu, daß der Systemdurchsatz bis zur nächsten Auftragsankunft maximal wird: Jeder Auftrag wird so zugewiesen, daß die Summe der Arbeit der Prozessoren bis zur nächsten Auftragsankunft maximal wird, wobei der Arbeitsumfang eines Prozessors das Produkt aus Prozessorleistung und der Intervalllänge ist, in der er Aufträge zur Bearbeitung hat.

Ansätze dieser Kategorie kommen dem Ablaufmodell und Balancierungskonzept des *HiCon*-Modells sehr nahe, allerdings wurden solche Ansätze bisher nicht realisiert und es fehlen geeignete Modellierungskonstrukte für Datenkommunikation.

5 Leistungsbewertung des Ansatzes

Das dem Lastbalancierungsansatz zugrundeliegende Ablaufmodell ist zu komplex, als daß es durch geschlossene stochastische Modelle oder durch Simulationen bewertet werden könnte. Zahlreiche Effekte, die signifikanten Einfluß auf den Bearbeitungsverlauf im System haben, würden außer acht gelassen. Vereinfachte synthetische Lastprofile treffen bei weitem nicht das Verhalten komplexer realer Anwendungen. Komplexe Anwendungen mit nichttrivialen Abhängigkeiten zwischen Aufträgen bilden jedoch den Anwendungsbereich des Lastbalancierungsansatzes. Die Leistungsbewertung soll daher anhand einer prototypischen Laufzeitumgebung unter Belastung durch diverse reale Anwendungen erfolgen. In diesem Kapitel werden zuerst die prototypische Implementierung der Umgebung, danach realisierte Applikationen und schließlich einige Meßergebnisse vorgestellt.

5.1 Die prototypische Lastbalancierungsumgebung

Die prototypische Lastbalancierungsumgebung soll hauptsächlich den Anforderungen der Portabilität, eines möglichst simplen Laufzeitsystems, das die geforderte Funktionalität des Ablaufmodells realisiert, eines geringen Zusatzaufwandes gegenüber unbalancierten Abläufen und der Flexibilität gegenüber verschiedenen zu integrierenden Lastbalancierungsverfahren genügen.

Aus Gründen der Portabilität wurde das UNIX-Betriebssystem als Plattform gewählt. Wegen teilweise erheblicher Unterschiede in den Realisierungen des Betriebssystemkerns verschiedener UNIX-Derivate basiert die Lastbalancierungsumgebung ausschließlich auf standardisierten Systemaufrufen und verlangt weder besondere Benutzerprivilegien, noch Modifikationen des Betriebssystems noch das Aufsetzen spezieller Dienstprozesse. Die Implementierung wurde in der plattformunabhängigen Programmiersprache C durchgeführt.

Da die Thread-Konzepte noch nicht durchgehend standardisiert bzw. noch nicht breit verfügbar sind, wurde nur das Prozeßkonzept verwendet; In Abschnitt 6.2 werden laufende Entwicklungsarbeiten an einem Thread-basierten Laufzeitsystem angesprochen. Aufgrund der hohen Kosten für Prozeßwechsel und Interprozeßkommunikation in UNIX-Betriebssystemen wurden möglichst wenige Prozesse eingesetzt. So sind Teile des Laufzeitsystems mit der Lastbalancierungskomponente zu einem zentralen Prozeß zusammengebunden und andere Teile des Laufzeitsystems als Bibliotheken an die Clients und Server gebunden. Jeder Client und jeder Server ist als Prozeß realisiert, und je Rechenknoten wird die Lastmessung durch einen weiteren Prozeß übernommen. In dezentralen Konfigurationen existiert für jedes Cluster ein Lastbalancierungsprozeß.

Die Interprozeßkommunikation basiert auf der TCP/IP-Protokollfamilie, die derzeit als einzige breit verfügbare Schnittstelle in heterogenen Systemen Kommunikation zwischen Prozeßpaaren ermöglicht. Da die Anzahl der möglichen Verbindungen je Prozeß und je Rechenknoten beschränkt ist, wurde das verbindungslose UDP-Protokoll verwendet, das schnelle, aber keine sichere, duplikatfreie oder Reihenfolge-erhaltende Paketübermittlung garantiert. Die Beschränkung auf eine einfache Kommunikations- und Prozeßstruktur erlaubt es, die Lastbalancierungsumgebung beliebig auf heterogenen Workstation-Netzen unterschiedlicher Hersteller ablaufen zu lassen. Dies ist vor allem wichtig, um Meßreihen auf großen Systemen und zwischen unterschiedlichen Rechenzentren zu ermöglichen.

Die Laufzeitumgebung stellt den Anwendungen Serverklassen-Aufrufe und Resultatrückgaben sowie Sperranforderungen zum Zugriff auf globale Datensätze zur Verfügung. Die Konvertierung globaler Datensätze der Anwendungen zum Versenden geschieht durch anwendungsspezifische Rückrufprozeduren in den Servern, wodurch die Struktur der Hauptspeicher- oder Sekundärspeicherdaten völlig beliebig ist. Die Lastbalancierung wird durch Rückruffunktionen vom Laufzeitsystem über alle relevanten Ereignisse informiert; sie kann beliebige Informationen über den System- und Anwendungszustand verwalten und Aufträge, Prozessoren, Server sowie Nachbar-Cluster für Zuweisungs- oder Verschiebungsentscheidungen bewerten. Das Laufzeitsystem und das Konzept der zentralen sowie Server-lokalen Auftragswarteschlangen erlaubt es, der Lastbalancierung völlig frei zu stellen, wann und wie sie die Aufträge an Server zuweist oder an Nachbar-Cluster verschiebt. Zugewiesene Aufträge können nicht mehr aus der Server-Warteschlange zurückgenommen werden, sondern werden vom Server in *first-in-first-out* Reihenfolge bearbeitet. Laufende Auftragsausführungen können nicht suspendiert werden, sondern werden pro Knoten mit gleicher Prozeßpriorität abgewickelt.

Meßreihen werden durch ein Skript pro Cluster beschrieben und durch Start einer Lastbalancierungskomponente initiiert. Diese startet die Lastbalancierungskomponenten für die konfigurierten Nachbar-Cluster und die weiteren benötigten Prozesse. Die Clients müssen nicht alle zugleich starten, sondern können mit beliebigen Verzögerungen relativ zum Startzeitpunkt der Meßreihe ihre Anwendung beginnen. Dadurch sind beliebige Mehrbenutzer-Betriebsszenarien realisierbar. Nachdem alle Clients ihre Anwendungen fertig abgewickelt haben, terminiert der Meßlauf. Während eines Meßlaufs protokolliert jede beteiligte Lastbalancierungskomponente in komprimiertem Format die wichtigsten Ereignisse in Dateien. Ein speziell entwickeltes Präsentationswerkzeug ermöglicht nach Meßläufen die Auswertung dieser Protokoll-dateien, um das Verhalten der Anwendungen und die Wirkung der Lastbalancierung zu analysieren.

Mehrbenutzerbetrieb und heterogen gemischte Lastprofile können durch beliebige Mischung von Anwendungen untersucht werden. Alle Anwendungen können dabei jeweils mehrfach zugleich ablaufen, auf gemeinsamen globalen Daten und auf anwendungseigenen globalen Daten arbeiten. Die Server sind in der Lage, wechselnd Aufträge verschiedener Anwendungen (ihrer Klasse) zu bearbeiten und verwenden dabei automatisch die betreffenden globalen Datensätze.

Da die Lastbalancierung als Betriebssystemdienst betrachtet wird, sollte sie alle im System laufenden Aufträge kennen. In der prototypischen Umgebung können jedoch auch Prozesse im System laufen, die nicht über die Balancierungsumgebung gestartet wurden. Sie werden von der Lastbalancierung nur implizit als Störlast wahrgenommen.

Die zentral strukturierte Lastbalancierung bringt in der prototypischen Realisierung Zeit- und Kommunikationsaufwand mit sich, der bei geeigneter Einbettung in ein verteiltes Betriebssystem entfällt. Die Erweiterungen für die dezentrale Struktur der Lastbalancierung sind vom Zeit- und Kommunikationsaufwand her einer realistischen Implementierung ähnlich. Für die Abläufe in der zentralen Lastbalancierung sind daher folgende Punkte zu beachten:

1. Bei Einbettung in ein Betriebssystem entstehen keine nennenswerten Nachrichtenkosten durch die Kooperation zwischen Anwendungsprozessen und Laufzeitsystem oder zwischen Laufzeitsystem und Lastbalancierung innerhalb eines Rechenknotens. In der prototypischen Umgebung kommunizieren die Anwendungsprozesse hingegen durch Nachrichten untereinander und mit dem zentralen Prozeß, der Teile des Laufzeitsystems und der Lastbalancierung enthält. Dabei sind Nachrichten innerhalb eines Rechenknotens nur unwesentlich weniger aufwendig als Nachrichten zwischen verschiedenen Rechenknoten.
2. Bei geeigneter Einbettung in ein Betriebssystem können die Anwendungen innerhalb eines Rechenknotens Dateien und Speicherblöcke gemeinsam benutzen. Im verwendeten Prototyp wird die Datenverwaltung hingegen auf Server-Ebene durchgeführt, d.h. Server auf demselben Knoten tauschen Speicherbereiche und Dateien ebenso durch Nachrichten aus, wie es über Knotengrenzen hinweg notwendig ist.
3. Die Komponente des Laufzeitsystems, die den Zugriff auf virtuell gemeinsame Daten koordiniert, sollte als separater Dienst im Betriebssystem angesiedelt werden. Die prototypische Realisierung schleust hingegen alle Anforderungen an nicht-lokale Daten durch den zentralen Prozeß, der gleichzeitig Lastbalancierungsfunktionen übernimmt und daher durch häufige Datenbewegungen erheblich gestört werden kann. Er arbeitet Anforderungen an globale Datensätze, die er jeweils an den aktuellen Besitzer der Daten weiterleitet, nach dem *first-in-first-out* Prinzip mit gleicher Priorität ab wie neue Aufträge, Resultate oder Lastinformationen.

5.2 Untersuchte Anwendungstypen

Der Schwerpunkt der hier vorgestellten Untersuchungen liegt auf der Lastbalancierung für große, parallelisierte Anwendungen. Daher werden keine kleinen oder sequentiellen, unkorrelierten Aufträge bzw. Prozesse untersucht, sondern Anwendungen, deren Teilaufträge auf gemeinsamen Daten operieren, wobei die Datenkommunikation einen nicht zu vernachlässigenden Anteil an den Gesamtkosten der Bearbeitung einnimmt. Weiterhin können die Anwendungen konkurrierend auf globalen, persistenten Daten arbeiten. Die Zielumgebung ist weniger der in heutigen Workstation-Netzen und an Universitäten charakteristische Entwicklungs-, Übungs-, Test- und Verwaltungsbetrieb, sondern mehr der Produktionsbetrieb. Dabei soll das Rechnernetz aus Workstations und lose gekoppelten Parallelrechnern nicht für einzelne Anwendungsläufe reserviert sein, sondern durchaus für gemischten Betrieb konkurrierender großer Anwendungen zur Verfügung stehen. Das Konzept der Serverklassen wird im *HiCon*-Modell neben dem klassischen Einsatzbereich in Datenbankanwendungen auch für numerische Algorithmen eingesetzt. Die untersuchten Anwendungen wurden dahingehend ausgewählt, daß sie einen breiten Bereich an Lastprofilen abdecken und unterschiedliche Schwierigkeiten für automatische dynamische Lastbalancierung aufweisen.

5.2.1 Parallele Wegesuche in gerichteten Graphen

Diese Anwendung sucht in einem gerichteten Graphen, dessen Kanten unterschiedliche Längen besitzen, den kürzesten Weg zwischen zwei vorgegebenen Knoten. Zur Lösung wurde eine Breitensuche ausgewählt. Ein Client steuert die Suche vom Startpunkt zum Ziel folgendermaßen: Er ruft die Such-Serverklasse auf mit einer Liste von Knoten, die zuletzt erreicht wurden. Am Anfang enthält diese Liste nur den Startknoten. Die Such-Server finden die Knoten, die in einem Schritt direkt erreichbar sind, und geben diese Liste zurück. Der Client trägt die Ergebnisse in seine Tabelle der erreichten Knoten ein und startet mit den neu entdeckten oder günstiger erreichten Knoten neue Suchaufträge. Der Graph ist in Form mehrerer Dateien abgespeichert. Eine Datei enthält jeweils alle Kanten, die von einem bestimmten Knotenbereich ausgehen (Abbildung 19 veranschaulicht links die Aufteilung eines Graphen). Jede Datei bildet einen Datensatz bezüglich der Lastbalancierung, ist also das Granulat der Verteilung und Synchronisation. Da der Graph während der Suche statisch ist, lohnt es sich, zum parallelen Suchen Kopien an die Server zu verteilen. Der Client teilt neue Aufträge so ein, daß jeweils alle Startknoten eines Partitionsbereichs zu einem Aufruf gebündelt werden. Abbildung 19 skizziert in der Mitte und rechts den Ablauf zweier Suchschritte.

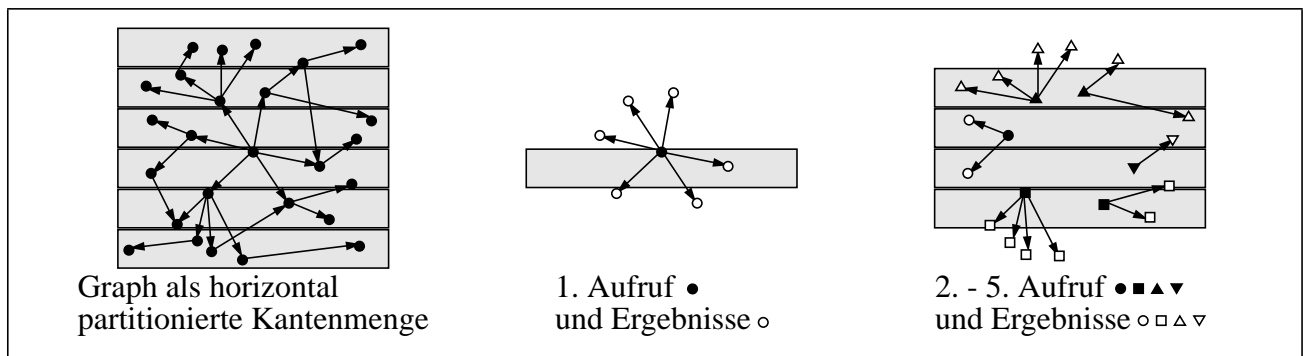


Abbildung 19: Aufteilung der Graphdaten und der Aufrufe in der Wegesuche.

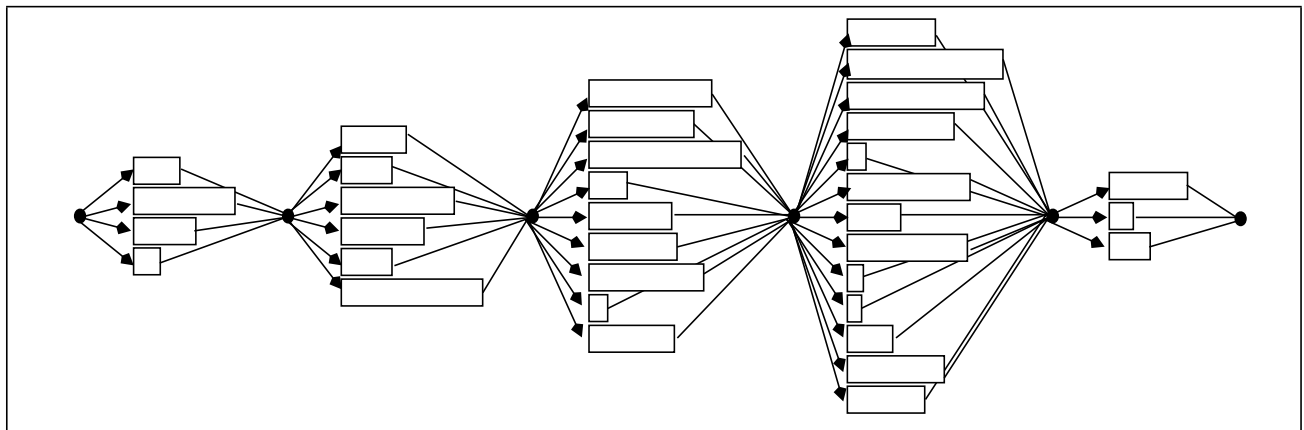


Abbildung 20: Typisches Ablaufschema der Aufträge einer Wegesuche.

Charakteristisch für diese Anwendung sind die wellenweise erscheinenden Aufträge (Abbildung 20) mit wechselnder, von der Struktur und Lokalität des Graphen abhängiger Parallelität. Es werden nur lesende Zugriffe auf gemeinsame Daten durchgeführt, da der Suchzustand jeweils vom Client verwaltet wird; ansonsten tritt keine Datenkommunikation auf. Die Anwendung ist sehr rechenintensiv, das mittlere Auftragsgranulat ist einstellbar; es schwankt stark, ist aber relativ präzise vorhersehbar. Datenreferenzen sind exakt vorhersehbar. Die Anwendung ist gut parallelisierbar, da sie, abgesehen von der Synchronisation durch den Client, kaum Netzlast und Wartezeiten durch Änderungsoperationen auf gemeinsamen Daten enthält. Lediglich das Auftragsgranulat und die Synchronisation nach jedem Suchschritt begrenzen die sinnvoll nutzbare Parallelität. Das Hauptproblem für die Lastbalancierung besteht darin, innerhalb einer Iteration die Aufträge so auf die Prozessoren zu verteilen, daß alle möglichst gleichzeitig fertig werden, um die potentielle Parallelität maximal zu nutzen.

In Messungen werden Auftragslaufzeiten im Bereich 1...20 Sekunden beobachtet. Die 20 Dateien des Graphen haben eine mittlere Größe von 15 KBytes. Gesucht wird der kürzeste Weg zwischen zwei Graphknoten auf einem Graphen mit 1000 Knoten und 200000 Kanten, der so generiert wurde, daß im Mittel 90% der Kanten zwischen Kno-

ten derselben Graph-Partition verlaufen. Dieses Maß an Lokalität im Graphen hat starken Einfluß auf das Parallelisierungs- und auf das Lastbalancierungspotential: Wenn sich die Suche innerhalb eines Schrittes sehr zerstreut, d.h. die Liste der erreichten Knoten viele Knoten aus verschiedenen Partitionen beinhaltet, entsteht eine Vielzahl sehr kleiner Folgeaufträge.

5.2.2 Parallele Flächenerkennung in Punktrasterbildern

Aufgabe der Flächensegmentierung ist es, ein gegebenes Punktrasterbild in eine Menge homogener Flächen (Polygone) zu konvertieren. Dieser Vorgang ist gewöhnlich die erste Phase einer Bilderkennung. Eine Fläche soll farblich beieinander liegende Punkte mit einem kleinen Anteil von Ausnahmen enthalten. Der verwendete Algorithmus besteht aus vier Schritten (Abbildung 22): Ausgehend von einer initialen Rasterung wird versucht, benachbarte Quadrate zusammenzufassen (Square Merge), sofern das neue Quadrat eine homogene Fläche ergibt. Parallel dazu werden die Quadrate solange verfeinert (Square Split), bis jedes Quadrat eine homogene Fläche enthält. Danach werden soviel als möglich benachbarte Quadrate zu beliebigen Polygonen zusammengefaßt (Polygon Merge). In der letzten Phase werden die Kantenzüge berechnet, welche die Polygone umgeben (Boundary Search). Abbildung 21 veranschaulicht das parallele Ablaufprinzip einer Flächensegmentierung.

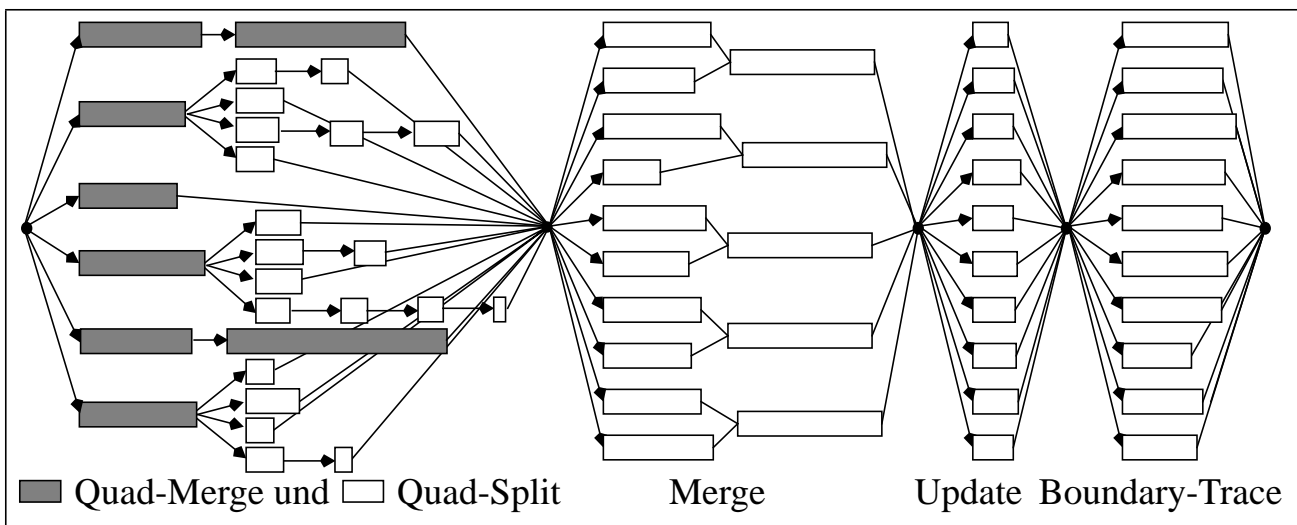


Abbildung 21: Paralleler Ablauf einer Flächenerkennung.

Im wesentlichen werden zwei globale Datenstrukturen verwendet: Ein zweidimensionales Datenfeld enthält für jeden Bildpunkt die Farbe und die derzeitige Zuordnung zu einer Fläche; eine Liste enthält für jede Fläche statistische Informationen (Farbmittelwert und Anteil der einzelnen Farben) bzw. eine Zugehörigkeit zu einer anderen Fläche.

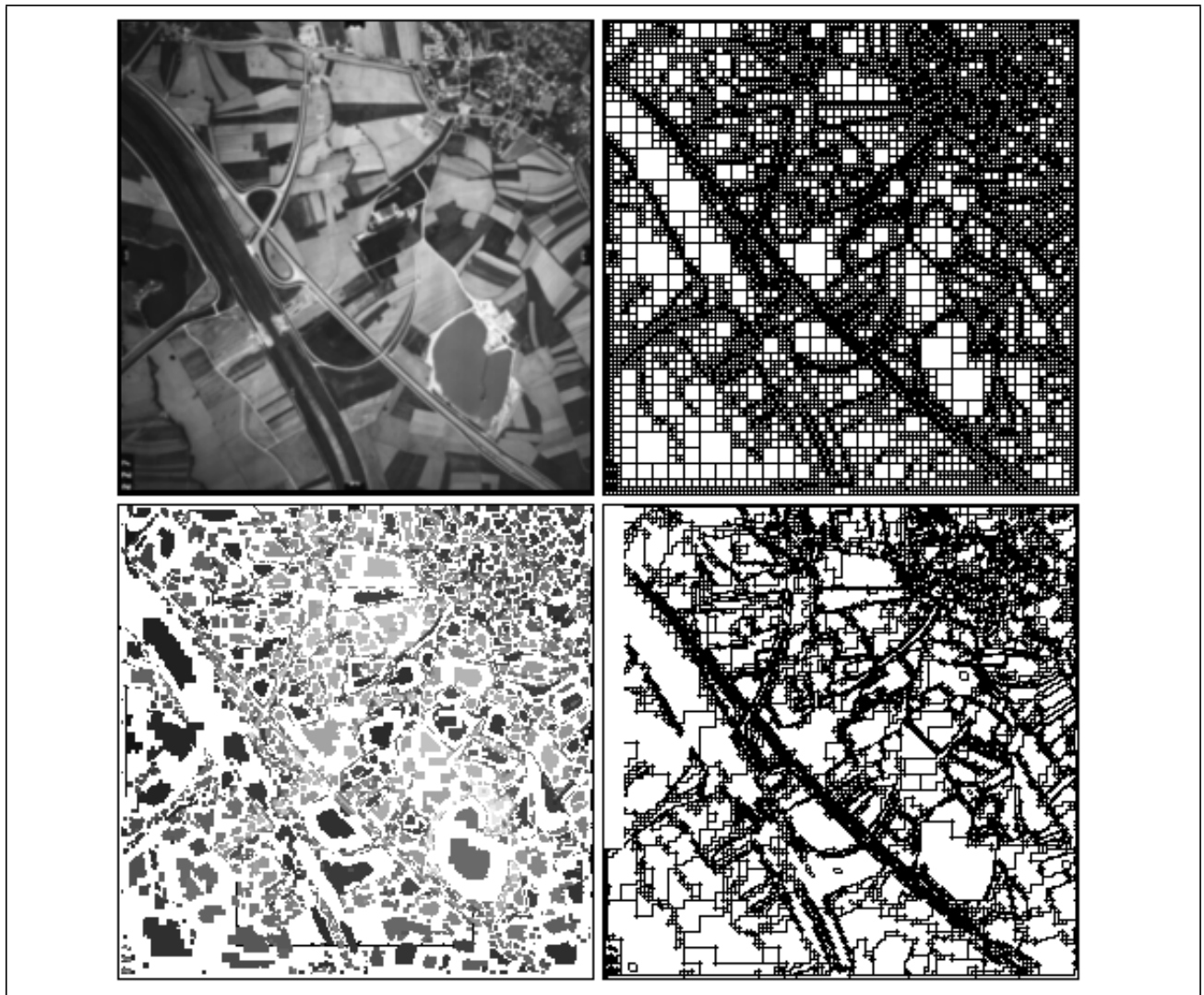


Abbildung 22: Schritte des Bildsegmentierungs-Algorithmus.

Split- und Merge-Operationen sind meist sehr feingranular (wenige Millisekunden). Daher werden sie soweit möglich nach Bildsegmenten zusammengefaßt und jeweils in einem Aufruf bearbeitet. Dennoch ist die Anzahl und das Granulat der Aufträge stark von der jeweiligen Bildstruktur abhängig und beeinflußt die sinnvoll nutzbare Parallelität. In der Polygon-Merge-Phase ist es nicht mehr so gut möglich, die Operationen auf disjunkte Bildteile anzusetzen, da die Polygonformen beliebig sind. Das verursacht je nach Bildstruktur starke Datenkommunikation und schränkt die Parallelität ein. Dieselbe Beobachtung gilt auch für die anschließende Berechnung der Polygonränder.

Die Anwendung besteht insgesamt aus mehreren aufwendigen Phasen mit sehr unterschiedlichen Profilen. Insgesamt und innerhalb mancher Phasen treten sehr unterschiedliche Auftragsgrößen auf. Die Aufträge verursachen sehr viele Datenzugriffe auf

große Mengen von Daten, viele davon schreibend. Die Datenzugriffe sind in manchen Phasen schwer vorhersehbar. Die Flächenerkennung ist nicht beliebig hoch parallelisierbar, da kurzlaufende Aufträge oft schreibend auf Daten zugreifen, die nicht disjunkt partitioniert werden können. Die Synchronisationspunkte zwischen den Phasen schränken hier die Parallelisierung nicht wesentlich ein. Die Anwendung ist schwer automatisch auszubalancieren, weil die Auftragsgrößen und Datenreferenzen von der Bildstruktur abhängen und nicht präzise vorabsehbar sind, andererseits aber die Lokalität der Datenzugriffe großen Einfluß auf die Laufzeit hat, und weil die Parallelität in der ersten Phase unstrukturiert und unvorhersehbar groß ist.

Für die Messungen gelten folgende Größenordnungen: Das Datenfeld der Bildpunkte besteht aus 900 Partitionen zu je 256 Bytes, die Liste der ca. 3300 entstandenen Flächen ist in 410 Blöcke der Größe 1400 Bytes partitioniert. Die Bearbeitungszeit pro Auftrag liegt zwischen 0.1 und 20 Sekunden. Von der Gesamtrechnenzeit entfallen etwa 15% auf die Phase Square-Merge und -Split, 75% auf Polygon-Merge und 10% auf die Phase Boundary-Search.

5.2.3 Parallele Verarbeitung komplexer relationaler Anfragen

Der relevante Anteil an Rechenaufwand und Ein-/Ausgabe-Aufwand in der kommerziellen Datenverarbeitung besteht im Auffinden, in der Verknüpfung und in der Filterung sowie Modifikation von Daten in einer Menge großer, inhaltlich korrelierter Datenbestände. In relationalen Datenbankverwaltungssystemen werden aufwendige Operationen auf großen Datenmengen in einer deskriptiven, mengenorientierten Sprache interaktiv oder in Anwendungsprogrammen formuliert. Diese Operationen werden automatisch in einen Satz sehr einfacher Grundoperationen konvertiert, die vom Datenbanksystem zur Laufzeit ausgeführt werden können.

Im Rahmen eines Anwendungsszenarios werden derartige Operationen ausgeführt. Die Grundoperationen bestehen hier im wesentlichen aus folgenden vier Typen: die Scan-Operation durchsucht große Datenmengen (üblicherweise Dateien oder Relationen) nach Datensätzen, die bestimmte Bedingungen erfüllen. Die Projektion filtert aus großen Datenmengen die relevanten Eigenschaften (Attribute) der Datensätze heraus. Die Verbundoperation verknüpft verschiedene Datenmengen aufgrund eines bestimmten Kriteriums, d.h. bildet inhaltlich zusammengehörige Paare von Datensätzen. Die Ladeoperation fügt große Mengen neuer Datensätze in die Datenbank ein.

Die Grundoperationen hängen teilweise voneinander ab und tauschen große Mengen von Daten (Zwischenresultate) aus. Anfrageübersetzer und -optimierer zerlegen deskriptiv vorgegebene, komplexe Operationen funktional und erzeugen eine Struktur von Aufträgen, die funktionale Parallelität und auch Datenparallelität innerhalb der

Grundoperationen enthalten können. Zur Ausnutzung der Datenparallelität werden die Basisrelationen sowie die Zwischenergebnisse nach Attributwertebereichen auf verschiedene Dateien partitioniert. Reihenfolge-Abhängigkeitsgraphen sind in diesem Bereich meist baumstrukturiert.

Die parallele Ausführung solcher Operationen ergibt charakteristische Lastprofile für kommerzielle Datenverarbeitung, die als Basisdienste für verschiedenartige Anwendungen verwendet werden. Für die Lastbalancierung solcher Profile können die Auftragsgraphen, meist mit Abschätzungen über die Größen und die Datenreferenzmuster der einzelnen Aufträge bereits zu Beginn der Bearbeitung der komplexen Operation verwendet werden, da sie von Anfrageübersetzern und Anfrageoptimierern generiert und abgeschätzt wurden. Statische Lastbalancierung zur Übersetzungszeit ist jedoch unzureichend, wenn sich die Größen der Basisrelationen und in Folge der Rechenaufwand der Aufträge im laufenden Betrieb ändern, wenn die Systemauslastung schwankt und wenn sich die Lageorte der Daten im System dynamisch ändern. Dynamische Anfrageoptimierung ist notwendig, stellt aber derzeit noch ein aktuelles Forschungsthema dar [Grae93], [Cole94].

Die realisierte Anwendung ermöglicht die Ausführung beliebiger als Auftragsgraph strukturierter, komplexer relationaler Anfragen und Ladeoperationen. Als Grundoperatoren sind die oben erklärten Scan-, Projektion-, Verbund- und Ladeoperationen auf Relationen verfügbar. Die unten vorgestellten Messungen basieren auf der Ausführung der folgenden Anfrage (in der Notation der relationalen Algebra): $R_7 = ((\sigma_{a1 > 15000} R_0) \times_{a0=a0} R_1) \times_{a0=a0} (\Pi_{a0} (\sigma_{a1 > 100} R_2))$. Abbildung 23 zeigt links den Operatorbaum einer Beispielanfrage und rechts den Abhängigkeitsgraphen einer möglichen Daten-parallelisierten Ausführung des Auftragsgraphen.

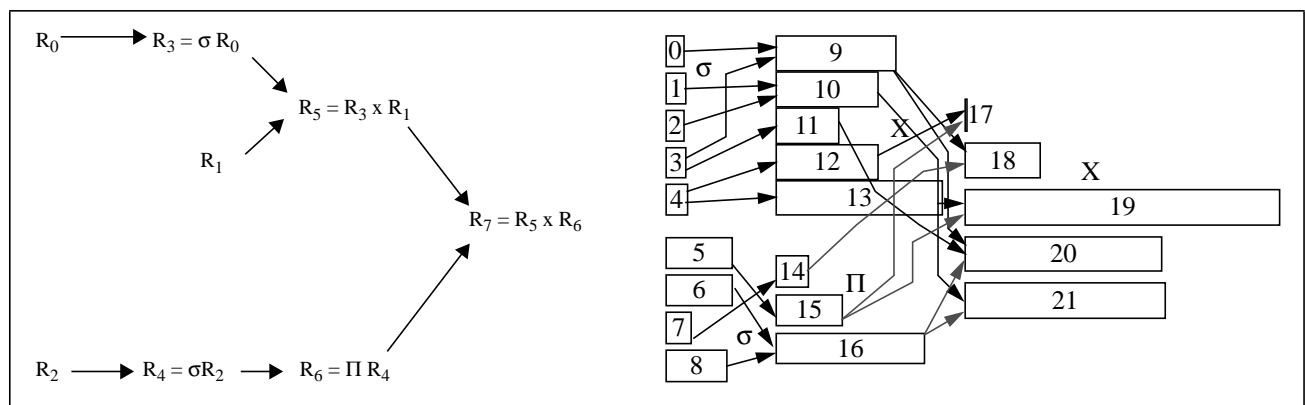


Abbildung 23: Operatorbaum und Ausführungsgraph für das Beispielszenario.

5.2.4 Datenbankoperationen auf geometrischen Objekten

Die vierte realisierte Anwendung führt geometrische Operationen auf Polygonen durch. Polygone werden in Relationen abgespeichert; zu jeder Polygon-Relation wird zusätzlich ein R-Baum verwaltet, der inhaltsbezogene Zugriffe beschleunigt. Dies ist eine in datenbankbasierten CAD-Systemen und geographischen Anwendungen übliche Struktur, die inhaltsbezogene Zugriffe wie Selektionen oder Verbundoperationen mit räumlichen Suchprädikaten wie Überlappung, Enthaltensein oder Höchstabstand zwischen geometrischen Objekten effizient ermöglicht.

Sowohl die Polygone einer Relation als auch die Knoten des zugehörigen R-Baums sind partitioniert und auf verschiedene Dateien verteilt. Abbildung 24 skizziert einen R-Baum, wobei die Rechtecke die Partitionierung der Daten angeben. Die Anwendung realisiert eine Mischung von Einfüge- und Verbundoperationen auf mehreren Polygon-Relationen, die größtenteils parallel ablaufen dürfen. Das Aufsuchen und Kombinieren von Tupeln besteht nur aus lesenden Zugriffen auf die Zugriffs- und Datenstrukturen. Diese Operationen können durch Verteilung der Polygone und der passenden R-Baum-Teile bzw. durch Anlegen von Kopien effizient parallelisiert werden.

Charakteristisch sind für diese Anwendung die wellenweise auftretenden Aufträge, denn die graphische Aufbereitung und geometrische Manipulation komplexer Objekte in höheren Anwendungsschichten resultiert auf Datenbankebene in einer Vielzahl von Scan- und Verbundoperationen. Das Laden und Generieren geometrischer Konstruktionen erzeugt eine Reihe von Einfügeoperationen einfacher Grundobjekte im Datenbanksystem. Für die Evaluierung wurden daher zwei Auftragsstypen, die Einfügeoperation und die Verbundberechnung, realisiert. Die Parallelität sowie der Ablauf und die Auftragsgrößen sind relativ frei konfigurierbar, da es sich bei der vorliegenden Anwendung nur um ein Ablaufskript handelt. Eine in der Praxis darauf aufsetzende Anwendung wie z.B. CAD wurde nicht realisiert, sondern nur deren typische Anfrageformen an die Datenverwaltung nachgebildet. Die Einfügeoperationen sind sehr kleine Aufträge, die Änderungen auf mehreren globalen Datensätzen durchführen. Die genauen Datenreferenzen sind schwer vorherzusagen. Die Verbundberechnungen bilden sehr große Aufträge, da jede Verbundberechnung sequentiell abläuft. Der Rechenaufwand und die Datenreferenzen sind auch hier schwer vorhersehbar, aber es wird nur lesend auf Daten zugegriffen und der Rechenaufwand ist im Vergleich zum Datenkommunikationsaufwand recht groß. Im allgemeinen lohnt sich daher eine beliebige Parallelisierung mehrerer anstehender Verbundoperationen. Die Einfügeoperationen sind daher datengebunden, die Verbundoperationen eher rechenleistungsgebunden.

Bei Einfügeoperationen müssen Zugriffs- und Datenpartitionen modifiziert werden. Im Gegensatz zu eindimensionalen B^+ -Bäumen beschränken sich die Änderungen meist

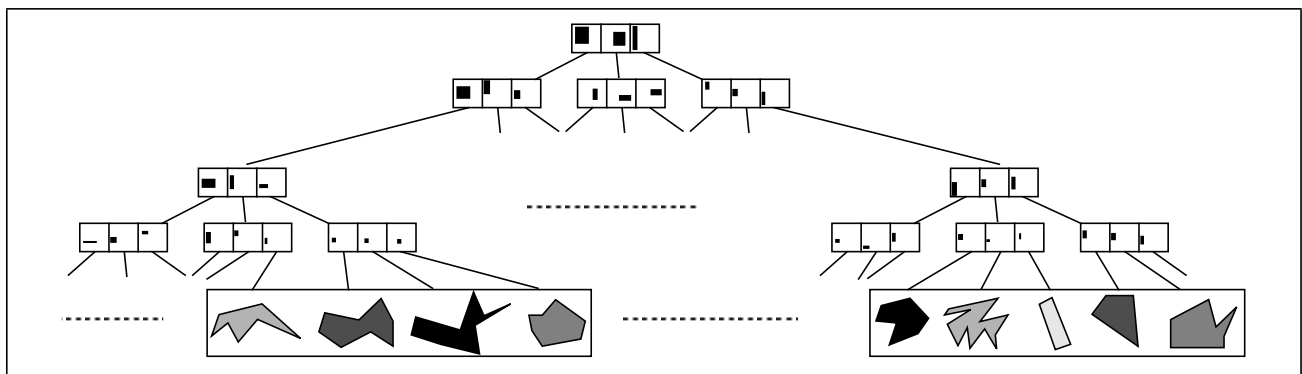


Abbildung 24: Verteilte Datenstrukturen zur Verwaltung der Polygone.

nicht auf die Blätter, sondern die umgebenden Rechtecke müssen relativ weit den Baum hinauf angepaßt werden. Parallele Einfügeoperationen verlangen daher von der Lastbalancierung gute Ausnutzung der Datenaffinität und schränken die Anzahl der Datenkopien ein. Ein weiteres Problem besteht darin, daß Einfügeoperationen sehr feingranular sind, während Verbundoperationen relativ große, jeweils sequentielle, Aufträge sind. Für die Messungen werden insgesamt $5 * 510$ Polygone eingefügt und $5 * 3$ Verbunde mit räumlichen Prädikaten berechnet; Abbildung 25 zeigt das Ablaufprofil, das ein Client erzeugt. Die Auftragsgrößen schwanken zwischen 2 und 700 Sekunden. Die Polygone sind in 27 Dateien zu je ca. 10 KBytes partitioniert, die Zugriffstrukturen belegen 350 Dateien einer Größe um je 100 Bytes.

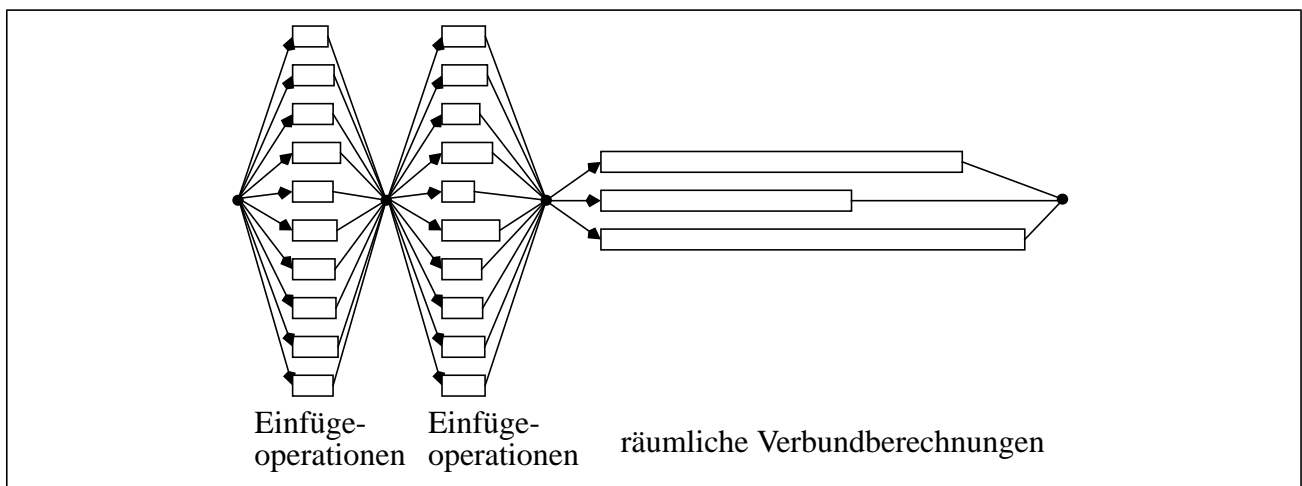


Abbildung 25: Möglicher Ablauf einer Anwendung auf R-Bäumen.

5.2.5 Parallele Spannungsberechnung nach der Methode der finiten Elemente

Die Methode der finiten Elemente wird vor allem in der Mechanik und Thermodynamik angewandt, um das Verhalten von Konstruktionen und Körpern unter Einwirkung von Kräften und Temperatureinflüssen durch Simulation zu untersuchen. Die zu unter-

suchenden Gebilde werden geeignet in eine Vielzahl kleiner Elemente zerlegt. Innerhalb der Elemente und zwischen benachbarten Elementen werden die physikalischen Gesetze, die gewöhnlich durch Differentialgleichungen spezifiziert sind, mit Hilfe von numerischen Näherungsverfahren berechnet. Dies führt insgesamt zu einem großen Gleichungssystem, das die Einflußgrößen der Einzelelemente aufeinander enthält. Die Lösung des Gleichungssystems ergibt einen Ergebnisvektor, der die Verschiebungen, Spannungen, Drücke oder Temperaturen der einzelnen Elemente des Gebildes enthält.

Die Berechnung der gegenseitigen Einflüsse der einzelnen Elemente auf das Gesamtsystem hängt sehr stark vom betrachteten physikalischen Problem, von der Struktur der Elemente, der Art der Randbedingungen und dem Grad der Ansatzfunktionen zur näherungsweise Lösung der durch Differentialgleichungen beschriebenen Gleichgewichtsbedingungen ab. Hier wurde ein einfaches Problem mit einem einzigen Elementtyp und linearen Ansatzfunktionen betrachtet: Es werden Spannungen und Dehnungen an einem Stab berechnet, der an einer Seite fixiert ist, und auf der anderen Seite belastet wird. Die Belastung und Verteilung der Fixknoten, die Ausmaße des Stabs und das Granulat der Diskretisierung können variiert werden. Die Lösung des großen Gleichungssystems erfolgt, zumindest in parallelen Implementierungen, üblicherweise durch das iterative Näherungsverfahren der konjugierten Gradienten. Die parallelisierte Anwendung realisiert kein generisches Finite-Elemente-Berechnungspaket, sondern die Berechnung eines speziellen Problems und Lösungsansatzes. Um andere Problemstellungen zu lösen, muß jedoch lediglich das Modul zur Berechnung der Einflüsse der einzelnen Elemente und die Module zur Berücksichtigung der Lasten und der Randbedingungen (die unten beschriebenen ersten drei Schritte) ausgetauscht werden. Das Laufzeitverhalten des speziellen Szenarios ist charakteristisch für alle Berechnungen nach der Methode der finiten Elemente; Variationen können nur im Zugriffsmuster auf die globale Matrix des Gleichungssystems bei den ersten drei Schritten auftreten, da es vom Numerierungsschema der Elemente und Knoten, von der Geometrie der Körper und von der Kardinalität der Nachbarschaftsbeziehungen abhängt.

Weiterhin wird hier nur eine statische Berechnung durchgeführt. Bei Berechnung von zeitabhängigen Vorgängen ist diese Berechnung für jeden Zeitschritt wiederholt durchzuführen.

Eine Berechnung besteht im wesentlichen aus 5 Schritten, die nacheinander ausgeführt werden (Abbildung 26):

1. *Einlesen der Szenenbeschreibung*: Hier werden die Elemente und Knoten des Körpers aus einer Datei eingelesen. Außerdem wird der Einfluß der Lasten, d.h. der von außen auf die Knoten einwirkenden Kräfte, berechnet.

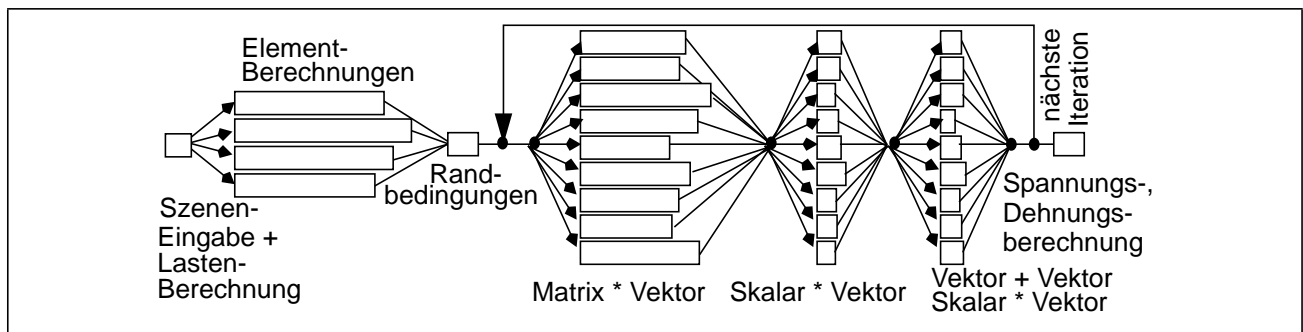


Abbildung 26: Ablauf einer Berechnung nach der Methode der finiten Elemente.

2. *Elementberechnung*: Für jedes Element des diskretisierten Körpers wird sein Einfluß auf das Gesamtsystem berechnet. Das Granulat der Parallelität kann durch eine Konstante eingestellt werden, die angibt, wieviele Elemente durch je einen Auftrag berechnet werden.
3. *Berücksichtigung der Randbedingungen*: Im verwendeten Szenario müssen die Einflüsse der fixierten Knoten auf das System einbezogen werden.
4. *Lösung des linearen Gleichungssystems*: Das Gleichungssystem beschreibt das globale Kräftegleichgewicht und ergibt die Verschiebungen und damit Spannungen und Dehnungen der Einzelelemente. Diese Phase besteht aus einer Sequenz von Iterationen, bis der verbleibende Fehler klein genug ist. Die Anzahl der notwendigen Iterationen ist problemabhängig. Jede Iteration besteht aus folgender Sequenz von Berechnungsschritten, wobei jeder Schritt in eine Menge paralleler Aufträge aufgespalten ist; Das Granulat der Parallelität kann wiederum durch eine Konstante eingestellt werden kann, die angibt, wieviele Zeilen bzw. Spalten der Matrix und der Vektoren in je einem Auftrag berechnet werden.
 - 4.1. Eine Matrix-Vektor-Multiplikation, die nach Zeilen der Matrix parallelisiert ist. Es werden nur die tatsächlich von Null verschiedenen Elemente der dünnbesetzten Matrix multipliziert. Dieser Schritt ist der aufwendigste der drei Teilschritte. Die Teilschritte 4.2 und 4.3 sind vor allem deshalb parallelisiert, um die Verteilung der Daten weiter zu nutzen und nicht alle Daten an einen zentralen Knoten schicken zu müssen.
 - 4.2. Zwei Skalar-Vektor-Multiplikationen zur Aktualisierung des Verschiebungsvektors und des Gradientenvektors und ein Skalarprodukt zur Bestimmung des Residuums. Diese Operationen sind ebenfalls nach Zeilen der Vektoren parallelisiert. Sie können jedoch nicht unmittelbar an die entsprechenden Berechnungen des Schritts 4.1 gehängt werden, weil der benötigte Faktor erst am Ende aller Teilberechnungen von Schritt 4.1 feststeht.
 - 4.3. Zwei Vektor-Vektor-Additionen und eine Skalar-Vektor-Multiplikation zum Vorschreiten des Lösungsvektors entlang des Gradienten. Auch diese Operationen sind nach Zeilen der Vektoren parallelisiert und können nicht unmittelbar an die ent-

sprechenden Berechnungen des Schritts 4.2 gehängt werden, weil der benötigte Faktor erst am Ende aller Teilberechnungen von Schritt 4.2 feststeht.

5. *Berechnung der Spannungen und Dehnungen in den einzelnen Elementen*: Diese Phase wird wiederum als ein einziger sequentieller Auftrag realisiert, da sie verhältnismäßig wenig Zeit beansprucht.

Abbildung 27 zeigt das Ergebnis einer Berechnung. Die Spannungen sind durch Helligkeitsabstufungen visualisiert, wobei dunkle Färbung starke Spannung bedeutet.

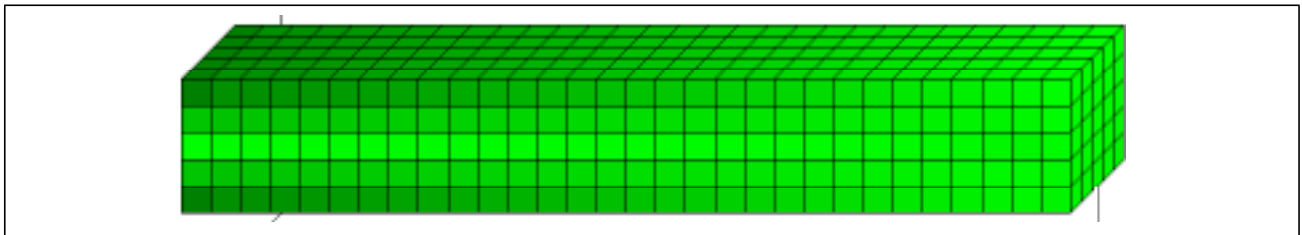


Abbildung 27: Visualisierung der berechneten Spannungen eines Beispielstabs.

Die globalen Daten für eine Berechnung bestehen aus folgenden Strukturen (die gemeinsamen Datenstrukturen sind in Blöcke mit je einer festen Anzahl an Elementen pro Block partitioniert, um paralleles Arbeiten auf den Daten zu ermöglichen):

- Beim Einlesen der Szenenbeschreibung aus einer Datei wird eine Liste der Knoten (Koordinaten und ein Indikator, ob fixiert oder beweglich), eine Liste der Elemente (mit je 8 Eckknotennummern), und ein Kraftvektor (Kräfte, die auf die Knoten einwirken) erzeugt. Weiterhin wird die globale Steifigkeitsmatrix in einer für dünnbesetzte Matrizen optimierten Speicherungsform angelegt: Für jeweils K Zeilen werden nur die von Null verschiedenen Elemente in einer Liste gespeichert.
- Während der Elementberechnungen werden die Einflüsse der Elemente in der globalen Steifigkeitsmatrix aufsummiert (assembliert).
- Für die Lösung des Gleichungssystems werden drei Vektoren benötigt. Der Residuenvektor enthält den Restfehler während der Gleichungslösung. Für den Relaxationsrichtungsvektor wird der Kraftvektor weiterverwendet. Zur Zwischenspeicherung des Matrix-Vektor-Produktes wird ein weiterer Vektor als globaler Datensatz verwendet. Weiterhin wird ein Vektor für das Ergebnis initialisiert, der die Verschiebungen der Knoten enthält. Er wird während der Lösung des Gleichungssystems iterativ korrigiert.

Diese Anwendung verlangt von der dynamischen Lastbalancierung, daß die parallelierten Schleifen möglichst gleichmäßig auf die Knoten verteilt werden, um abfallende Parallelität vor Synchronisationspunkten zu vermeiden. Andererseits sollte durch Wiederverwendung derselben Server für Indexbereiche Datenaffinität genutzt werden, da die Aufträge teilweise relativ kurz sind, aber große Datenmengen modifizieren.

5.3 Leistungssteigerung durch dynamische Lastbalancierung

Die tatsächlich erreichte Leistungssteigerung durch ein dynamisches Lastbalancierungskonzept muß durch Vergleich mit unbalancierten Abläufen bewertet werden. In den häufiger anzutreffenden dezentralen Lastbalancierungsverfahren erhält man unbalancierte Referenzabläufe, indem man jeden Auftrag auf dem Knoten ausführen läßt, auf dem er initiiert wurde. Wenn auf dem System jedoch nicht nur unkorrelierte sequentielle Aufträge, sondern auch parallele Anwendungen ablaufen, ist es bereits schwieriger, einen geeigneten unbalancierten Ablauf zu erhalten. Wenn solche Anwendungen bereits einen eigenen Verteilungsmechanismus enthalten, so muß die Leistungssteigerung des automatischen Lastbalancierungsverfahrens gegenüber diesen vorhandenen anwendungsinternen Mechanismen bewertet werden. Parallele Anwendungen, die selbst keine Auftragsverteilung vornehmen, benötigen einen Lastverteilungsmechanismus; hier können zum Vergleich nur triviale, primitive Balancierungsverfahren betrachtet werden. Das gilt auch für zentrale Lastbalancierungsverfahren wie im *HiCon*-Konzept: Bei zentralen, nicht-preemptiven Lastbalancierungsverfahren gibt es den unbalancierten Fall von Natur aus nicht, sondern Aufträge können nur mit mehr oder weniger aufwendigen Strategien zugewiesen werden. Auch wenn Aufträge lokal, d.h. auf ihrem Entstehungsknoten, bearbeitet werden, gehen sie den Weg über die zentrale Lastbalancierungskomponente. Ein Grundaufwand ist daher stets vorhanden (Abschnitt 5.4), und eine einfache Lastverteilung durch Reihum-Zuweisung oder Zufallsverteilung zeigt in vielen Fällen bereits eine deutliche Verbesserung gegenüber unbalancierten Abläufen in dezentralen Ansätzen.

In diesem Abschnitt soll die tatsächlich erreichbare Leistungssteigerung des zentralen Balancierungskonzepts im *HiCon*-Modell für jeden Anwendungstyp und für Mischlasten evaluiert werden. Das Potential zur Durchsatzsteigerung durch die dezentrale Balancierungsstrategie im *HiCon*-Modell wird lediglich in Abschnitt 5.5 grob evaluiert.

Die Anwendungen werden jeweils alleine sowie im Mehrbenutzerbetrieb unter *HiCon*-Lastbalancierung, mit simpler Lastbalancierung (zufällige Verteilung der Aufträge) und ohne Lastbalancierung beobachtet. Ohne Lastbalancierung läuft jede Anwendung sequentiell auf dem Knoten des Client ab. Im Mehrbenutzerbetrieb wurde im Falle der Flächenerkennung und der Datenbankoperationen bereits eine recht günstige, d.h. balancierte Situation vorgegeben, da jeder Client, d.h. jede der Anwendungen, auf einem einzelnen Knoten ablief. Die Prozessoren sind jedoch unterschiedlich leistungsfähig. Bei der Finite-Elemente-Berechnung wurde im Mehrbenutzerbetrieb ein Lastungleichgewicht vorgegeben, indem nur auf dreien der fünf Knoten Anwendungen gestartet wurden. Als Rechnersystem wurde das in Abbildung 28 links oben gezeigte heterogene Workstation-Cluster gewählt. Im Bild sind auch die real gemessenen Lauf-

zeiten der Anwendungen gegenüber gestellt. Im Mehrbenutzerbetrieb ergibt sich die Zeit für den unbalancierten Ablauf aus der Zeit für die Anwendung auf dem langsamsten Knoten; Die unbalancierten Einzelanwendungen wurden hingegen auf einem Knoten mittlerer Leistung gemessen.

Eine Zufallsverteilung erbringt durch die Parallelität bereits eine starke Beschleunigung aller einzelnen Anwendungen sowie des R-Baum-Operationsmixes (eine Folge jeweils 100 parallelen Polygon-Einfügeoperationen, abgeschlossen von 15 parallelen aufwendigen Verbundoperationen). Dennoch kann die Lastbalancierung durch Berücksichtigung der unterschiedlichen Rechnerleistungen, der tatsächlichen aktuellen Systemlast, der verschiedenen Auftragsgrößen und der Datenkommunikation zwischen den Servern eine weitere Verbesserung um 24% (Bilderkennung), 16% (FE-Berechnung), 44% (Datenbank) bzw. 31% (Wegesuche) erreichen. Die Verbesserung um weitere 10% resultiert beim R-Baum-Operationsmix vor allem aus der Einschränkung der genutzten Parallelität in Phasen hoher Einfügeparallelität.

Im Mehrbenutzerbetrieb ohne Balancierung hängt die Gesamtlaufzeit von den Anwendungen auf den langsamen Knoten ab, da jeweils Anwendungen gleichen Aufwands gewählt wurden. Eine einfache Lastbalancierung, die alle Aufträge wahllos verteilt, verschlechtert bei der Flächenerkennung sogar den Durchsatz, weil hohe Datenkommunikationskosten entstehen, obwohl die Rechenkapazitäten im Mittel besser genutzt werden könnten als ohne Lastbalancierung. Die Lastbalancierung nach dem *HiCon*-Modell bewirkt bei den Flächenerkennungs-Rechnungen trotz der im Prinzip mit Anwendungen gleich beladenen Rechner eine Durchsatzsteigerung von 5.4%, indem sie unter Beachtung der entstehenden Kommunikationskosten, d.h. geeigneter Partitionierung der Daten, die Anwendungen parallel laufen läßt und stets die Phasen geringerer Parallelität in einer Anwendung für Berechnung anderer Anwendungen nutzt. Eine Zufallsverteilung der Finite-Elemente-Berechnung im Mehrbenutzerbetrieb verbessert den Ablauf bereits um deutliche 34%, da sie die beiden freien Prozessoren nutzt. Die *HiCon*-Lastbalancierung kann den Durchsatz jedoch um weitere 17% steigern. Die Datenbankoperationen sind aufgrund der Abhängigkeiten innerhalb der einzelnen Anfragen und wegen der sehr unterschiedlichen Auftragsgrößen schwer effizient zu balancieren. Eine wahllose Verteilung des Mehrbenutzerbetriebs erbringt 18% Steigerung, was durch geeignete Lastbalancierung um weitere 28% gesteigert werden kann. Vergleichsmessungen ergeben, daß allein durch priorisierte Zuweisung der kritischen Aufträge ca. 10% Gewinn erzielt werden. Bei der recht simplen Wegesuche erbringt Lastbalancierung weitere 42%, weil durch Abschätzung der Auftragsgrößen und späte Zuweisung innerhalb jeder Iteration alle Knoten bis zum Ende genutzt werden.

Zuletzt soll ein heterogenes Gemisch von Lasten auf dem parallelen System unter verschiedenen Balancierungsstrategien und -strukturen beobachtet werden. Dazu werden

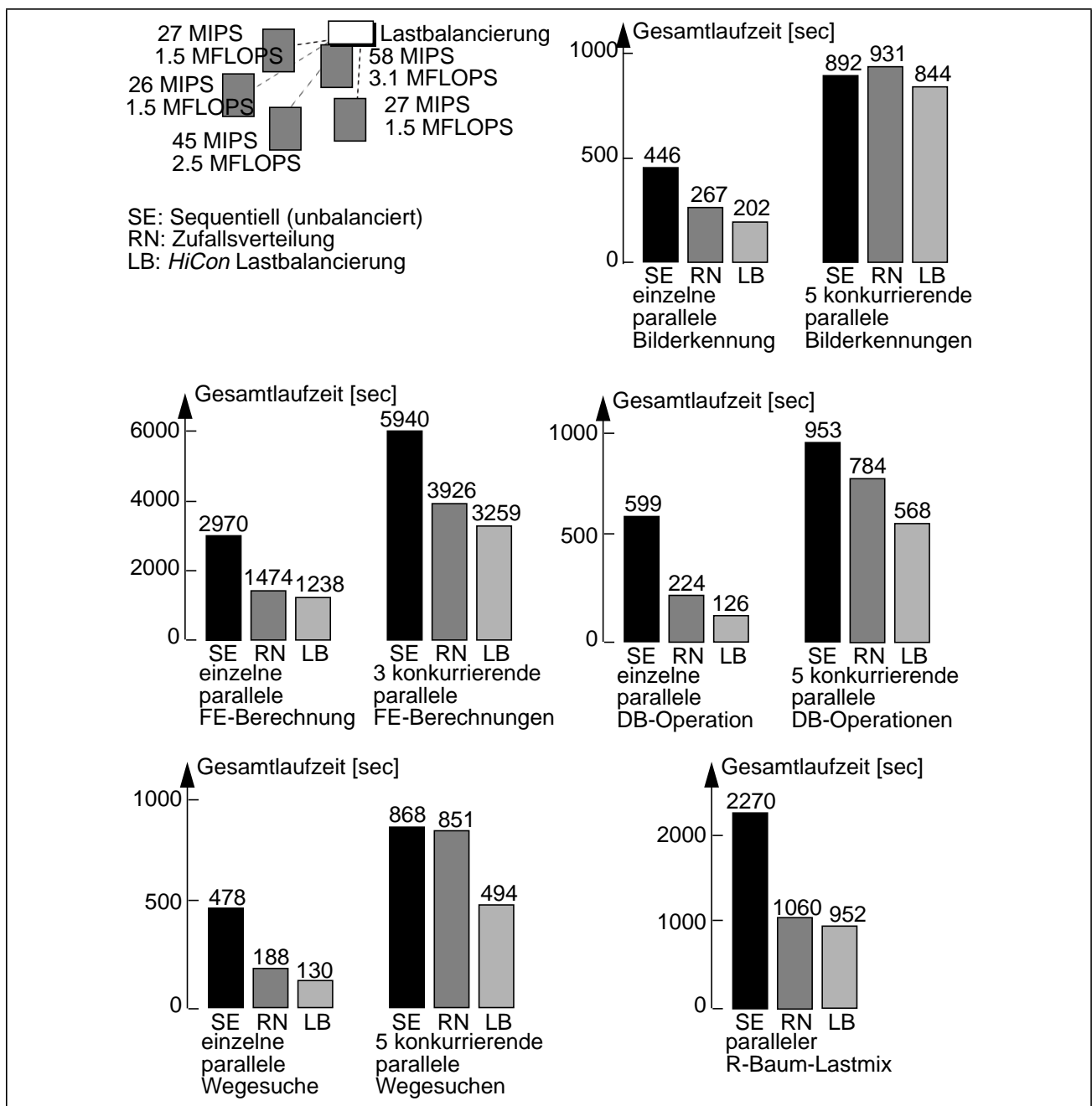


Abbildung 28: Systemkonfiguration und Laufzeiten der Anwendungen im Ein- und Mehrbenutzerbetrieb bei unterschiedlicher Lastbalancierungsunterstützung.

Flächenerkennungen, Finite-Elemente-Berechnungen und Datenbankoperationen jeweils zweifach konkurrierend abgewickelt. Dabei wurden die Problemstellungen so dimensioniert, daß alle 6 Anwendungen etwa denselben Gesamtrechenaufwand verlangen. Abbildung 29 zeigt die Konfiguration und die Laufzeiten bei verschiedener Lastbalancierungsunterstützung: Weder die Zuweisung an den ersten jeweils freien Server der Klassen noch eine Zufallsverteilung können der Komplexität des Balancierungs-

problems gerecht werden. Erst die komplexe Strategie erreicht eine hohe, sinnvolle Systemnutzung und reduziert den Datenkommunikationsaufwand.

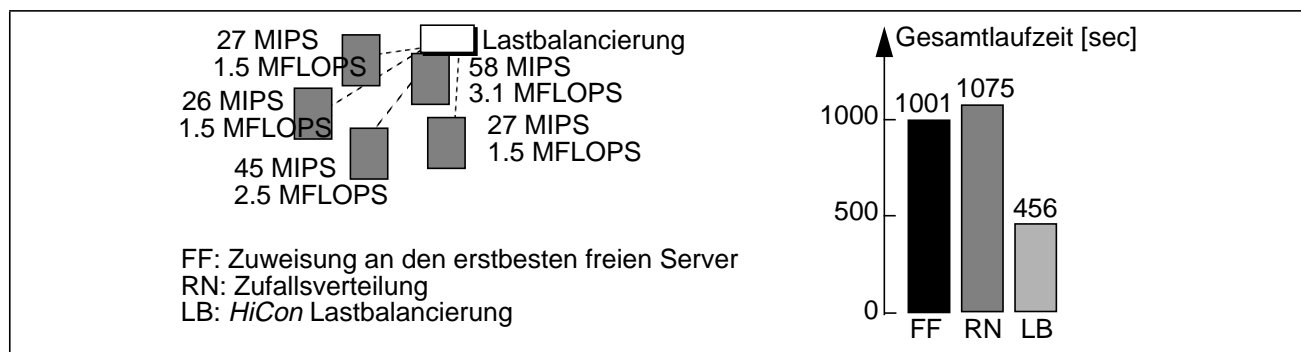


Abbildung 29: Konfiguration und Laufzeiten des heterogenen Lastszenarios.

5.4 Zusatzaufwand durch dynamische Lastbalancierung

Dynamische Lastbalancierung bringt als Betriebssystemdienst neben zusätzlicher Komplexität auch zusätzliche Kosten mit sich. In diesem Abschnitt sollen die Kosten des *HiCon*-Ansatzes anhand einiger Meßreihen betrachtet werden. Die Beurteilung des Zusatzaufwands bringt drei Probleme mit sich: Erstens läßt sich der Aufwand des Laufzeitsystems zur Auftrags- und Datenverwaltung schlecht vom eigentlichen Lastbalancierungsaufwand trennen, da die Lastbalancierungsfunktionalität in die Prozeß- und Kommunikationsstruktur des Laufzeitsystems eingebunden ist. Zweitens besteht die Grundidee des *HiCon*-Ansatzes darin, die Lastbalancierung effizient in ein verteiltes Betriebssystem zu integrieren. Der Zusatzaufwand, der in dem realisierten Prototyp entsteht, ist also nicht so entscheidend, weil die Realisierung Betriebssystem-unabhängig auf Anwendungsebene erfolgte. Drittens können keine völlig unbalancierten Abläufe ohne Zusatzaufwand als Vergleich herangezogen werden: In vielen anderen Ansätzen beobachtet die Balancierung lediglich ein laufendes System und optimiert den Durchsatz, indem sie neue Aufträge auf andere Knoten umlenkt - meist werden nur unabhängige sequentielle Aufträge betrachtet, die ohne Balancierung auf dem Knoten gestartet würden, auf dem sie entstanden sind und indem sie laufende Aufträge (Prozesse) auf andere Knoten migriert. Im *HiCon*-Konzept werden jedoch innerhalb eines Clusters alle Aufträge zentral zugewiesen und keine laufenden Aufträge migriert. Ohne zentrale Auftragsverteilung würden alle Aufträge auf den Rechenknoten bearbeitet werden, auf denen der Client der Anwendung liegt, was bei großen parallelisierten Anwendungen katastrophal wäre. Die Lastbalancierung zwischen Clustern kann analog zu dezentralen Ansätzen abgeschaltet werden, um den Zusatzaufwand beurteilen zu können. Die zentrale Lastbalancierung kann allenfalls auf Zufallsverteilung eingestellt, nicht aber ganz herausgenommen werden. Im Folgenden werden die verschiedenen Kostenfaktoren für die dynamische Lastbalancierung gemessen.

5.4.1 Rechenaufwand für Lastbalancierungsentscheidungen

Lastbalancierungsentscheidungen müssen zur Laufzeit auf den Knoten des Systems berechnet werden und verbrauchen somit Rechenzeit, die ansonsten zur Anwendungsverarbeitung genutzt werden könnte. Im *HiCon*-Ansatz werden die Entscheidungen innerhalb eines Clusters durch einen zentralen Prozeß auf einem der Knoten berechnet. Da hier Systeme mit einer mäßigen Anzahl leistungsfähiger Rechenknoten je Cluster zugrundegelegt werden, kann kein Knoten für Lastbalancierung dediziert werden, sondern auf einem der Knoten werden zusätzlich die Lastbalancierungsentscheidungen durchgeführt. Wie in Abschnitt 3.7.5 beschrieben, wird diese Belastung adaptiv in den Entscheidungen berücksichtigt.

Hohe Rechenlast für Lastbalancierungsentscheidungen ist bei einer großen Anzahl (und damit einer hohen Auswahlmöglichkeit) an Servern und einer hohen Ankunftsrate von Aufträgen zu erwarten. Sie ist außerdem um so höher, je schwieriger die Aufträge zu beurteilen sind, d.h. zu entscheiden, wann und wohin sie zugewiesen werden sollten. In Situationen hoher Gesamtlast im System hält der Lastkontrollmechanismus der Lastbalancierung verstärkt Aufträge zurück; trotzdem müssen bei jeder Situationsänderung, die den Entscheidungsalgorithmus aktiviert, die zentral einbehaltenen Aufträge erneut bewertet werden, um zu ermitteln, ob sie nun auf andere, weniger belastete Rechenknoten zugewiesen werden können. Wenn alle Knoten ausgelastet sind, ist keine Neubewertung notwendig und bei Überlastung der Balancierungskomponente wird auch auf Neubewertung verzichtet (Abschnitt 3.7.5). Hohe Gesamtlast im System bei ungleichverteilter Knotenauslastung kann also die Anzahl der Entscheidungsversuche pro Auftrag erhöhen. Ungleiche Knotenauslastung ist bei Abwicklung weniger komplexer parallelisierter Anwendungen aufgrund von Datenkommunikationserwägungen oft sinnvoll.

Bei kleinen Clustern, hoher Ressourcenauslastung und großen Anzahlen aufwendig zu balancierender Aufträge konsumiert der *HiCon*-Lastbalancierungsdienst typischerweise 1%, in Extremfällen bis zu 12.5% der Gesamtrechnkapazität des Systems. Die Lastkontrolle verhindert ja, daß die Knoten durch parallele Aufträge überlastet werden; wenn jedoch die Balancierungskomponente selbst überlastet wird, dann vereinfacht sie ihren Entscheidungsaufwand adaptiv, wodurch sowohl der Rechenaufwand als auch der Gewinn durch Lastbalancierung sinkt (Abschnitt 5.6.2). Als Extrembeispiel kann das Diagramm in Abschnitt 5.6.2, unten links in Abbildung 40 betrachtet werden. Die schwarz gefüllte Kurve zeigt den Rechenbedarf der Balancierungskomponente (ohne Überlastschutz) für ein sehr aufwendiges Szenario mit Flächenerkennung im Mehrbenutzerbetrieb.

5.4.2 Verzögerung durch Entscheidungsfindung

Neu entstandene Aufträge werden im *HiCon*-Ansatz nicht unmittelbar durch einen Server bearbeitet, sondern zunächst in einer zentralen Warteschlange gepuffert, bis die Lastbalancierung die Zuweisung an einen Server für günstig hält und die Entscheidung über den bestgeeigneten Server getroffen hat. Intuitiv muß diese Zeit bis zum Bearbeitungsbeginn zur Antwortzeit des Auftrags mitgezählt werden, denn sie verzögert den Ablauf komplexer paralleler Anwendungen mit Reihenfolgeabhängigkeiten und kann Leerlaufzeiten im System bewirken. Die Wartezeiten durch Aufenthalt in der zentralen Warteschlange sind jedoch kein wirklich negativer Effekt der Lastbalancierung, denn sie dienen nur der Lastkontrolle auf den Rechenknoten und der möglichst späten Zuweisungsentscheidung. Aufträge werden solange in der zentralen Warteschlange aufbewahrt, bis der Knoten des momentan bestgeeigneten Servers durch die Bearbeitung des Auftrags nicht überlastet wird und der Server den Auftrag in Kürze bearbeiten kann, bevor die Zuweisungsentscheidung veraltet ist. Nachteilig ist also lediglich die Verzögerung des Auftragsbearbeitungsbeginns durch den Zeitbedarf für die Zuweisungsentscheidung. Ebenso sollen die Wartezeiten der Aufträge in den lokalen Warteschlangen der Server nicht betrachtet werden; im *HiCon*-Modell wird ja nicht für jeden Auftrag ein Bearbeitungsserver kreiert, sondern ein existierender Serverprozeß wird wiederverwendet, um Prozeßerzeugung und -Terminierung, Prozeßwechsel und den Speicherbedarf durch viele Prozesse auf den Rechenknoten auf ein vernünftiges Maß beschränken.

Die Verzögerungszeiten durch die Entscheidungsberechnung innerhalb eines Clusters wurden anhand der Szenarios aus Abschnitt 5.3 quantifiziert. Der mittlere Zeitbedarf pro Auftrag für Entscheidungen inklusive Bewertungen betrug im Mittel 16...25 msec. Die Auftragsgrößen liegen, je nach Phasen der Anwendungen, im Bereich von 0.7...40 sec. Auf üblichen Netzwerken (Ethernet) kommt je Auftrag eine Sendeverzögerung von ca. $2 \cdot 500 \mu\text{sec}$ hinzu, da jeder Auftrag vom Client über die Balancierungskomponente zum Server und das Ergebnis über die Lastbalancierung zurück geschickt wird (zwei zusätzliche Nachrichten). Die Verzögerungen durch die Balancierung liegen daher in der prototypischen Realisierung pro Auftrag im Bereich von $\frac{0.016 \dots 0.025 + 2 \cdot 0.0005}{0.7 \dots 40} = 0.04 \dots 3.7\%$.

5.4.3 Kommunikationslast durch Informations- und Auftragsaustausch

Durch die zentrale Lastbalancierung entsteht pro Auftrag, wie oben erklärt, neben den notwendigen (meist kurzen) Aufruf- und Resultatnachrichten zwischen Client und Server ein Zusatzaufwand von zwei kurzen Nachrichten (ca. 50 Bytes). In der Flächenerkennungsanwendung mit phasenweise sehr feinem Auftragsgranulat werden, bei guter Balancierung in kleinen Clustern (5 Knoten), im Mittel 15 Datenkommunikationsnach-

richten je Auftrag gesendet, von denen 66% kurze und 33% lange Nachrichten (100..3000 Bytes) sind. Die Nachrichten zur Messung der Knotenauslastungen können vernachlässigt werden. Zentrale Lastbalancierung erzeugt daher grob

$$\frac{2 \cdot 50\text{Byte}}{(10 + 2) \cdot 50\text{Byte} + 5 \cdot 1550\text{Byte}} = 1.2\% \text{ Zusatzlast auf dem Netzwerk.}$$

Die Lastbalancierungskomponenten benachbarter Cluster tauschen Lastinformationen mit Nachbarn aus und schicken sich gegenseitig Aufträge zu. Hierzu wurde ein Szenario betrachtet, in dem 4 Cluster mit je 3 - 4 heterogenen Knoten dezentral gekoppelt waren. Ein extremes Lastungleichgewicht wurde erzeugt, indem 7 parallele Flächenerkennungen in einem der Cluster gestartet wurden. Dadurch wurden fast alle Anwendungen mit insgesamt mehreren tausend Aufträgen zwischen Clustern verschoben. In diesem Szenario wurden im Gesamtsystem im Mittel $6 + 278$ Bytes pro Sekunde (Bps) zur Lastinformation bzw. Auftragsverschiebung zwischen Clustern verschickt. Bei 76842 Bps Nutz-Nachrichtenaufkommen und 643 Bps Nachrichtenaufwand zur zentralen Lastbalancierung, erzeugte die Lastbalancierung insgesamt etwa $\frac{284\text{Bps} + 643\text{Bps}}{76842\text{Bps}} = 1.2\%$ Zusatzlast auf dem Netzwerk.

5.5 Skalierbarkeit der Lastbalancierung

In der Literatur ist es üblich, die “unbegrenzte Skalierbarkeit” als eine der wichtigsten Anforderungen an Lastbalancierungskonzepte zu betrachten. Während alle realisierten Ansätze bisher Systeme in der Größenordnung von $10 - 10^2$ Knoten verwalten, werden fast ausschließlich Ansätze für Systeme der Größenordnung $10^2 - 10^5$ Knoten konzipiert. Während noch nicht klar ist, ob Systeme dieser Größenordnung sinnvoll in homogener Weise, d.h. ohne hierarchische Strukturierung im Betriebssystem, betrieben werden können, ist es offensichtlich, daß globale Lastbalancierung mit zentraler Informationsverwaltung oder zentraler Entscheidungsfindung nicht mehr möglich ist. Die früheren zentralen Ansätze wurden daher durch völlig dezentrale abgelöst, bei denen jeder Knoten Teilinformationen verwaltet und autonom Entscheidungen trifft. Der Verlust an Lastbalancierungspotential gegenüber zentralen Ansätzen wird selten evaluiert.

Lastbalancierungsansätze mit einer Struktur, die dem Clustering in realen Systemen entspricht, werden erst seit kurzem untersucht. Das liegt vor allem an der bisher sehr theoretischen Ausrichtung der Disziplin: während zentrale und völlig dezentrale Verfahren - nach starker Vereinfachung - durch Warteschlangenmodelle oder ähnliche stochastische Modelle geschlossen darstellbar sind und somit Konvergenz gegen eine Lastgleichverteilung bzw. garantierte Höchstabweichung von einer optimalen Balancierung nachgewiesen werden kann, können Clustering-Ansätze meist nur durch Simulation oder reale Messungen bewertet werden. Der *HiCon*-Ansatz soll hier ebenfalls aufgrund weniger Meßreihen auf seine Skalierbarkeit hin untersucht werden.

Zuerst soll ein großes Netzwerk von Workstations unter verschiedenen Clustering-Strukturen der Lastbalancierung beobachtet werden, um die Grenzen der zentralen Balancierung und die Güte dezentral kooperierender Lastbalancierungsagenten für Cluster zu ermitteln. Abbildung 30 zeigt drei verschiedene Lastbalancierungsstrukturen für ein großes Workstation-Netz. Hier wird speziell die Quad-Split- und Quad-Merge-Phase der Flächenerkennung betrachtet, in der sehr viele kurze, aber durch hohen Anteil exklusiver Zugriffe auf gemeinsame Daten nicht leicht balancierbare Aufträge in hoher Parallelität auftreten. Die Anwendung läuft 15-fach konkurrierend auf dem System ab, wobei durch die Lage der Clients bereits ein recht gutes Lastgleichgewicht zwischen den Knoten sowie zwischen den Clustern vorliegt. Auf jedem Knoten sind drei Server konfiguriert. Die dezentrale Struktur zeigt deutlich das beste Resultat (links unten in Abbildung 30). Die zentrale Lastbalancierung ist hier überlastet: Rechts unten im Bild ist die Warteschlange der zu bearbeitenden Ereignisse der Lastbalancierung gezeigt. Nachdem die Mehrzahl der Anwendungen etwa gleichzeitig die kritischen Phasen abgeschlossen haben, verkürzt sich die Ereigniswarteschlange der Lastbalancierung wieder. Abschnitt 2.5.6 stellt Konzepte zur Vermeidung solcher Überlastung vor, die hier nicht aktiviert wurden. Das System ist jedoch mit dieser Belastung deutlich zu groß für ein Cluster. Man beachte, daß die Balancierungskomponente hier zusätzlich durch die häufigen entfernten Datenzugriffe belastet wird, weil in der Realisierung die Datenortverwaltungskomponente im selben Prozeß realisiert ist wie die Lastbalancierung. Die gemischte und die völlig verteilte Struktur erzeugen weniger Balancierungsaufwand, weil im vorliegenden Szenario bereits eine gleichmäßige Vollauslastung des Gesamtsystems vorliegt, wodurch zwischen den Clustern wenig Interaktion stattfindet.

Während obiges Szenario ein idealer Fall für dezentrale Balancierung ist, soll ein zweites Szenario quantifizieren, daß zentrale Lastbalancierung ein deutlich höheres Potential zur Leistungssteigerung besitzt. In großen Systemen ist also stets durch vernünftiges Clustering ein Kompromiß zwischen Lastbalancierungsaufwand und -gewinn zu erreichen.

Suboptimale Lastbalancierung resultiert in dezentralen Strukturen auch aus dem Verlust und der Vergrößerung von Informationen zwischen den Clustern. Nirgends im System ist globale Zustandsinformation verfügbar, und Balancierung basiert auf partiellen Informationen. Information veraltet, bis sie in anderen Clustern verwendet werden kann, und durch die Aggregation von Informationen zwischen Clustern können wichtige Details verloren gehen. Die Messung vergleicht das durch Informationsverlust sinkende Balancierungspotential. Explizit verteilte Balancierungsstrategien (Abschnitt 2.5.3) verwenden nur sehr wenig Informationen und streben lediglich eine grobe Lastverteilung zwischen den Clustern an. Implizit verteilte Strategien können genauere Informationen verwenden. Da sie die Nachbarn jeweils als Server betrachten,

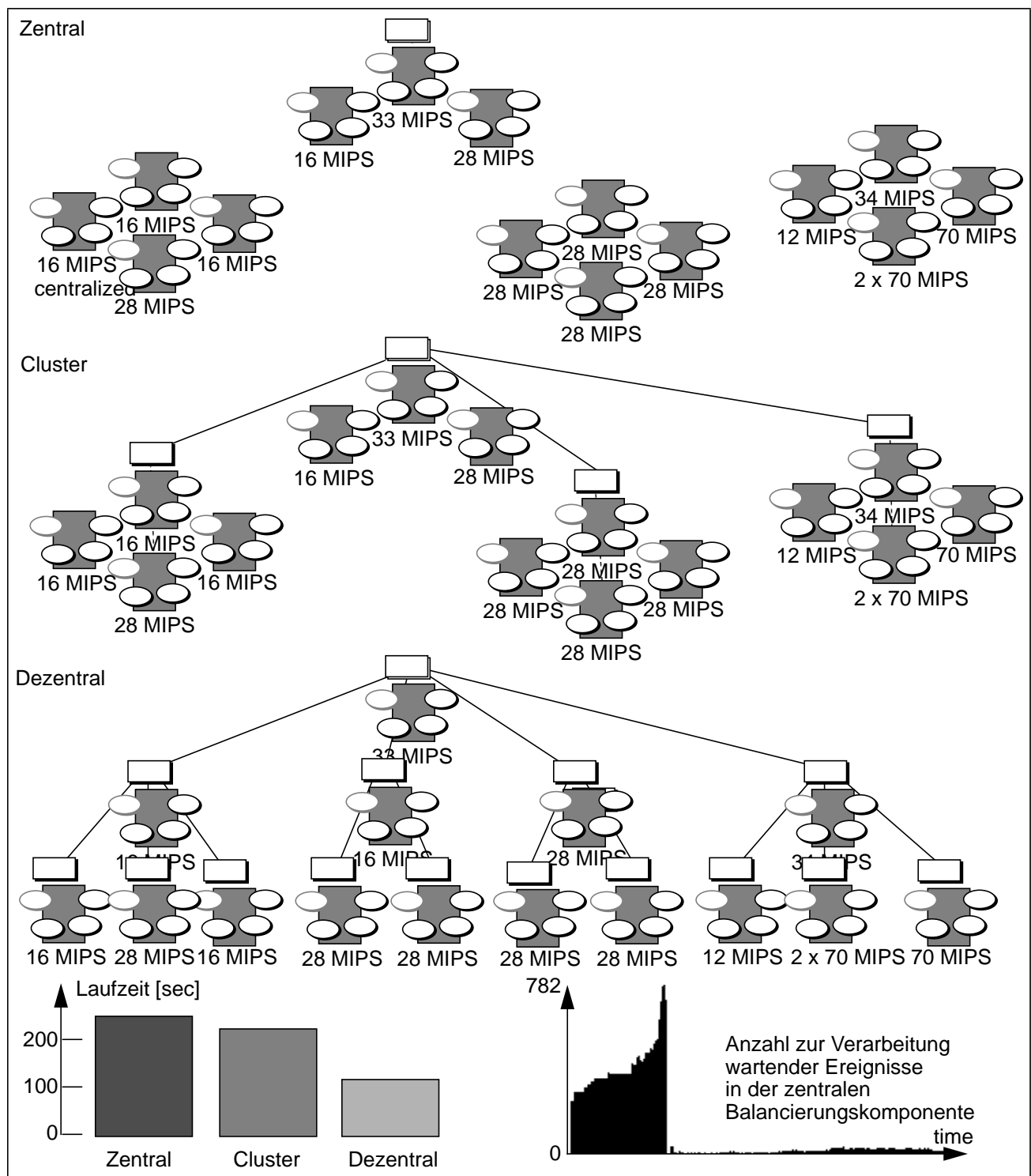


Abbildung 30: Konfiguration und Laufzeiten verschiedener Lastbalancierungsstrukturen zur Verwaltung großer Systeme.

können sie entsprechende Größen wie für lokale Server austauschen. Trotzdem geht durch die Aggregation Information verloren. Beispielsweise ist die Auslastung der ein-

zernen Knoten des Nachbarn oder die genaue Verteilung der Daten auf den dortigen Servern nicht bekannt.

In der Messung wurde die R-Baum-Anwendung eingesetzt, weil in Phasen häufiger kurzer Einfügeoperationen die genaue Berücksichtigung der Datenaffinitäten äußerst wichtig ist, wovon implizit verteilte Balancierung noch eine grobe Übersicht hat, während die explizit verteilte diese Information zwischen Clustern nicht nutzen kann. Abbildung 31 zeigt die Konfiguration und die Laufzeiten, wobei 1510 Einfügeoperationen mit einer Parallelität von 20, gefolgt von 3 parallelen Verbundoperationen ablaufen. Die zentrale Balancierung zeigt das beste Ergebnis, wobei die implizit verteilte Konfiguration mit grobem Wissen über die Datenorte gegenüber der expliziten auch noch recht gut abläuft. Der Vergleich mit einer einfachen zentralen Reihumverteilung zeigt, daß auch explizit dezentrale Lastbalancierung Leistungssteigerung ergibt. Um die Messung übersichtlich zu halten, wurde eine Variante der dezentralen Balancierung verwendet, die nicht nur Anwendungen, sondern auch einzelne Aufträge zwischen Clustern austauscht.

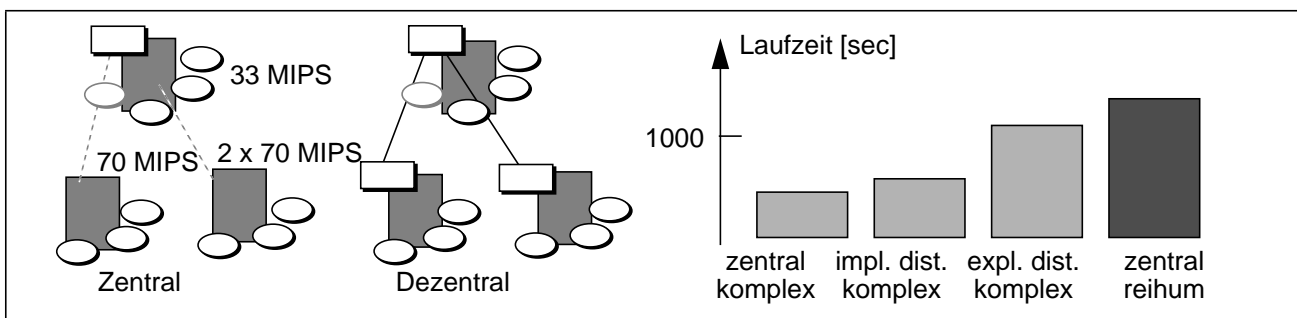


Abbildung 31: Messungen zum Informationsverlust durch dezentrale Balancierung.

5.6 Flexibilität der Lastbalancierung

Unter Flexibilität eines Lastbalancierungsverfahrens versteht man die Breite des Spektrums an verschiedenen Systemen, Lastmustern und Lastsituationen, die das Verfahren effektiv kontrollieren kann, sowie die Fähigkeit, sich geeignet auf ändernde Situationen anzupassen. Die Flexibilität kann auch als Allgemeingültigkeit und automatische Anpassungsfähigkeit des Verfahrens betrachtet werden. In den vorigen Abschnitten wurde das *HiCon*-Konzept bereits verschiedentlich unter Last durch unterschiedliche Anwendungstypen auf verschiedenen Systemstrukturen untersucht. Die Anwendungstypen unterscheiden sich deutlich im Auftragsgranulat, in der möglichen Parallelität, in der Regelmäßigkeit der Ablaufstrukturen und Synchronisationsmuster sowie in der Sensitivität gegenüber Datenaffinitäten.

In diesem Abschnitt sollen noch weitere Aspekte evaluiert werden: Das Potential durch explizite Berücksichtigung von abhängigen Aufträgen innerhalb kleiner Gruppen und das Potential automatischer adaptiver Justierung von Entscheidungsparametern.

5.6.1 Berücksichtigung von Auftragsabhängigkeiten

Das Verhalten der Datenbankanwendung soll durch Vergleichsmessungen mit unterschiedlichen Verfahren zur Berücksichtigung von Reihenfolgebeziehungen in Auftragsgruppen betrachtet werden. Die Messungen wurden auf einem kleinen Cluster aus drei heterogenen Workstations (im Bild $P_1 \dots P_3$) durchgeführt. Lastbalancierung muß daher sowohl die unterschiedlichen Prozessorleistungen berücksichtigen, als auch mit der schwierigen Situation fertig werden, daß mehr ausführbare Aufträge als Knoten vorhanden sind. Abbildung 32 vergleicht die Laufzeiten des in Abschnitt 5.2.3 gezeigten Beispielszenarios unter verschiedenen Varianten zur Balancierung. Die Breite der Auftragsrechtecke ist proportional zur Ausführungszeit der Aufträge, die in der Größenordnung von 3 Sekunden bis zu 2 Minuten liegen. Die Basisrelationen wurden in 14, 21 bzw. 28 Partitionen unterteilt und werden zu Beginn mit 1000 Datensätzen pro Partition geladen. Eine nichttriviale Parallelisierung (Abbildung 23) wurde ausgewählt um beurteilen zu können, inwiefern die Fähigkeit des Lastbalancierungskonzepts, dynamisch Reihenfolgebeziehungen zwischen Aufträgen zu berücksichtigen, auf allgemeine Auftragsstrukturen anwendbar ist. Die Meßergebnisse zeigen, daß Ignorieren der Reihenfolgebeziehungen oder die alleinige Gewichtung der Aufträge nach ihrer Größe ebenso unzureichend ist wie die Gewichtung nach Anzahl der Nachfolger. Die Gewichtung gemäß der kritischen Pfadlängen (*exit paths*) zeigt signifikant kürzere Laufzeiten.

Abbildung 33 zeigt schließlich das Verhalten der Strategien im Mehrbenutzerbetrieb, wobei der Einfachheit halber die Beispielanfrage dreifach konkurrierend, auf disjunkten Relationen arbeitend, ins System gespeist wurde. Der Laufzeitvergleich zeigt, daß die Beachtung der Abhängigkeiten im Mehrbenutzerbetrieb noch deutlichere Vorteile bringt. Während nämlich die Ressourcenbedürfnisse einer einzelnen Anwendung grob Ebene für Ebene anfallen und somit implizit die Vorteile des *Highest Level First* Scheduling nutzen, werden bei konkurrierenden Anwendungen - ohne Berücksichtigung von Abhängigkeiten bzw. ohne Berücksichtigung verschiedener Auftragsgraphen im Zusammenhang - oft Aufträge in Zeitscheiben gelegt, die andere Anwendungen für dringende Aufträge benötigt hätten.

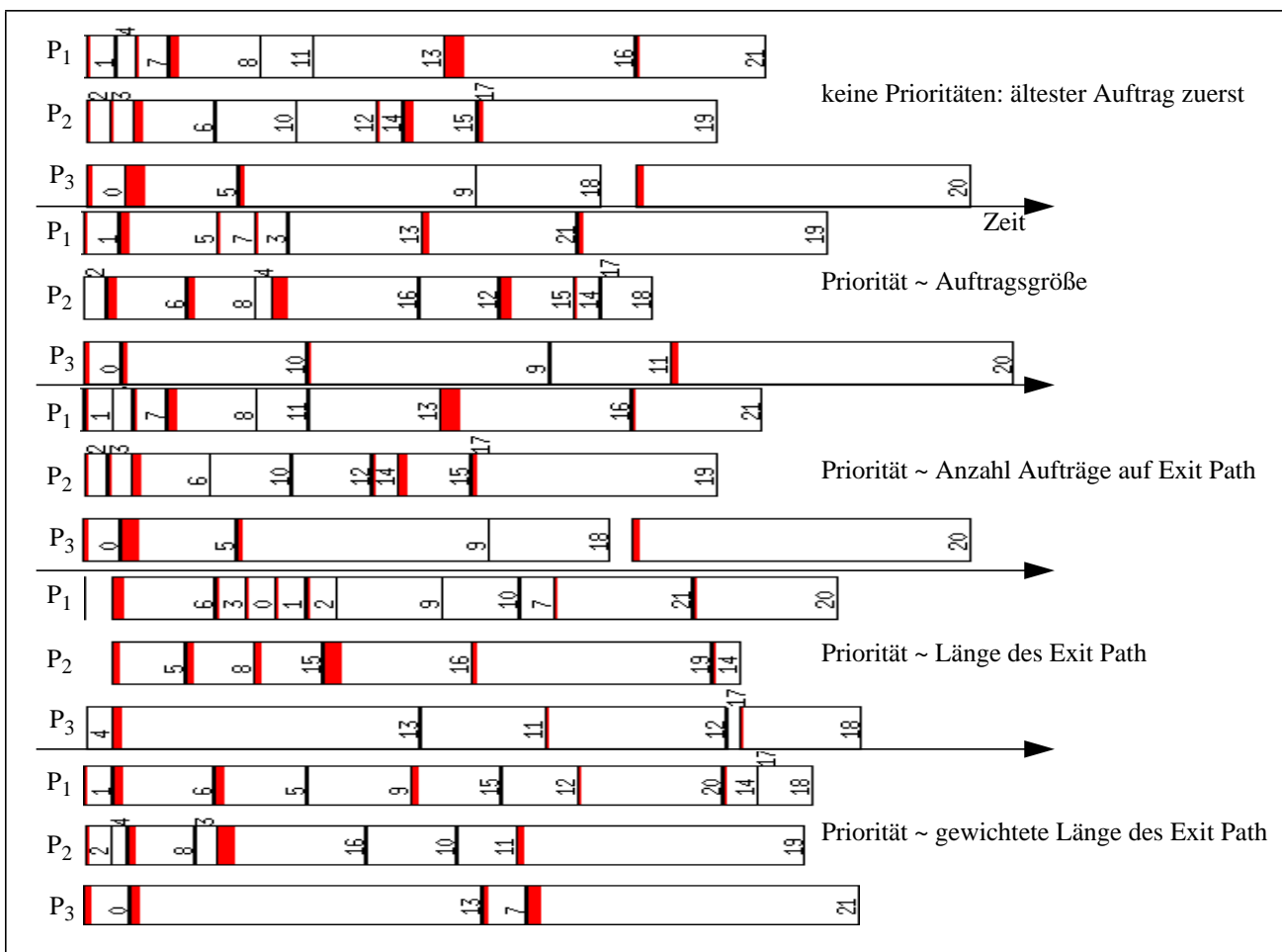


Abbildung 32: Gantt-Diagramm der Ausführung bei verschiedenen Verfahren zur Einplanung von Auftragsgruppen

5.6.2 Automatische Anpassung von Entscheidungsparametern

Die in Abschnitt 3.7.5 vorgestellten Ansätze zur adaptiven Balancierung sollen durch je eine Meßreihe kurz bewertet werden. Die *HiCon*-Lastbalancierung ist durch diese Ansätze für ein breiteres Spektrum von Lastmustern und Lastsituationen anwendbar.

1. Adaptive Regelung des Schwellwerts zur Anwendungsverschiebung zwischen Clustern

Zur Bewertung wurde ein Netzwerk aus vier heterogenen Clustern in Stuttgart, Bonn, Toulouse und Belfast mit je vier Workstations betrachtet. Durch die langsamen Weitverkehrsverbindungen kommen die Datentransferkosten bei Anwendungsverschiebungen deutlich zum tragen. Sieben Bilderkennungsanwendungen wurden im Abstand weniger Sekunden in einem Cluster gestartet, um deutliches Lastungleichgewicht zu erzeugen, bei dem sich geeignete Verschiebungen lohnen. Abbildung 34 vergleicht die Laufzeiten verschiedener Schwellwerte und

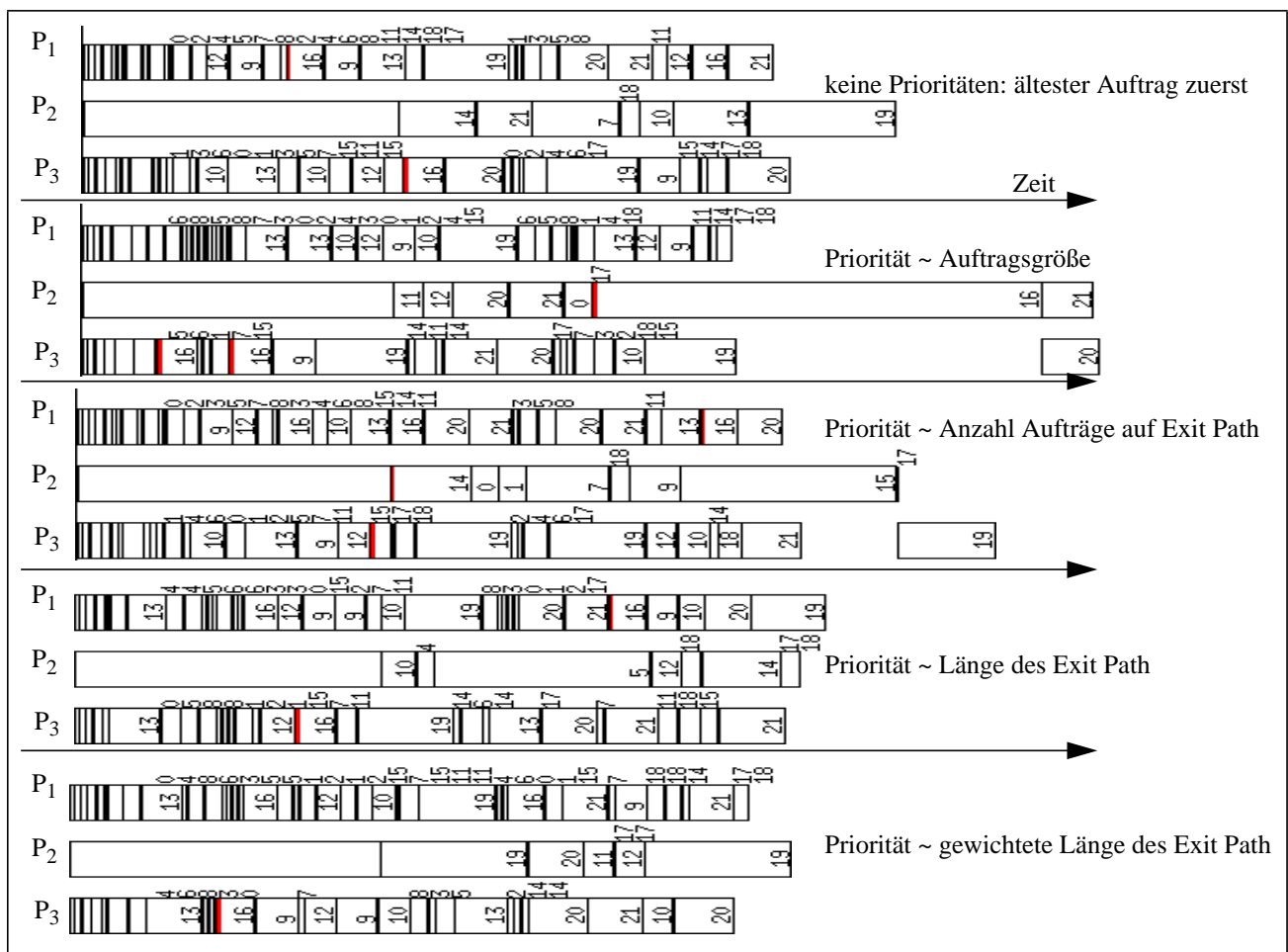


Abbildung 33: Gantt-Diagramm der Ausführung bei verschiedenen Verfahren zur Einplanung von Auftragsgruppen im Mehrbenutzerbetrieb.

Anwendungsauswahl-Strategien der dezentralen Balancierung. Der erste nicht-adaptive Ansatz (1) mit Lastdifferenzschwellwert verschiebt jeweils eine beliebige Anwendung, während der zweite Ansatz (2) die bestgeeignete ermittelt. Der adaptive Ansatz (3) schränkt diese Auswahl auf die Anwendungen ein, deren Verschiebung sich trotz der Datentransferkosten noch lohnen könnte. In diesem Szenario wird der Zeitverlust durch zu häufige, ungeeignete Verschiebungen zwischen Clustern, trotz der Beobachtung sehr weniger, großer Anwendungen, deutlich.

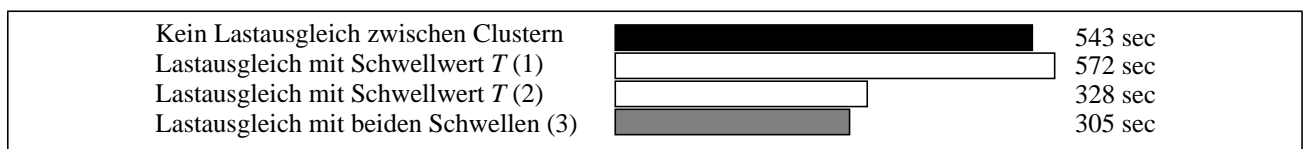


Abbildung 34: Effekt des Datenkommunikations-sensitiven Verschiebungsschwellwerts für Anwendungen.

2. Automatische Vermeidung von Überlastungen der Balancierungskomponente

Hier wird ein Cluster mit 11 heterogenen Workstations und 2 Servern je Workstation betrachtet, das durch 15 parallele Bilderkennungs-Anwendungen mit unterschiedlichen Problemgrößen belastet wird. Während zwei Anwendungen sofort starten und der Balancierung mäßigen Aufwand bescheren, folgen nach 50 Sekunden die anderen 13 Anwendungen in einem Zeitbereich von 5 Sekunden, und generieren große Mengen kleiner paralleler Aufträge, die wegen ihrer Datenabhängigkeiten sorgfältig verteilt werden müssen, und so die komplexe zentrale Balancierung stark überlasten können. Abbildung 35 zeigt links den Balancierungsaufwand über ein Zeitintervall ohne Einsatz der Adaptionstechnik: Die Warteschlange der zu bearbeitenden Ereignisse wächst rapide, was starke Verzögerungen bei der Auftragsverteilung und dadurch auch Leerlaufzeiten bei den Servern verursacht. Die Lastbalancierung konsumiert die gesamte Rechenzeit, die sie auf dem Knoten erhalten kann, anstatt die Rechenzeit den Servern zur Verfügung zu lassen. Auf der rechten Seite ist entsprechend der Balancierungsaufwand bei eingeschalteter Adaption gezeigt, wo die Lastbalancierung ihre Überlastung erkennt und erfolgreich bekämpft. Sie konsumiert immer noch erhebliche Prozessorrechenzeit, die aber durch den Gewinn gerechtfertigt ist.

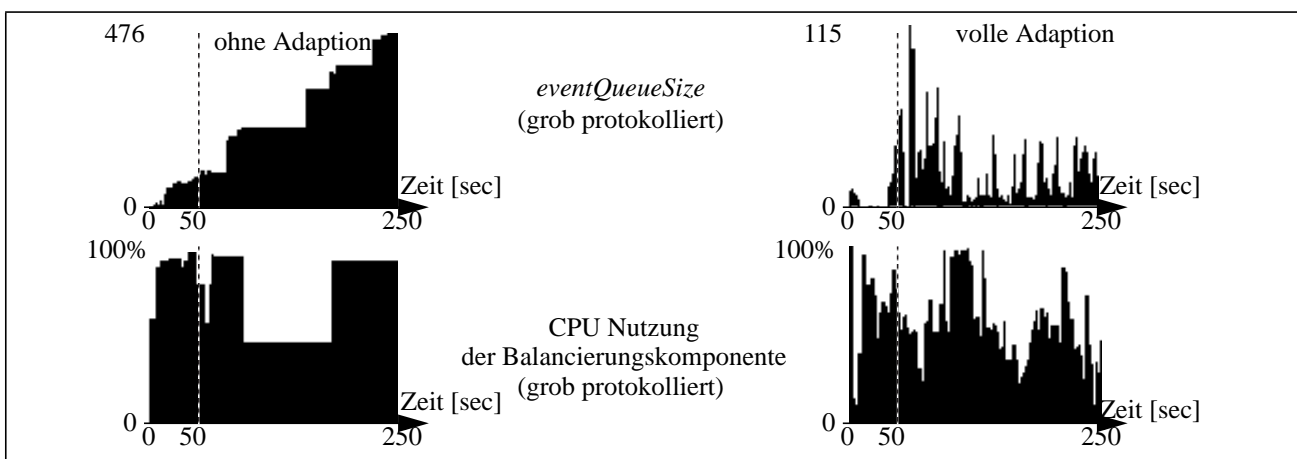


Abbildung 35: Überlastung einer zentralen Balancierungskomponente.

Abbildung 36 vergleicht die Gesamtausführungszeiten bei verschiedenen Techniken. Lastbalancierung ohne Überlastschutz wirkt bei Überlastung verheerend, eine generelle Erhöhung von $t_{advance}$ um den Faktor 100 kann das Problem noch nicht beheben. Erst durch Umschaltung des Entscheidungsalgorithmus bei einem Schwellwert von $eventQueueSize \geq 40$ oder gar schon bei 4 kann die Verstopfung erfolgreich verhindern. Es wurde jeweils bei $eventQueueSize < 2$ zurückgeschaltet, wodurch die Balancierung hier etwa die halbe Zeit die simple Strategie benutzte und alle 3..15 Sekunden wechselte.

3. Adaptive Justierung von Auftragsgrößen-Vorabschätzungen

Keine Adaption: durchweg komplexe Strategie		12000 sec
Fixe Erhöhung des Zuweisungsintervalls, durchweg komplexe Strategie		10800 sec
Volle Adaption, Umschaltsschwelle 40		1065 sec
Volle Adaption, Umschaltsschwelle 4		678 sec

Abbildung 36: Vergleich von Ausführungszeiten in Überlastsituationen.

Um diese dynamische Justierung zu evaluieren, wurden zwei überlappend parallele Flächenerkennungsanwendungen beobachtet, die durch eine zentrale Lastbalancierung auf vier heterogenen Workstations zu verteilen waren. Für jede der vier Anwendungsphasen (unterschiedliche Auftragsstypen) wird ein separater Faktor $computeTimeAdapt_{cs}$ verwaltet. Abbildung 37 zeigt die charakteristische Minderung der Abweichung zwischen den Vorabschätzungen der Clients und den real beobachteten Rechenzeiten für zwei Zeitintervalle auf logarithmischer Skalierung. Insgesamt konnte das Adaptionsverfahren die Abschätzungsfehler um Faktoren bis zu 1000 reduzieren. Zu Beginn der Split-Aufträge wurden die Auftragsgrößen um den Faktor 100 unterschätzt, während sie später stark überschätzt wurden. Das führte zu zwischenzeitlichen Verschlechterungen der Vorabschätzungen durch Adaption, bis der Korrekturfaktor wieder eingeregelt war. Derlei Effekte sind in dieser Anwendung abhängig von den Bildstrukturen, was eine dynamische Nachregelung der Korrekturfaktoren unerlässlich macht. Die Adaptionstechnik konnte in diesem Szenario die Laufzeiten um 8% von 333 auf 276 sec reduzieren. In einigen Anwendungen werden ganze Schübe von Aufträgen generiert (typisch etwa für parallelisierte Schleifen), was der Adaption keine Gelegenheit zur direkten Nachregelung gibt, weil das Feedback fehlt, so daß die Adaption erst bei nachfolgenden Iterationen wirksam wird.

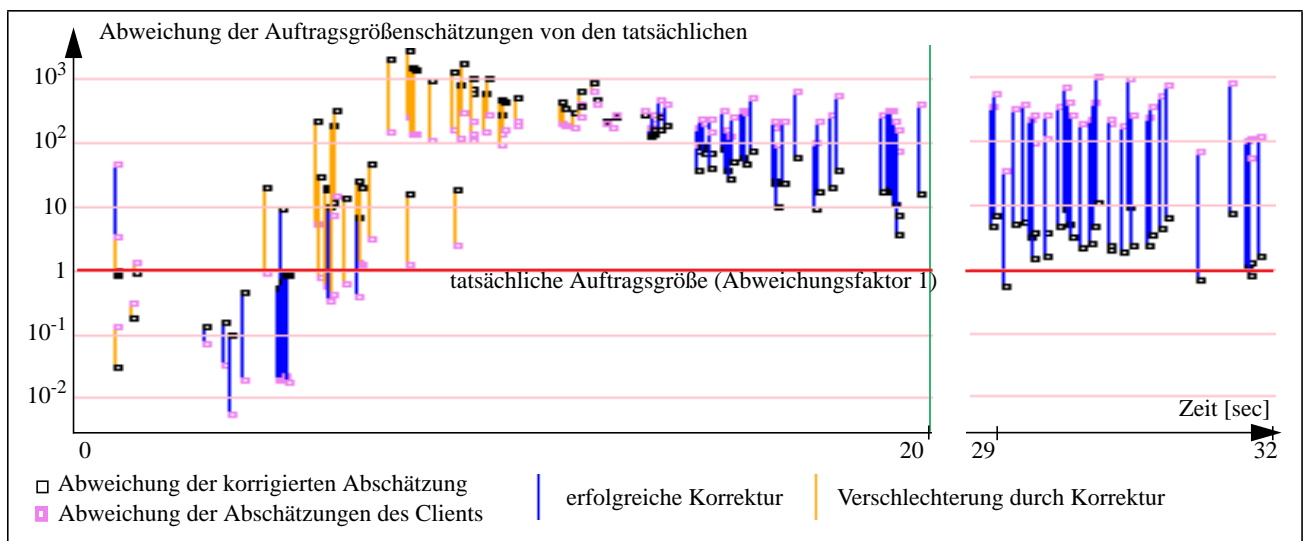


Abbildung 37: Adaptive Korrektur der Auftragsgrößen-Vorabschätzungen.

4. Adaptive Bestimmung von Datenkommunikationskosten

Zur Bewertung der Adaptionstechnik wurde ein dem obigen ähnliches Szenario verwendet, wobei zuerst eine Flächenerkennung auf ein großes, aufwendiges Bild, und nach 400 sec zusätzlich die Erkennung eines kleineren Bildes gestartet wurde. Für die Vergleichsmessung ohne Adaption wurden die bestmöglichen statischen Kostenabschätzungen verwendet, die durch einen vorhergehenden Lauf mit Adaption ermittelt wurden. Adaption konnte die Gesamtlaufzeit um 9% von 950 auf 860 sec hauptsächlich dadurch reduzieren, daß sie in Situationen mit stark konkurrierenden Datenzugriffen bzw. hoher Netzbelastung (Quad-Split und -Merge Phasen) die erhöhten Kosten erkannte und infolgedessen die Parallelität einschränkte und Server mit guter Datenaffinität trotz eventueller Prozessorleistungs- / Belastungs-Nachteile stärker bevorzugte. Abbildung 38 zeigt die zeitliche Nachregelung der Datenzugriffskosten für zwei wesentliche Datenstrukturen der Anwendung. Hier ist weiterhin erkennbar, daß die Datenaustauschkosten im Mittel während der zweiten, kleineren Anwendung geringer waren. Der starke Anstieg gegen Ende rührt vom Zusammenreffen der Boundary-Trace Phasen beider Anwendungen, die hohe Zugriffsparallelität und Netzlast erzeugen. Beim Zusammentreffen der Quad-Phasen zweier Anwendungen in einem ähnlichen Szenario wurde ein Ansteigen der Datenzugriffskosten um den Faktor 1.5 gegenüber den hier abgebildeten beobachtet, was die Notwendigkeit dynamischer Regelung dieser Kostenschätzung unterstreicht.

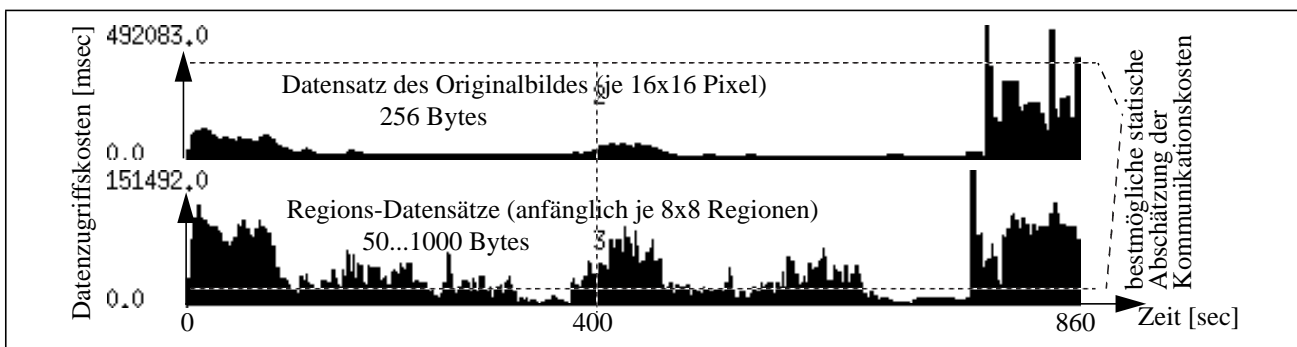


Abbildung 38: Dynamische Regelung der Datenzugriffskostenschätzungen.

Für die Beobachtung des Lese-/Schreibverhältnisses von Datentypen zur künstlichen Abwertung der Kosten für das Anlegen von Kopien wird keine separate Messung vorgestellt. Die Notwendigkeit wird bei Betrachtung von Anwendungen wie z.B. der Wegesuche, die massiv auf große Datenmengen lesend zugreifen, offensichtlich: Ohne die adaptive Abwertung werden solche Anwendungen weitgehend sequentiell abgewickelt, da sich für jeden einzelnen Auftrag das Verschicken der Kopien nicht lohnt.

5. Adaptive Korrektur der Vorabschätzungen des Datenkommunikationsaufwands von Aufträgen

Wiederum diene das obige Szenario zur Bewertung dieses Regelungskonzepts. Abbildung 39 zeigt, wie die Vorabschätzungen in der Anfangsphase durch die gere-

gelten Korrekturfaktoren verbessert werden konnten. Die Abweichungen der Vorabschätzungen wurden im Mittel um den Faktor 20 verringert. Einige Ausnahmen, wo durch falsche Datenreferenzangaben Fehleinschätzungen um bis 10^6 vorkommen, können nicht vollständig kompensiert werden, die Regelung bleibt jedoch stabil. In den anderen Phasen der Bilderkennung wurden Verbesserungen um den Faktor 100 erreicht.

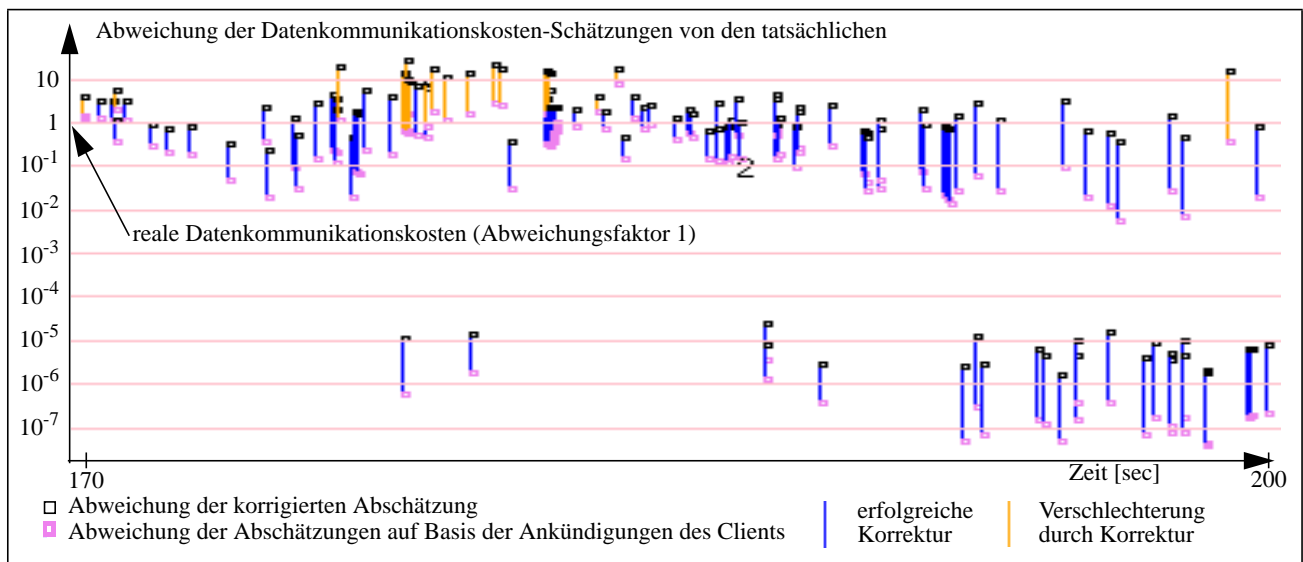


Abbildung 39: Adaptive Korrektur der Datenkommunikationskosten für Aufträge.

6. Adaptive Einschätzung der CPU-Last durch Auftragsausführungen und Lastbalancierungsaufwand

Die adaptive Prozessorlastermittlung wurde anhand eines Mehrbenutzer-Szenarios von vier Flächenerkennungen unterschiedlicher Problemgröße auf einem heterogenen Cluster von vier Workstations mit je drei Servern evaluiert. Abbildung 40 zeigt die geschätzten Prozessorauslastungen und die vom Betriebssystem erhaltene CPU *run queue length* für drei der Prozessoren. Links wurde die Last mithilfe der Auftragstyp-spezifischen, langfristig anhand der CPU *run queue length* nachgeregelten Lastfaktoren sowie der gemessenen Balancierungslast (P4) abgeschätzt, rechts wurde angenommen, daß jeder aktive Server seinen Prozessor voll nutzen möchte. Man sieht, daß die Quad-Split-Aufträge (hellgrau) etwa 30% CPU-Nutzung erzeugten, weil sie stark kommunizieren. Ohne Adaption wurde daher die Belastung der Knoten stark überschätzt (siehe etwa P3). Die CPU *run queue length* kann, wie in Abschnitt 3.7.5 erläutert, wegen der groben Auflösung, der trotz Glättung verbleibenden Lastspitzen, der fehlenden Korrelation zu Auftragsbeginn und -Ende und der fehlenden Möglichkeit zur sinnvollen Extrapolation nicht direkt verwendet werden.

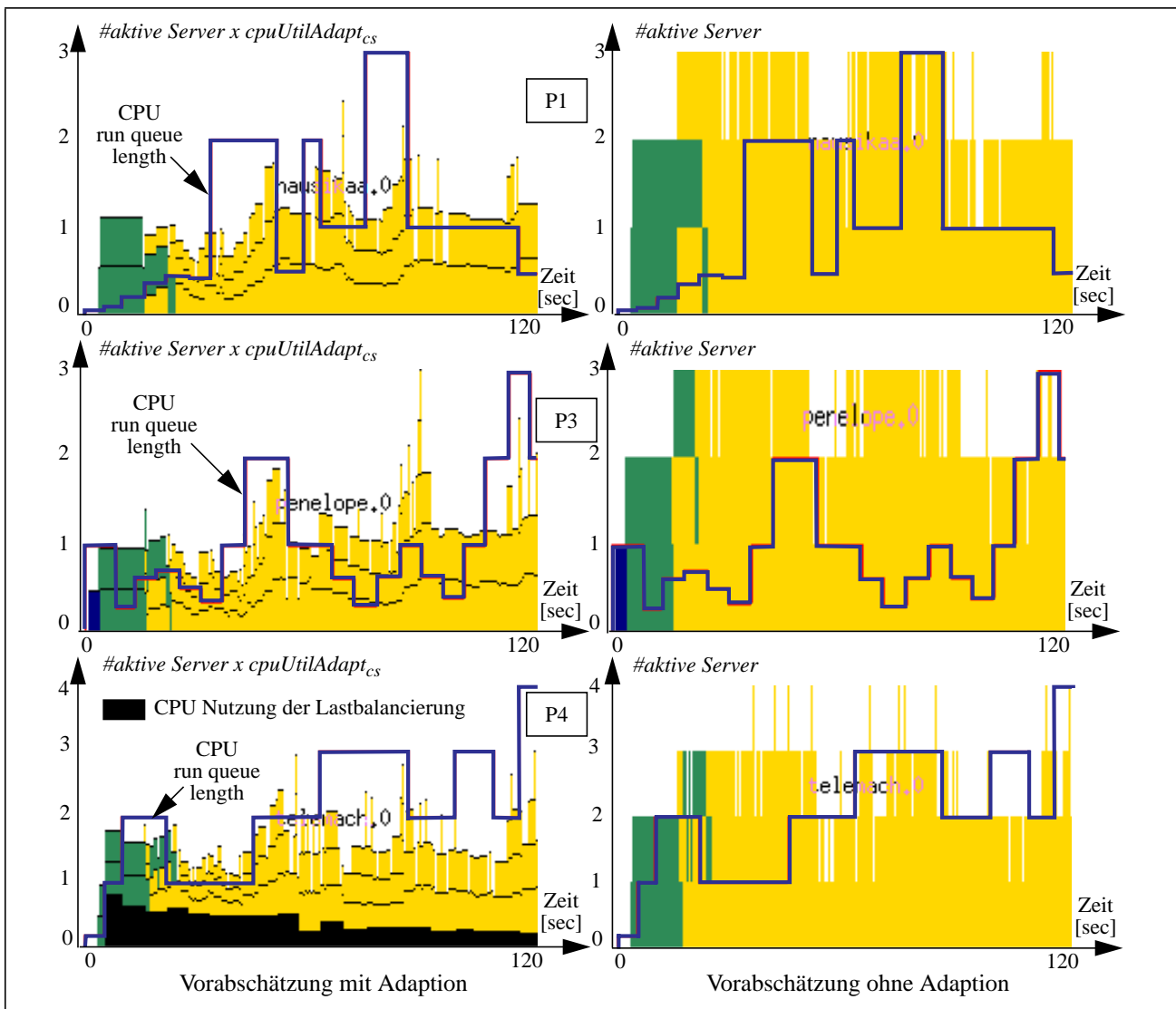


Abbildung 40: Adaptive Abschätzung der Prozessorbelastungen.

6 Zusammenfassung der Ergebnisse

6.1 Zusammenfassung

In dieser Arbeit wurden die Grundkonzepte dynamischer Lastbalancierung eingeführt. Es wurde ein Konzept zur dynamischen Lastbalancierung großer paralleler, konkurrierender Anwendungen auf lose gekoppelten parallelen und verteilten Systemen vorgestellt. Workstation-Cluster sind eine derzeit zukunftssträchtige Systemarchitektur für viele Bereiche in Wirtschaft und Wissenschaft. Das Konzept wurde prototypisch implementiert und für ein breites Anwendungsspektrum anhand realer Messungen validiert. Neben neuen und weiterentwickelten Balancierungstechniken, wie komplexer zentraler Balancierung für Cluster und dezentralem Ausgleich zwischen Clustern, der Berücksichtigung von Datenaffinitäten und Auslegung für große, parallele Anwendungen im Mehrbenutzerbetrieb, erbrachte die Arbeit folgende allgemeinen Ergebnisse, die für paralleles Rechnen und automatische Lastbalancierung interessant sind:

- Im allgemeinen kann anwendungsunabhängige dynamische Lastbalancierung in Situationen mit bereits recht guter, zufälliger Lastverteilung etwa 5 bis 10% Durchsatzsteigerung erzielen. Lastbalancierung verhindert aber katastrophale Verteilungen und kann dabei den Durchsatz um Größenordnungen steigern. Sie erhöht weiterhin die Portabilität und Flexibilität verteilter und paralleler Anwendungen, und reduziert den hohen Aufwand, um in parallelen und verteilten Systemen akzeptablen Durchsatz zu erzielen. Daher sollte das Hauptaugenmerk bei der Entwicklung von Balancierungskonzepten auf breiter Anwendbarkeit und hoher Flexibilität bezüglich der möglichen Lastprofile im System liegen.
- Der Forderung nach grenzenloser Skalierbarkeit dynamischer Lastbalancierungsverfahren wird meist durch völlig dezentrale Ansätze Rechnung getragen. Dabei werden die großen Vorteile zentraler Balancierungsverfahren nicht beachtet, die besonders in realen, komplexen Systemen und Lastprofilen deutlich werden. Hier können Entwicklungen aus dem Bereich der statischen Lastbalancierung und des Transaktions-Routing in Datenbanksystemen einbezogen werden.
- Für paralleles und verteiltes Hochleistungsrechnen ist hohe Netzwerkleistung absolut notwendig, erhöht aber nicht primär die Gesamtrechenleistung oder den Durchsatz von Anwendungen. Sie ermöglicht nur ein feineres Granulat an Parallelität innerhalb von Anwendungen, mehr Datenkommunikation und Zugriffe auf entfernte Daten sowie eine feinere Lastverteilung im System. Erst wenn Anwendungen und dynamische Lastbalancierung das optimale Verhältnis zwischen voller Nutzung der Rechenkapazitäten und dem entstehenden Kommunikationsaufwand erkennen, kann hohe Netzwerkleistung durch Anpassung der Parallelität, des Auftrags- und Datengranulats sowie der Genauigkeit des Lastausgleichs den Systemdurchsatz erhöhen.

Dies ist bisher nur für einfache, einzelne Anwendungen möglich. Da feingranulare Abläufe häufigere, kurze Kommunikationsvorgänge hervorrufen, wird dabei neben der Netzbandbreite zunehmend die Latenzzeit ein kritischer Faktor.

- Bei der Parallelisierung großer Anwendungen wird zunehmend ein portables und flexibles Ablauf- und Kooperationskonzept wichtig. Der bisher häufigste Ansatz, kommunizierende Prozesse statisch auf Knoten zu verteilen, ist sehr effizient, während Client - Server Strukturen bei feinem Auftragsgranulat und stark gekoppelten parallelen Abläufen üblicherweise mehr Zusatzaufwand verursachen. Sie verlangen größere Dekomposition und Vermeidung unnötiger Datenkommunikation, sind jedoch deutlich flexibler, um in heterogenen parallelen und verteilten Systemen im Mehrbenutzerbetrieb effizient ablaufen zu können.
- Die dynamische Berücksichtigung von Abhängigkeiten zwischen Aufträgen hat sich als sinnvoll erwiesen. Die Erfahrung zeigte allerdings, daß die sinnvolle Zusammenstellung von Auftragsgruppen durch die Clients nur in wenigen Anwendungsklassen naheliegend ist. Daher wird im *HiCon*-Modell nur noch eine vereinfachte Planung eingesetzt, bei der Clients für einzelne Aufträge kritische Folgepfade und nachfolgende Parallelität explizit angeben können. Die Lastbalancierung benötigt dann kein Auftragsgruppenkonzept, und für viele Anwendungen ist es einfacher, Folgepfadlängen von Aufträgen abzuschätzen, als explizit Gruppen anzugeben.
- Lastbalancierung für komplexere Anwendungen benötigt ein einfaches, flexibles Modell zur Erfassung und Berücksichtigung von Datenkommunikation im System. Das in dieser Arbeit vorgeschlagene Konzept kann als mögliches, im betrachteten Anwendungs- und Systembereich erfolgreiches, Beispiel dafür angesehen werden.

6.2 Ausblick

Im vorgestellten Lastbalancierungskonzept mußten einige Aspekte unberücksichtigt bleiben bzw. konnten nicht detaillierter untersucht und entwickelt werden. Sie bieten Ansätze, um das Konzept weiter zu flexibilisieren, so daß die Lastbalancierung mehr Optimierungspotential und stabileres Verhalten erlangt und auch mit weiteren schwierigen Situationen zurecht kommt. Außerdem blieben Optimierungen im Konzept des Laufzeitsystems bisher ungenutzt.

- Der dezentrale Lastausgleich zwischen Clustern wurde nur sehr grob entwickelt, da er nicht Hauptgegenstand des Projekts war. Die dezentrale Balancierung konnte nicht für sehr große Systeme erprobt werden. Im dezentralen Bereich existieren zahlreiche Veröffentlichungen, die zunehmend auch eine zweistufige Struktur - intra Cluster und inter Cluster - aufweisen. Der *HiCon*-Ansatz könnte beispielsweise gemäß dem Gradientenverfahren Ziel-Cluster ermitteln, oder gemäß einer Erweiterung dessen die Schwelle zur Verschiebung von Anwendungen einstellen: Wenn ein Nachbar sehr hohe Last hat, gibt er bereits Last an andere ab, auch wenn noch nicht

höher belastet ist, weil vom hochbelasteten Cluster Aufträge zu erwarten sind. Das beschleunigt die Lastverteilung bei großen Ungleichgewichten. Datenkommunikation ist dagegen in dezentralen Strukturen schwer zu berücksichtigen und wurde bisher meist ignoriert. Zur Verbesserung der groben Lastbalancierung zwischen Clustern sind genauere Abschätzungen nötig, wieviel Last Anwendungen wirklich erzeugen, wie leistungsfähig und ausgelastet die Cluster wirklich sind, um abschätzen zu können, ob eine Anwendung im Nachbar-Cluster wirklich schneller abläuft.

- Die tatsächliche Migration von Clients zwischen Clustern könnte vermeiden, daß der Auftrags- und Ergebnisstrom verschobener Anwendungen dauernd über die Clustergrenzen laufen müssen. Das verlangt keine konzeptuelle Änderung sondern ist lediglich ein Implementierungsproblem, da Prozeßmigration oder *Checkpointing* auf Anwendungsebene notwendig wird.
- Eine automatische dynamische Anpassung der dezentralen Balancierungsstruktur ist denkbar, wobei entweder Knoten an andere Cluster abgegeben werden oder neue Cluster kreiert bzw. wieder zusammengefaßt werden können. Diese Flexibilisierung verlangt keine Abänderung, aber eine Erweiterung des Balancierungskonzepts um die Entscheidung, wann und wie die Cluster-Struktur verbessert werden kann; Meist soll eine längerfristige Überlastung der zentralen Balancierungskomponente eines Clusters behoben werden, oder auf eine Änderung der Topologie reagiert werden. Weiterhin könnte die Anzahl der bereitstehenden Server pro Knoten dynamisch variiert werden. Dadurch könnte Lastbalancierung einerseits höhere Parallelität ermöglichen und andererseits Hauptspeicherbedarf und Verwaltungsaufwand einsparen.
- Die Systemkomponente zur Verwaltung der globalen Daten im System könnte auf verschiedene Weisen effizienter bzw. flexibler gestaltet werden, was jedoch geringe Auswirkungen auf das Lastmodell der Balancierungsstrategie hat. Optimierungsmöglichkeiten bestehen etwa in der Begrenzung der Anzahl gleichzeitig im System existierender Kopien pro Datensatz, um in kritischen Fällen ein Überlauf des Haupt- oder Sekundärspeichers zu vermeiden und um den Invalidierungsaufwand bei exklusiven Datenzugriffen gering zu halten. Die ideale Anzahl an Kopien könnte durch die Lastbalancierung anhand der Speicherkapazitäten und der Änderungshäufigkeiten geregelt werden. Alternativ zum Konzept der Kopien-Invalidierung könnte auch das Konzept der Aktualisierung aller Kopien nach Änderungsoperationen verwendet werden. Eine weitere Optimierungsmöglichkeit besteht darin, daß Anwendungen asynchron Daten vorab anfordern, während sie noch andere Dinge rechnen können. Server innerhalb eines Knotens sollten real gemeinsamen Speicher für Zugriff auf globale Daten nutzen können, um Nachrichten und Datenkopien einzusparen. Dadurch kann *Multitasking* auf den Knoten auch innerhalb einer parallelen Anwendung effizienter genutzt werden. Dazu wurde eine Variante des *HiCon*-Prototyps auf Basis leichtgewichtiger Prozesse (Threads) entwickelt, die jedoch in dieser Arbeit noch nicht genauer vorgestellt werden kann.

- Der entwickelte und prototypisch realisierte Ansatz ist nicht geeignet, um direkt in kommerzielle Software integriert zu werden. Neben der oben erwähnten Aspekte, die weiterer Entwicklung bedürfen, enthält das Ausführungs- und Datenmodell für die Praxis zu starke Einschränkungen, die für die Forschungsarbeiten ein kompaktes Modell und eine effiziente Realisierung ermöglichen, aber für viele existierenden Anwendungen und Basissysteme gravierende Umstellungen verlangen würden. Die entwickelten Konzepte können auszugsweise innerhalb von Betriebssystemen, Transaction Processing Monitoren und Datenbanksystemen integriert werden. Vom Modell und der Umsetzbarkeit her am nächsten stehen jedoch neuere Umgebungen wie PVM [PVM93] oder MPI [MPI94], die Plattform-unabhängig Primitiven zur Prozeßverwaltung, Kommunikation und zur Verwaltung und Nutzung von Diensten anbieten. Diese verwenden ebenfalls einfache, restriktive Ablaufmodelle, die dem Lastbalancierungsdienst Plattform-unabhängig mehr Informationen und bessere Einwirkungsmöglichkeiten geben (Abschnitt 4.2).

7 Literaturverzeichnis

- [Ahma94] I. Ahmad, A. Ghafoor, G. Fox, *Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers*, Journal of Parallel and Distributed Computing 20, 1994
- [Ange90] F. Anger, J. Hwang, Y. Chow, *Scheduling with Sufficient Loosely Coupled Processors*, Journal of Parallel and Distributed Computing 9, 1990
- [Bara85] A. Barak, A. Shiloh, *A Distributed Load-balancing Policy for a Multicomputer*, Software-Practice and Experience Vol. 15 No. 9, 1985
- [Baum88] K. Baumgartner, B. Wah, *A Global Load Balancing Strategy for a Distributed Computer System*, Workshop on the Future Trends of Distributed Computing Systems in the 1990's, 1988
- [Beck92] W. Becker, *Lastbalancierung in heterogenen Client-Server Architekturen*, Fakultätsbericht 1992-1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992
- [Beck93] W. Becker, *Globale dynamische Lastbalancierung in datenintensiven Anwendungen*, Fakultätsbericht 1993-1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1993
- [Beck94] W. Becker, *Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen*, Fakultätsbericht 1994-4, Universität Stuttgart, Institut für parallele und verteilte Höchstleistungsrechner, 1994
- [Beck94a] W. Becker, R. Pollak, *Efficiency of Server Task Queueing for Dynamic Load Balancing*, Fakultätsbericht 1994-9, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1994
- [Beck94b] W. Becker, G. Waldmann, *Exploiting Inter Task Dependencies for Dynamic Load Balancing*, Proc. IEEE 3rd Int. Symp. on High-Performance Distributed Computing (HPDC), San Francisco, 1994
- [Beck94c] W. Becker, J. Zedelmayer, *Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures*, Mitteilungen GI, Parallele Algorithmen und Rechnerstrukturen, GI/ITG Workshop Potsdam, 1994
- [Beck95] W. Becker, *Lastverteilung in Workstation-Netzen*, BI Sonderheft Paralleles Rechnen, RUS, Universität Stuttgart, 1995
- [Beck95a] W. Becker, *Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen*, Informatik Forschung und Entwicklung Vol. 10 No. 1, Springer Verlag, 1995
- [Beck95b] W. Becker, G. Waldmann, *Adaption in Dynamic Load Balancing: Potential and Techniques*, Tagungsband 3. Fachtagung Arbeitsplatz-Rechensysteme (APS), 1995
- [Beck95c] W. Becker, *Dynamic Balancing Complex Workload in Workstation Networks - Challenge, Concepts and Experiences*, Proc. Int. Conf. High Performance Computing and Networking (HPCN) Europe, LNCS, Springer Verlag, 1995
- [Bemm90] T. Bemmerl, A. Bode, O. Hansen, T. Ludwig, *A Testbed for Dynamic Load Balancing on Distributed Memory Multiprocessors*, Working Paper ESPRIT Projekt PUMA 2701, 1990

- [Bern93] G. Bernard, D. Steve, M. Simatic, *A survey of load sharing in networks of workstations*, Distributed Systems Engineering Vol.1 No.2, 1993
- [Blaz86] J. Blazewicz, M. Drabowski, J. Weglarz, *Scheduling Multiprozessor Tasks to Minimize Schedule Length*, IEEE Transactions on Computers Vol. 35 No. 5, 1986
- [Bogl92] Y. Boglaev, *Exact dynamic load balancing of MIMD architectures with linear programming algorithms*, Parallel Computing 18, 1992
- [Bokh81] S. Bokhari, *A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System*, IEEE Transactions on Software Engineering Vol. 7 No. 6, 1981
- [Bono88] F. Bonomi, A. Kumar, *Adaptive Optimal Load Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler*, Proceedings Distributed Computing Systems, 1988
- [Bono90] F. Bonomi, A. Kumar, *Adaptive Load Balancing in a Nonhomogeneous Multiserver System with a Central Job Scheduler*, IEEE Transactions on Computers Vol. 39 No. 10, 1990
- [Borr90] A. Borr, *Guardian 90: A Distributed Operating System Optimized Simultaneously for High-Performance OLTP, Parallelized Batch/Query and Mixed Workloads*, Tandem Technical Report 90.8, Cupertino, 1990
- [Bowe88] N. Bowen, C. Nikolaou, A. Ghafoor, *Hierarchical Workload Allocation for Distributed Systems*, Proceedings Parallel Processing, 1988
- [Bowe92] N. Bowen, C. Nikolaou, A. Ghafoor, *On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems*, IEEE Transactions on Computers Vol. 41 No. 3, March 1992
- [Brun85] J. Bruno, *On Scheduling Tasks with Exponential Service Times and In-Tree Precedence Constraints*, Acta Informatica 22, 1985
- [Cap92] C. Cap, V. Strumpfen, *The PARFORM - A High Performance Platform for Parallel Computing in a Distributed Workstation Environment*, Technical Report, Institut für Informatik, Universität Zürich, 1992
- [Casa88] T. Casavant, J. Kuhl, *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Engineering Vol. 14 No. 2, 1988
- [Casa94] J. Casas, R. Konuru, S. Otto, R. Prouty, J. Walpole, *Adaptive Load Migration Systems for PVM*, Proc. Supercomputing '94, 1994
- [Chan75] K. Chandy, P. Reynolds, *Scheduling Partially Ordered Tasks with Probabilistic Execution Times*, Proceedings Operating System Principles, Operating Systems Review Vol. 9 No. 5, 1975
- [Chou82] T. Chou, J. Abraham, *Load Balancing in Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 8 No. 4, 1982
- [Chow79] Y. Chow, W. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System*, IEEE Transactions on Computers Vol. 28 No. 5, 1979
- [Cole93] R. Cole, G. Graefe, *Optimization of Dynamic Query Evaluation Plans*, technical report CU-CS-671-93, University of Colorado at Boulder, 1993
- [Coli91] J. Colin, P. Chretienne, *C.P.M. Scheduling with Small Communication Delays and Task Duplication*, Operations Research Vol. 39 No. 4, 1991

- [Cope88] G. Copeland, W. Alexander, E. Boughter, T. Keller, *Data Placement in Bubba*, Proceedings SIGMOD, 1988
- [Deva89] M. Devarakonda, R. Iyer, *Predictability of Process Resource Usage: A Measurement-Based Study on UNIX*, IEEE Transactions on Software Engineering, Vol. 15 No. 12, 1989
- [Doug91] F. Douglass, J. Ousterhout, *Transparent Process Migration: Design Alternatives and the Sprite Implementation*, Software-Practice and Experience Vol. 21 No. 8, 1991
- [Eage86] D. Eager, E. Lazowska, J. Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering Vol. 12 No. 5, 1986
- [Eage88] D. Eager, E. Lazowska, J. Zahorjan, *The Limited Performance Benefits of Migrating Active Processes for Load Sharing*, ACM SIGMETRICS, Performance Evaluation Review, 1988
- [Efe89] K. Efe, B. Groselj, *Minimizing Control Overheads in Adaptive Load Sharing*, Proceedings 9th International Conference on Distributed Computing Systems, 1989
- [Emc95] P. Huish (Ed.), *European Meta Computing Utilising Integrated Broadband Communications - Interim Report*, Deliverable CEC Project B2010 TEN-IBC E=MC2, 1995
- [Evan94] D. Evans, W. Butt, *Load balancing with network partitioning using host groups*, Parallel Computing 20, 1994
- [Ezza86] A. Ezzat, *Load Balancing in NEST: A Network of Workstations*, Proceedings Fall Joint Computer Conference, Dallas, 1986
- [Ferg88] D. Ferguson, Y. Yemini, C. Nikolaou, *Microeconomic Algorithms for Load Balancing in Distributed Computer Systems*, Proceedings Distributed Computing Systems, 1988
- [Ferr86] D. Ferrari, S. Zhou, *A Load Index for Dynamic Load Balancing*, Proceedings Fall Joint Computer Conference, Dallas, 1986
- [Gopi91] P. Gopinath, R. Gupta, *A Hybrid Approach to Load Balancing in Distributed Systems*, Symposium on Experiences with Distributed and Multiprocessor Systems USENIX, 1991
- [Gosw93] K. Goswami, M. Devarakonda, R. Iyer, *Prediction-Based Dynamic Load-Sharing Heuristics*, IEEE Transactions on Parallel and Distributed Systems Vol. 4 No. 6, 1993
- [Grae93] G. Graefe, W. McKenna, *The Volcano Optimizer Generator: Extensibility and Efficient Search*, Proc. IEEE Conf. on Data Engineering, 1993
- [Graf93] T. Graf, R. Assini, J. Lewis, E. Sharpe, J. Turner, M. Ward, *HP Task Broker: A Tool for Distributing Computational Tasks*, Hewlett Packard Journal August, 1993
- [Grim91] A. Grimshaw, V. Vivas, *FALCON: A Distributed Scheduler for MIMD Architectures*, Symposium on Experiences with Distributed and Multiprocessor Systems, Usenix, 1991
- [Hac86] A. Hac, T. Johnson, *A Study of Dynamic Load Balancing in a Distributed System*, Proc. ACM SIGCOMM Symposium on Communications, Architectures and Protocols, Stowe, Vermont, 1986
- [He89] X. He, *Eine Übersicht über die Lastverteilung in verteilten Systemen*, Bericht 190/89, Universität Kaiserslautern, Fachbereich Informatik, 1989
- [Hsu86] C. Hsu, J. Liu, *Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems*, Proceedings Distributed Computing Systems, 1986

- [IBM95] *IBM LoadLeveler User's Guide*, IBM Corporation, 1995
- [Indu86] B. Indurkha, H. Stone, L. Cheng, *Optimal Partitioning of Randomly Generated Distributed Programs*, IEEE Transactions on Software Engineering Vol. 12 No. 3, 1986
- [Iqba86] M. Iqbal, J. Saltz, S. Bokhari, *A Comparative Analysis of Static and Dynamic Load Balancing Strategies*, Proceedings Parallel Processing, 1986
- [Jian89] Y. Jiang, P. Chaudhuri, G. Gupta, *Dynamic Data Balancing Algorithm for a Shared-nothing Multiprocessor Database System*, Technical Report 89/8, Department of Computer Science, James Cook University of North Queensland, 1989
- [John93] T. Johnson, *A Concurrent Dynamic Task Graph*, Proc. Int. Conf. on Parallel Processing, 1993
- [Kale88] L. Kale, *Comparing the Performance of two Dynamic Load Distribution Methods*, Proceedings Parallel Processing, 1988
- [Kane91] J. Kanet, V. Sridharan, *PROGENITOR: A generic algorithm for production scheduling*, Wirtschaftsinformatik Heft 4, 1991
- [Kim92] C. Kim, H. Kameda, *An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems*, IEEE Transactions on Computers Vol. 41 No. 3, 1992
- [Krem92] O. Kremien, J. Kramer, *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Transactions on Parallel and Distributed Systems Vol. 3 No.6, 1992
- [Kreu89] M. Kreuzmann, M. Müller, *Dynamischer Lastausgleich in verteilten Systemen*, Diplomarbeit, Mathematisch-Naturwissenschaftliche Fakultät der Rheinischen Freidrich-Wilhelms-Universität, Bonn, 1989
- [Krue88] P. Krueger, M. Livny, *A Comparison of Preemptive and Non-Preemptive Load Distributing*, Proceedings Distributed Computing, 1988
- [Kuch90] H. Kuchen, A. Wagener, *Comparison of 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, 1991
- [Lam75] S. Lam, R. Sethi, *Analysis of a Level Algorithm for Preemptive Scheduling*, Proceedings Operating Systems Principles, Operating Systems Review, Vol. 9 No. 5, 1975
- [Lee92] C. Lee, D. Lee, M. Kim, *Optimal Task Assignment in Linear Array Networks*, IEEE Transactions on Computers, Vol. 41 No. 7, 1992
- [Lewi93] T. Lewis, H. El-Rewini, *Parallax: A Tool for Parallel Program Scheduling*, IEEE Parallel and Distributed Technology, 1993
- [Li90] K. Li, K. Cheng, *Static Job Scheduling in Partitionable Mesh Connected Systems*, Journal of Parallel and Distributed Computing No. 10, 1990
- [Lin87] F. Lin, R. Keller, *The Gradient Model Load Balancing Method*, IEEE Transactions on Software Engineering Vol. 13 No. 1, 1987
- [Lin92] H. Lin, C. Raghavendra, *A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC)*, IEEE Transactions on Software Engineering Vol. 18 No. 2, 1992

- [Litz88] M. Litzkow, M. Livny, M. Mutka, *Condor - A Hunter of Idle Workstations*, Proc. 8th Int. Conference on Distributed Computing Systems, San Jose, 1988
- [Lo88] V. Lo, *Heuristic Algorithms for Task Assignment in Distributed Computing Systems*, IEEE Transactions on Computers Vol. 37 No. 11, 1988
- [Lüli91] R. Lüling, B. Monien, F. Ramme, *Load Balancing in Large Networks: A Comparative Study*, IEEE Symposium Parallel and Distributed Processing, Dallas, 1991
- [Ma82] P. Ma, E. Lee, M. Tsuchiya, *A Task Allocation Model for Distributed Computing Systems*, IEEE Transactions on Computers Vol. 31 No. 1, January 1982
- [Mirc89] R. Mirchandaney, D. Towsley, J. Stankovic, *Analysis of the Effects of Delays on Load Sharing*, IEEE Transactions on Computers Vol. 38 No. 11, 1989
- [Mirc89a] R. Mirchandaney, D. Towsley, J. Stankovic, *Adaptive Load Sharing in Heterogeneous Systems*, Proceedings Distributed Computing Systems, 1989
- [MPI94] Message Passing Interface Forum, *MPI: A message-passing interface standard*, International Journal of Supercomputer Applications, Vol. 8 No. 3/4, 1994
- [Mutk87] M. Mutka, M. Livny, *Scheduling Remote Processing Capacity In A Workstation-Processor Bank Network*, IEEE Proc. 7th Int. Conf. Distributed Computing Systems, 1987
- [Mutk92] M. Mutka, *Estimating Capacity For Sharing in a Privately Owned Workstation Environment*, IEEE Transactions on Software Engineering Vol. 18 No. 4, 1992
- [Osse92] W. Osser, *Automatic Process Selection for Load Balancing*, Master Thesis, University of California, Santa Cruz, 1992
- [Papa87] C. Papadimitriou, J. Tsitsiklis, *On Stochastic Scheduling with In-Tree Precedence Constraints*, SIAM J. Comput. Vol. 16 No. 1, 1987
- [PVM93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM 3 Users's Guide and Reference Manual*, Oak Ridge Nat. Lab., Internal Report ORNL/TM-12187, 1993
- [Rahm86] E. Rahm, *Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen*, Angewandte Informatik 4/86, 1986
- [Rahm93] E. Rahm, R. Marek, *Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems*, Proc. 19th VLDB Conference, Dublin, 1993
- [Ross91] K. Ross, D. Yao, *Optimal Load Balancing and Scheduling in a Distributed Computer System*, Journal of the ACM Vol. 38 No. 3, 1991
- [Sale90] V. Saletoore, *A Distributed and Adaptive Dynamic Load Balancing Scheme for Parallel Processing of Medium-Grain Tasks*, Proceedings 5th Distributed Memory Computing Conference (DMCC5), 1990
- [Scha92] J. Schabernack, *Lastenausgleichsverfahren in verteilten Systemen - Überblick und Klassifikation*, Informationstechnik Vol. 34 No. 5, 1992
- [Shen85] C. Shen, W. Tsai, *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion*, IEEE Transactions on Computers, Vol. 34 No. 3, 1985
- [Shir92] B. Shirazi, K. Kavi, *Parallelism Management: Synchronisation, Scheduling and Load Balancing*, Tutorium, University of Texas at Arlington, 1992

- [Sinh93] A. Sinha, L. Kale, *A Load Balancing Strategy For Prioritized Execution of Tasks*, IEEE 7th International Parallel Processing Symposium, 1993
- [Smit80] R. Smith, *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, IEEE Transactions on Computers Vol. 29 No. 12, 1980
- [Stra88] B. Stramm, F. Berman, *Communication-Sensitive Heuristics and Algorithms for Mapping Compilers*, ACM SIGPLAN Vol. 23, No. 9, 1988
- [Tant85] A. Tantawi, D. Towsley, *Optimal Static Load Balancing in Distributed Computer Systems*, Journal of the ACM Vol. 32 No. 2, 1985
- [Thei89] M. Theimer, K. Lantz, *Finding Idle Machines in a Workstation-Based Distributed System*, IEEE Transactions on Software Engineering Vol. 15 No. 11, 1989
- [Tilb81] A. van Tilborg, L. Wittie, *Wave Scheduling: Distributed Allocation of Task Forces in Network Computers*, IEEE Proceedings Distributed Computing Systems, 1981
- [Tilb84] A. van Tilborg, L. Wittie, *Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers*, IEEE Transaction on Computers Vol. 33 No. 9, 1984
- [Vara88] R. Varadarajan, E. Ma, *An Approximate Load Balancing Model with Resource Migration in Distributed Systems*, Proceedings Parallel Processing, 1988
- [Will91] R. Williams, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency: Practice and Experience Vol. 3 No. 5, 1991
- [Winc92] A. Winckler, *Context-Sensitive Load Balancing in Distributed Computing Systems*, Proc. ISCA Int. Conf. on Computer Applications in Industry and Engineering, Honolulu, 1993
- [Yu86] P. Yu, S. Balsamo, Y. Lee, *Dynamic Load Sharing in Distributed Database Systems*, Proceedings Fall Joint Computer Conference, 1986
- [Yu91] P. Yu, A. Leff, Y. Lee, *On Robust Transaction Routing and Load Sharing*, ACM Transactions on Database Systems Vol. 16 No. 3, 1991
- [Zhou87] S. Zhou, D. Ferrari, *An Experimental Study of Load Balancing Performance*, Report No. 87/336, Computer Science Division, University of California, Berkeley, 1987
- [Zhou92] S. Zhou, X. Zheng, J. Wang, P. Delisle, *Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems*, Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Canada, 1992
- [Zhu92] Y. Zhu, C. McCreary, *Optimal and Near Optimal Tree Scheduling for Parallel Systems*, Proc. IEEE Symposium on Parallel and Distributed Processing, 1992

Lebenslauf

11/1966	geboren in Hoengen (Nordrhein-Westfalen)
9/1972 - 8/1976	Grundschule in Marbach (Baden-Württemberg)
9/1976 - 6/1985	Gymnasium in Marbach
7/1985 - 9/1986	Wehrdienst in Walldürn (Baden-Württemberg)
10/1986 - 9/1990	Informatikstudium an der Universität Stuttgart
11/1990 - 9/1995	wissenschaftlicher Mitarbeiter am Institut für Parallele und Verteilte Höchstleistungsrechner der Universität Stuttgart

