veröffentlicht vom

Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR)

in der Abteilung Anwendersoftware (Prof. Dr. A. Reuter)

Fakultät für Informatik

Universität Stuttgart

Breitwiesenstr. 20-22, D-70565 Stuttgart

Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen in Workstation-Netzen

Wolfgang Becker

Fakultätsbericht Nr. 1994 / 4

CR-Klassifikation C.2.4, C.4, D.4.8

Wolfgang.Becker@informatik.uni-stuttgart.de

Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen in Workstation-Netzen

Kurzfassung:

Das hier vorgestellte Modell versucht im wesentlichen durch drei Ansätze, die Leistungssteigerungen durch dynamische Lastbalancierung für ein deutlich breiteres Feld von Anwendungen und Systemen anwendbar zu machen, als es derzeit möglich ist.

Teilsysteme werden durch zentrale Balancierungsverfahren verwaltet; erst zwischen Teilsystemen werden dezentrale Verfahren eingesetzt, um hohe Skalierbarkeit zu gewährleisten. Das ermöglicht Lastverteilung im Zusammenspiel der Anwendungen und Ressourcen und vermeidet kontra-produktive Entscheidungen unabhängiger Balancierungskomponenten.

Der zweite wichtige Ansatz ist die Berücksichtigung mehrerer Ressourcen für Lastbalancierungsentscheidungen. So können Verteilungsstrategien im HiCon-Modell beispielsweise sowohl Prozessorauslastungen als auch Datenaffinitäten der Anwendungen im Entscheidungsalgorithmus kombinieren. Das eröffnet der Lastbalancierung ein weiteres Spektrum unterschiedlicher Anwendungen auf heterogenen Systemen.

Als dritter Schwerpunkt soll Lastbalancierung im HiCon-Modell adaptiv auf aktuelle Systemlast- und Anwendungsprofile reagieren können. Dazu wird eine Sammlung einfacher Strategien verwaltet, zwischen denen zur Laufzeit umgeschaltet wird, um wechselnde Lastprofile mit möglichst geringem Zusatzaufwand auf dem System zu balancieren.

Der Artikel stellt die Grundideen vor und validiert die Konzepte durch Messung verschiedener Anwendungen auf einem heterogenen Rechnernetz.

Abstract:

The HiCon model investigates three approaches to exploit performance improvement by dynamic load balancing for a wider range of applications and systems than it is possible nowadays.

Centralized load balancing schemes manage cells of the whole system. Between cells decentralized strategies are employed to achieve high scalability. This enables harmonized load balancing of applications and resources and avoids contra-productive decisions of independent load balancing agents.

The second approach is the consideration of multiple resources for load balancing decisions. For example, load balancing policies in the HiCon model may combine both processor utilization and data affinities of the applications for decision making. This opens a wide range of different application types for load balancing in heterogeneous environments.

The third issue is the facility to dynamically adapt HiCon load balancing to current system load and application profiles. Therefore, a collection of simple strategies is maintained. At run-time, load balancing switches between these strategies to distribute changing load patterns with minimal overhead across the system.

This article introduces the main ideas and validates the concepts through measurement of different applications on a heterogeneous computer network.

Inhaltsverzeichnis

1 Einleitung

1.1 Das Lastverteilungsproblem in Parallelrechnern und Rechnernetzen

Parallele Rechnersysteme und Computernetze bieten ein Vielfaches der Rechenleistung, die selbst durch einen schnellen Einzelrechner alleine erreicht werden kann. Daher scheinen sich parallele und verteilte Rechnersysteme noch in diesem Jahrzehnt zur Lösung der anstehenden rechenaufwendigen Probleme (große verteilte Datenbanken, numerische Simulationen, Bildverarbeitung, etc.) durchzusetzen.

Große Rechnersysteme und Netze werden nicht exklusiv durch eine Anwendung belegt, sondern viele Benutzer und Rechenaufgaben sind gleichzeitig aktiv. Das 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.

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 Teilen 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 (Synchronisation und Datenaustausch) gering gehalten wird.Gegenüber der Balancierung vieler unabhängiger Anwendungen bringt die Balancierung parallelisierter Anwendungen die Probleme der Datenkommunikation mit sich.

Im Gegensatz zu lokaler Lastbalancierung, die einzelne Aufträge so schnell als möglichst abzuarbeiten versucht, ist es Hauptziel einer globalen Optimierung, den Gesamtdurchsatz des Systems zu maximieren. Dabei werden Nachteile für einzelne, unkritische Aufträge in Kauf genommen.

In dem hier vorgestellten Projekt sollen heterogene Gemische aus konkurrierend laufenden parallelisierten Anwendungen automatisch balanciert werden, so daß der Gesamtdurchsatz des Systems maximal wird.

1.2 Datenintensive Anwendungen auf lose gekoppelten Systemen

Nicht alle Sorten von Anwendungen lassen sich durch eine einheitliche Methode auf parallelen Systemen erfolgreich balancieren. Neben der ungeheuren Komplexität eines allumfassenden Verfahrens ist vor allem der damit verbundene Aufwand prohibitiv. Da in der Praxis nicht alle zur Lastverteilung relevanten Faktoren im voraus bekannt sind, muß dynamische Lastbalancierung während der Laufzeit der Anwendungen Entscheidungen treffen. Das verursacht einerseits erheblichen Rechenaufwand und verbraucht so selbst einen Teil der Rechenkapazitäten; andererseits bringt es Verzögerungen mit sich, weil eine Teilaufgabe erst dann berechnet werden kann, wenn sie ihrem Zielknoten zugeteilt und dort eingeplant ist.

Wir beschränken uns im HiCon-Modell auf eine Klasse von Anwendungen, in der große, persistente Datenbestände verarbeitet werden. Die später vorgestellten Anwendungen zeigen, daß sich die Anwendungsklasse nicht allein auf typische Datenbank-Funktionen beschränkt. In Bezug auf die Lastverteilung haben diese Anwendungen die Eigenschaft, daß nicht nur die Nutzung der Rechenleistung, sondern auch Beachtung von Datenaffinitäten für den Durchsatz entscheidend ist. Zudem können häufig beim Start einer Teilaufgabe Abschätzungen über Rechenaufwand und benutzte Datensätze gemacht werden. Das befähigt die Lastbalancierung, nicht nur auf Beobachtung der Systemlast hin zu reagieren, sondern die Last bereits bei der Entstehung vorausschauend zu plazieren.

Das Granulat sinnvoller Parallelität und die Möglichkeiten der Lastverteilung sind auf verschiedenen Rechnerarchitekturen sehr unterschiedlich. So können auf SIMD Rechnern sehr fein parallelisierte Algorithmen ablaufen, auf Systemen mit gemeinsamem Speicher kann aufgrund geringer Nachrichtenkosten feingranular auf gemeinsamen Daten operiert werden. Unsere Untersuchungen beziehen sich jedoch auf lose gekoppelte Systeme mit schnellen Prozessoren, die über relativ langsame Verbindungen verfügen und keine gemeinsamen Ressourcen haben. Wir legen daher ein heterogenes Netz von Workstations und grobgranularen MIMD Parallelrechnern zugrunde.

Die lose Kopplung der Rechenknoten verlangt, daß Anwendungen in einige große, in sich sequentielle Teilaufgaben zerlegt sind. Lastbalancierung hat darauf zu achten, daß der Aufwand zur Synchronisation und für den Datenaustausch signifikant und daher gering zu halten ist. Zudem ist in derartigen Rechnernetzen zu berücksichtigen, daß meist mehrere unabhängige Anwendungen konkurrierend auf dem System abzuwickeln sind.

1.3 Relevante Forschungsarbeiten und Klassifikation

Der Großteil der Forschungsprojekte im Bereich der Lastverteilung auf parallelen Systemen hat sich, ausgehend von der statischen Vorplanung für große Batch-Anwendungen (Scheduling, statische Anfrageoptimierung), verstärkt in Richtung sehr einfacher, verteilter Verfahren zum dynamischen Lastausgleich unkorrelierter Anwendungen bewegt.

Die Grundidee dieser Ansätze ist folgende: Auf jedem Prozessor im System mißt eine Lastverteilungskomponente die Prozessorlast (dabei sind verschiedene Metriken möglich) und tauscht diese Werte periodisch, oder nach signifikanten Änderungen, mit benachbarten Prozessoren aus. Wenn auf einem stark belasteten Prozessor ein neuer Auftrag gestartet wird, so versucht die Lastverteilungskomponente, ihn an den Nachbar abzugeben, der zur Zeit am geringsten belastet ist. Einige Verfahren setzen auch die Migration laufender Prozesse ein, um bestehende Ungleichverteilungen in der Systemlast zu verringern. Hier versucht die Lastverteilungskomponente, wenn ihr Prozessor erheblich stärker belastet ist als einer der benachbarten, einen laufenden Prozeß an diesen abzugeben.

Für Übersichten und Klassifikationen der diversen Ansätze verweisen wir auf [13], [25] und [44]. Auf dem Gebiet der statischen Zuweisung von parallelisierten Anwendungen finden sich zahlreiche Ergebnisse: In [7], [10] und [37] werden Aufträge zur Minimierung der Kommunikationskosten auf Prozessoren gruppiert; [8] und [38] verteilen unabhängige Aufträge so, daß die Gesamtlaufzeit minimal wird, bzw. daß die Knotenauslastungen gleich sind ([26], [34]). Zusätzlich werden in [11] und [14] Reihenfolgebeziehungen zwischen Aufträgen berücksichtigt. Bei [31] und [47] werden Transferraten für gegebenes statistisches Lastaufkommen ermittelt, um die mittlere Antwortzeit einzelner Aufträge zu minimieren.

In [29] und [43] werden statische Lastverteilungsverfahren um dynamische Anpassungen erweitert. Zentral gesteuerte dynamische Lastbalancierungstechniken finden sich in [9], [36], [48] sowie [49]. Viele Ansätze wurden zur dezentralen dynamischen Auftragszuweisung veröffentlicht; man unterscheidet, ob überlastete Knoten neue Aufträge weitergeben ([19], [27], [30], [35], [39], [54]), freie Knoten Aufträge anfordern ([19], [39]), oder explizite Absprachen zwischen benachbarten Knoten stattfindet [2], [23], [45], [46]. Die Projekte [17], [18], [20], [22], [32], [40] sowie [55] beschäftigen sich mit der Migration laufender Prozesse zum Lastausgleich bzw. modifizieren die Auftragsgrößen zur Laufzeit [12]. Dynamische Adaption der Lastbalancierungsstrategien, d.h. Anpassung von Schwellwerten und Meßintervallen, wird in [21] und [33] untersucht. Im Bereich der datenintensiven Anwendungen finden sich Ansätze in [15], [16], [24], [28], [42], [50] und [52].

Wir wollen das HiCon-Modell[1] lediglich anhand zweier Kriterien in das Umfeld dieser Lastbalancierungsarbeiten einordnen. Abbildung 1 zeigt die Informationen, die Lastbalancierungsmethoden verwenden können. Da die Lastverteilungskomponente im HiCon-Modell als Teil des Betriebssystems angesehen wird, dürfen keine Eigenschaften und Kenntnisse einer speziellen Anwendung

1. Das Acronym *HiCon* ist älteren Ursprungs und steht für *HIerarchical COntrolled Network computing*.

eingesetzt werden. Andererseits genügt es in der hier betrachteten Anwendungsklasse nicht, die Ressourcenauslastung auf Betriebssystemebene zu messen und für Entscheidungen zu verwenden, sondern Anwendungen können der Lastbalancierung durch Profil-Abschätzungen und Datenzugriffs-Charakteristiken Vorabinformationen über Teilaufgaben liefern.

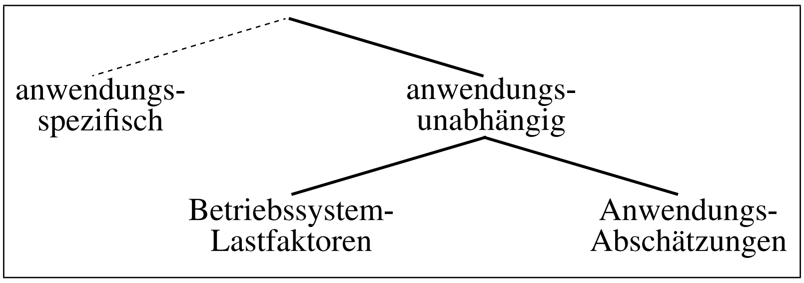

Abbildung 1: Klassifikation der genutzten Lastbalancierungsinformationen.

In Abbildung 2 sind die Aufgaben und damit Einflußmöglichkeiten der Lastbalancierung klassifiziert. Wir beschränken uns im HiCon-Modell in der Auftragsverwaltung auf die Zuweisung von Aufträgen, da die Migration laufender Prozesse in heterogenen Systemen noch nicht mit vernünftigem Aufwand realisierbar ist. Die Möglichkeit der Auftragsduplikation ist nur für voll funktionale Teilrechnungen, keinesfalls für datenintensive Berechnungen anwendbar: Wenn man einen Auftrag mehrfach ausgibt, um dann das Ergebnis des schnellsten Bearbeiters weiter zu verwenden, darf der Auftrag keine globalen Daten verändern. In der Datenverwaltung stehen den Methoden im HiCon-Modell sowohl die Möglichkeit zur Verlagerung von Daten als auch zur (korrekt synchronisierten) Verteilung von Datenkopien zur Verfügung.

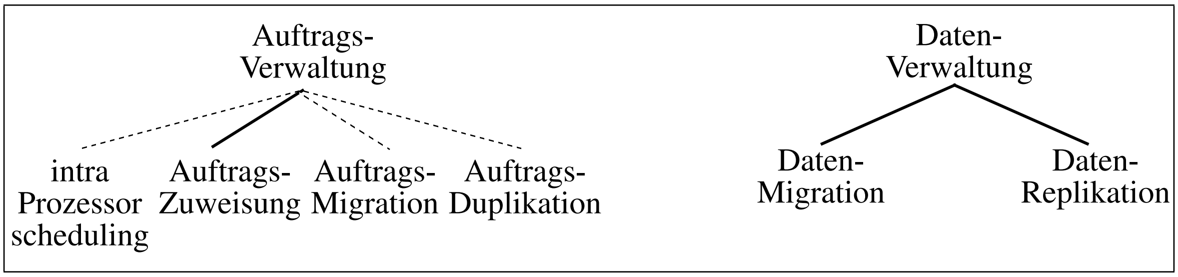

Abbildung 2: Klassifikation der Lastbalancierungsaufgaben.

2 Der Lastbalancierungsansatz im HiCon-Modell

Die Untersuchungen im HiCon-Modell ([3], [4]) basieren auf dem Konzept einer Lastbalancierungsumgebung, die es ermöglicht, Effekte verschiedener Methoden zur Lastverteilung anhand verschiedenartiger Anwendungen auf einem heterogenen Netz von Workstations und Parallelrechnern zu evaluieren. Die Anforderungen an eine solche Umgebung sind ein relativ allgemeines Modell für das Rechnersystem und die Anwendungen, um praxisrelevante Resultate zu erhalten, sowie geeignete Informations- und Eingriffsmöglichkeiten für Lastverteilungsstrategien. Die Umgebung selbst muß so effizient sein, daß paralleles Rechnen nutzbar und verschiedene Balancierungsverfahren unterscheidbar bleiben. In diesem Abschnitt werden die wesentlichen Eigenschaften des Ansatzes vorgestellt.

2.1 Ablaufmodell für Anwendungen

Je allgemeiner die Ablaufstrukturen, die Kooperation und die Synchronisation in parallelisierten 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 ein beliebiges Netz kommunizierender Prozesse betrachtet wird. Automatische Lastverteilung ist hier aufgrund der mangelnden Informationen über die laufenden Anwendungen nur begrenzt möglich. Extreme Leistungssteigerungen wurden hingegen in Projekten erreicht, die für spezielle parallelisierte Anwendungen dedizierte Lastausgleichsverfahren entwickelt haben.

Im HiCon-Modell wird als Ablaufstruktur das Client-Server-Konzept vorausgesetzt, das sich in Datenbank-Umgebungen und bei nahezu allen größeren, parallelen und verteilten Anwendungen bewährt hat. Die Teilfunktionen einer Anwendung werden als Serverklassen bezeichnet. Prozesse (allgemeiner: Ausführungseinheiten), die eine Teilfunktion ausführen können, sind die Server einer Klasse. Eine Anwendung besteht nun aus Clients, die Aufrufe an Serverklassen durchführen, und Servern, die diese Aufrufe bearbeiten. Server können selbst Unteraufrufe an andere Serverklassen absenden und nehmen dabei die Rolle eines Clients an.

In datenintensiven Anwendungen sollten persistente Daten explizit im Verarbeitungsmodell berücksichtigt werden. Im HiCon-Modell arbeiten Anwendungen mit globalen Datensätzen, die über Namen identifiziert werden und für die Dauer einer Anwendung existieren oder persistent sind. Globale Daten können beliebige Hauptspeicher- und Sekundärspeicher-Datenstrukturen sein. Synchronisation, Datenlokalisierung und Replikationsverwaltung ist für die Anwendung unsichtbar. Server schützen Zugriffe auf globale Datenbestände durch geeignete Lese- bzw. Änderungssperren.

Dieses Ablaufmodell verlangt in der Praxis, daß parallelisierte Anwendungen umstrukturiert werden müssen. Die unten angeführten Beispielanwendungen zeigen jedoch, daß ein breites Spektrum an Anwendungen sinnvoll nach dem Client-Server-Konzept mit gemeinsamen Datensätzen ablaufen kann. Dadurch ermöglicht das HiCon-Modell anwendungsunabhängige Lastbalancierung unter Einsatz vieler relevanter Lastfaktoren.

2.2 Die Struktur der Lastbalancierung

Abbildung 3 gibt einen Überblick über die Komponenten zur Abwicklung der Lastbalancierung, die Komponenten der Anwendungen (Clients, Server und Datenbestände) und das umgebende Laufzeitsystem. Das Laufzeitsystem sollte Teil eines geeigneten verteilten Betriebssystems sein, wobei manche Teile heutzutage als Datenbankkomponente oder Transaction-Processing Monitor realisiert werden. Charakteristisch sind in diesem Systemmodell - im Vergleich zu vielen anderen Projekten - zum einen die logisch zentralisierten Funktionen der Lastverteilung (Verwaltung der Systemzustands-Information und Treffen der Balancierungsentscheidungen), zum anderen die Aufteilung der Lastbalancierung in eine unmittelbar agierende Komponente und eine sogenannte strategische, welche die dynamische Adaption der Lastverteilung ermöglicht. Es wird keine spezifische Prozessor- oder Netzwerktopologie vorausgesetzt. Das Bild gibt die Struktur eines Teilsystems wieder. Zur Skalierung auf sehr große Systeme können beliebig viele Teilsysteme dieser Art vernetzt werden (siehe Abschnitt 2.5).

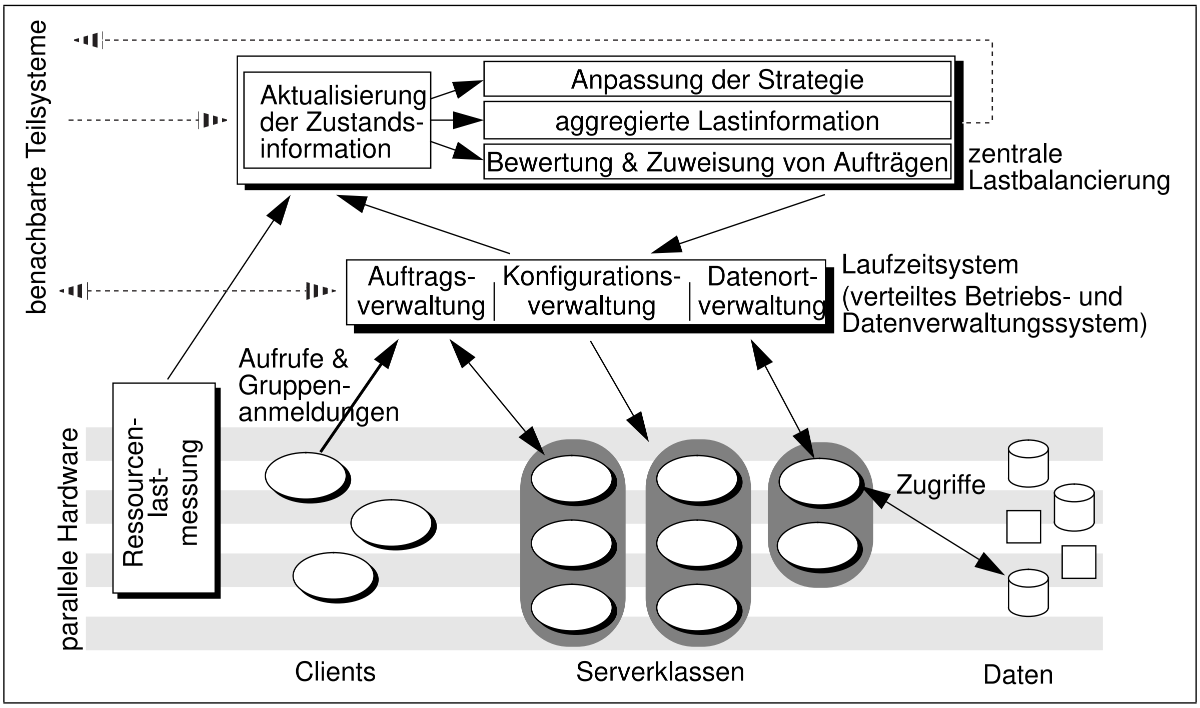

Abbildung 3: Struktur der Lastbalancierungsumgebung.

Die Komponentenstruktur zeigt lediglich die logischen Zusammenhänge. In einer Realisierung werden Teile der Lastverteilungskomponente als zentraler Prozeß, andere Teile als Verwaltungsprozeß je Knoten angelegt und weitere Teile werden als Bibliotheksfunktionen an die Server der Anwendung gebunden.

Lastbalancierung darf das laufende System nicht durch unnötige Aktivitäten stören. Daher ist im HiCon-Modell nur die Ressourcen-Lastmessungskomponente periodisch aktiv. Sie beobachtet die Auslastung der Prozessoren, Hauptspeicher, Platteneinheiten sowie der Netzwerkverbindungen. Signifikante Änderungen werden in der Systemzustandstabelle eingetragen.

Die eigentliche Lastbalancierungskomponente ist passiv, d.h. sie reagiert auf Ereignisse. Für die Lastverteilung relevante Ereignisse sind im folgenden, nach Häufigkeit sortiert, aufgelistet:

- Entstehung neuer Aufträge. Clients schicken Aufrufe ab, die durch einen Server einer bestimmten Klasse zu bearbeiten sind.
- Änderungen im Arbeitszustand von Servern. Wie unten erläutert, haben Server lokale Auftragswarteschlangen. Sie können zu beliebigen Zeitpunkten Informationen abgeben, wie weit sie den aktuellen Auftrag bearbeitet haben, und wie viele Aufträge sich derzeit in der Warteschlange befinden. Der einfachste Fall besteht darin, am Ende einer Berechnung mit dem Ergebnis eine Fertigmeldung zu geben.
  Auch Änderungen in der Server-Konfiguration sind möglich.
- Änderungen in der aktuellen Auslastung von Ressourcen. Als Ressourcen werden hier Prozessoren, Hauptspeicher, Platten und Netzverbindungen angesehen. Auch Änderungen in der Systemkonfiguration (Hardware-Komponenten) fallen in diese Ereignisklasse.

- Bewegung von Datensätzen und Entstehung von Datenkopien. Dazu gehört auch die Entstehung von globalen Datensätzen, denn Server können zur Laufzeit Datensätze erzeugen und löschen.
- Zustandsänderungen in einer benachbarten Balancierungskomponente. Bei signifikanten Änderungen im aggregierten Zustand einer Balancierungskomponente (siehe Abschnitt 2.5) benachrichtigt diese ihre unmittelbaren Nachbarn.
  Zur Laufzeit können auch Änderungen in der hierarchischen Struktur der Lastverteilungskomponenten auftreten, da diese Struktur dynamisch ist.

Je nach momentan aktiver Lastbalancierungsstrategie werden manche dieser Ereignisse berücksichtigt, andere ignoriert. Wie unten erklärt wird, muß das Ziel darin bestehen, stets eine möglichst simple Strategie zu verwenden. Im Regelfall werden daher nur wenige Ereignisse größere Balancierungs-Aktionen nach sich ziehen. Als Folge eines Ereignisses können je nach derzeit aktiver Lastverteilungsstrategie

- Einträge in der Zustandstabelle vorgenommen werden,
- Routing-Tabellen und Bewertungen von Aufträgen, Servern oder möglichen Zuweisungen aktualisiert werden,
- Aufträge an Server zugewiesen werden,
- Daten umverteilt bzw. Datenkopien kreiert werden,
- Server zugefügt oder abgeschaltet werden und
- andere Lastbalancierungseinheiten zugefügt oder abgeschaltet werden.

2.3 Auftragsverwaltung und Datenverwaltung

Abbildung 4 zeigt die Struktur der Auftragsverwaltung im HiCon-Modell. Ein Auftrag ist ein Aufruf eines Clients an einen unbestimmten Server einer bestimmten Klasse. Es ist nicht möglich, stehende Verbindungen (Sessions) zwischen einem Client und einem bestimmten Server herzustellen. Derartiger Verarbeitungskontext muß in Form globaler Daten in der Serverklasse gespeichert werden.

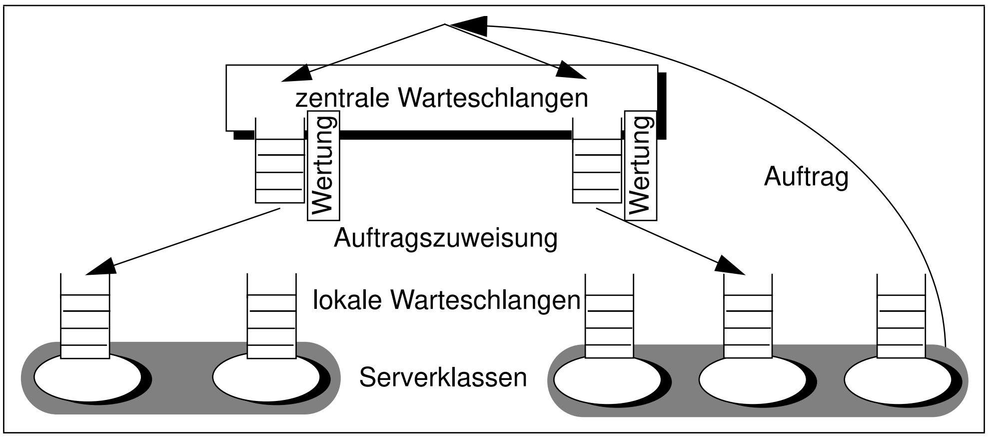

Abbildung 4: Auftragsverwaltung der Lastbalancierungsumgebung.

Der Auftrag wird zunächst in eine zentrale Warteschlange für die Ziel-Serverklasse eingereiht und durch die Lastbalancierung bewertet. Nun kann der Auftrag zu einem beliebigen Zeitpunkt an einen Server zugewiesen werden; er braucht dazu nicht am Anfang der zentralen Warteschlange zu stehen und die lokale Warteschlange des Servers muß auch nicht leer sein. Diese Zuweisung ist endgültig, d.h. es werden keine Aufträge zwischen lokalen Warteschlangen ausgetauscht. Jeder Server arbeitet die Aufträge in seiner Warteschlange der Reihe nach ab (sofern er nicht multithreaded operiert). Resultate gelangen zum Aufrufer zurück, der entweder direkt auf das Resultat wartet (synchroner Aufruf) oder später auf verschiedene Resultate wartet (parallele Aufrufe) oder kein Resultat erwartet.

Lastverteilungsstrategien können bei der Ankunft eines Auftrags Wertungen berechnen, wie vorteilhaft momentan eine Zuweisung an welchen Server ist (z.B. nach welcher Zeit der Auftrag dort vollendet würde) und ihn evtl. sofort zuweisen. Wenn sich danach der Lastzustand im System, der Arbeitszustand eines Servers oder die Datenverteilung ändern, können die Strategien die wartenden Aufträge noch einmal durchsehen, neu bewerten und evtl. zuweisen.

Durch diese logisch zentralisierte Auftragsverwaltung ist die übliche Unterscheidung zwischen sender- und empfängerinitiierten Lastverteilungsstrategien nicht notwendig. Im HiCon-Modell können viel allgemeiner an verschiedene Ereignisse (siehe oben) unterschiedliche Reaktionen der Lastbalancierung geknüpft werden.

Weiterhin können Clients Auftragsgruppen anmelden. Dabei bezeichnet der Client die erwarteten Aufträge mit Namen und kann Informationen über die Gruppe angeben, wie z.B. Reihenfolgeabhängigkeiten zwischen Aufträgen und vermutete Auftragsgrößen. Die Aufträge der Gruppe laufen später als gewöhnliche Serverklassenaufrufe ein, die mit den Namen versehen sind. Die Lastbalancierung kann sie daran identifizieren und sie den vorgegebenen Reihenfolgebeziehungen oder Datenaffinitäten innerhalb der Gruppe angepaßt behandeln. Ein typisches Beispiel für derartige Auftragsgruppen sind komplexe Datenbankoperationen. Die Teilaufträge eines Anfragegraphen haben gewisse Ausführungsreihenfolgen und arbeiten auf Zwischenresultaten von Vorgänger-Teilaufträgen. Hier ist es wichtig, die Aufträge in kritischen Pfaden vorrangig abzuwickeln, um die Gesamtausführungszeit der Auftragsgruppe zu minimieren. Positive Resultate werden in [6] bzw. [51] vorgestellt.

Wie die Auftragsverwaltung im Client-Server-Konzept, so ist auch die Datenverwaltung im HiCon-Modell an Konzepte aus dem Datenbank-Bereich angelehnt. Datensätze sind durch Anwendungen definierte, mit Namen identifizierbare Objekte. Einzelne Datensätze können zwischen Servern ausgetauscht und kopiert werden. Der aktuelle Ort von Datensätzen und Kopien wird durch die Zugriffsanforderungen der Server bestimmt: Ein Server fordert Lese- oder Änderungszugriff auf einen Datensatz und erhält daraufhin die Daten. Durch Lastbalancierungsentscheidungen kann die Datenverteilung und der Replikationsgrad indirekt über die Zuweisung von Aufträgen beeinflußt werden.

Die Lastbalancierung kann also die Lokalität von Datenzugriffen begünstigen, indem sie Server benutzt, die bereits die voraussichtlich benötigten Daten besitzen, oder indem sie Daten auf die Server verteilt, die später darauf arbeiten sollen. Das setzt voraus, daß Clients zu ihren Aufträgen Abschätzungen über vermutliche Datenreferenzen mitgeben (siehe Abschnitt 2.4). Keine Lastbalancierungsmethode kann jedoch den korrekten Ablauf der Anwendung stören, da die Daten automatisch auf Zugriffsanforderungen hin geschickt werden und dort bleiben, bis die Sperren freigegeben sind.

2.4 Lastkenngrößen

Strategien zur dynamischen Lastverteilung benötigen zur Laufzeit aktuelle Informationen über die Belastung der Ressourcen und den Verlauf der Anwendung. Man unterscheidet zumindest zwischen aktuellen Zustandskenngrößen und aggregierten Angaben, die meist Mittelwerte über kurze Zeitintervalle sind. Die Gewinnung und Verwaltung akkurater Informationen sowie die Nutzung dieser Informationen für Balancierungsentscheidungen bringen hohen Aufwand mit sich (siehe Abschnitte 2.5 und 2.6). Wir listen daher in Tabelle 1 alle verfügbaren Kenngrößen auf mit dem Hinweis, daß keine Balancierungsstrategie auch nur annähernd alle Größen zusammen nutzen kann:

Objekt	aktuelle Zustandsgrößen	aggregierte Informationen
Serverklasse	• Zahl der vorhandenen Server • Zahl der unbeschäftigten Server • Zahl der Aufträge in der zentralen Warteschlange	• mittlere Zahl freier Server • mittlere Länge der zentralen Auftragswarteschlange • mittlere Länge der lokalen Auftragswarteschlangen aller Server • Ankunftsrate der Aufträge • mittlere Bearbeitungszeit und Datenwartezeit je Auftrag, • Leerlaufzeit zwischen zwei Aufträgen der Server
Server	• Länge der lokalen Auftragswarteschlange + verbleibender Restanteil des momentan bearbeiteten Auftrags • vermuteter Zeitpunkt, an dem der Server seine Warteschlange komplett abgearbeitet hat	• Ankunftsrate der Aufträge • mittlere Bearbeitungsdauer pro Auftrag • Rechenzeitanteil, Anteil an Wartezeit auf Daten pro Auftragsbearbeitung • mittlere Leerlaufzeit zwischen zwei Bearbeitungen
Auftrag bzw. Auftragsgruppe	• erwarteter Rechenumfang • Zahl der Plattenzugriffe • Abschätzungen über zu lesende und zu ändernde Datensätze oder Bereiche von Daten. • vermutete Ausführungsdauer unter derzeitigen Bedingungen auf bestimmten Servern	• Reihenfolge-Beziehungen zwischen den Aufträgen der Gruppe • Weiterverwendung derselben Datensätze in der Gruppe • aktueller Bearbeitungszustand der Auftragsgruppe
Datensatz	• Speicherplatzbedarf • Ort des Originals • Kopienverteilung im System	• mittlere Transfer-Kosten • Häufigkeit der Datenverschiebungen • mittlere Anzahl existierender Kopien.

Tabelle 1: Mögliche Zustandsinformationen über System und Anwendung.

Objekt	aktuelle Zustandsgrößen	aggregierte Informationen
Rechen-knoten und Verbindungen	• Rechenleistung, Speicherplatz, Plattenzugriffszeit (statisch) • Nachrichtendurchsatz /-Verzögerung (statisch) • Zahl der hier liegenden Server • freier Speicher	• Run Queue Length (mittlere Zahl laufbereiter Prozesse) • Speicherseiten-Auslagerungsrate (Paging und Swapping) • Nutzungszeit der Platten-Controller • Netzauslastung (Nutzungszeit oder Anteil von Kollisionspaketen)
Teilsystem	• aggregierte Rechenleistung • mittlere Belastung der Prozessoren • Warteschlangenlänge je Klasse • mittlere Länge der lokalen Server-Warteschlangen je Klasse	• aggregierte Rechenleistung • mittlere Belastung der Prozessoren • Warteschlangenlänge je Klasse • mittlere Länge der lokalen Server-Warteschlangen je Klasse

Tabelle 1: Mögliche Zustandsinformationen über System und Anwendung.

Informationen über Aufträge können vom Client beim Aufruf mitgegeben werden. Diese Angaben sind lediglich Zusatzhinweise für die Lastbalancierung und haben keinen Einfluß auf den korrekten Ablauf der Anwendung. Weiterhin kann die Lastbalancierung eigene Bewertungen zu den Aufträgen abspeichern.

Informationen über den Systemzustand benachbarter Lastbalancierungskomponenten (siehe Abschnitt 2.5) werden ebenfalls pro Serverklasse des Nachbarn ausgetauscht. Allerdings werden sämtliche Daten über die Systemleistung und -Belastung des Nachbarn auf Mittelwerte pro Serverklasse komprimiert, da eine Lastbalancierungseinheit weder die einzelnen Server noch die Systemkonfiguration der Nachbarn kennen soll.

Die direkt auf Hardware basierenden Lastfaktoren wurden absichtlich zuletzt aufgeführt, um zu betonen, daß Lastbalancierung im HiCon-Modell - obwohl sie anwendungsunabhängig bleibt - auch Faktoren auf höherer Ebene sinnvoll einsetzen kann. Die meisten Forschungsprojekte haben sich bisher darauf beschränkt, die Faktoren auf Systemebene zu betrachten.

2.5 Netzwerk unabhängiger Lastverteilungskomponenten

Zahlreiche Studien haben nachgewiesen, daß eine zentralisierte Lastverteilung nicht unbegrenzt skalierbar ist. Wird das zu kontrollierende System oder die Anzahl und Häufigkeit von Aufträgen sehr groß, so verbraucht die Balancierungskomponente selbst viel Speicher und Rechenkapazitäten, und die Verzögerung zwischen Absendung eines Auftrages und dessen Bearbeitungsbeginn wächst an. Zentralisierte Lastverteilung wird also 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 Auftragsbearbeitungszeit liegt.

Zentralisierte Ansätze haben freilich den Vorzug, sämtliche Abhängigkeiten im System und den Anwendungen zu beachten und einzusetzen, während dezentrale Verfahren nur jeweils kleine Teile des Systems balancieren und größere Unausgewogenheiten in der Auslastung bzw. Abhängigkeiten von Anwendungen zwischen den Teilsystemen nicht bemerken bzw. sogar kontra-produktiv agieren. Diese Schwächen treten bei großen Anwendungen und solchen, die auf globalen, verteilten Daten operieren, verstärkt zu Tage.

Im HiCon-Modell bekommt eine Lastverteilungskomponente einen möglichst großen Teil des Systems zur zentralen Balancierung zugewiesen und kann die oben genannten Vorteile nutzen. Was über die Kapazität einer Komponente hinaus geht, wird auf mehrere Komponenten verteilt. Zwischen den Balancierungskomponenten können, völlig transparent für die Anwendungen, Lastinformationen, Aufträge und Daten ausgetauscht werden. Solange Anwendungen innerhalb einer Balancierungskomponente vernünftig ablaufen können und zwischen verschiedenen Komponenten keine allzu großen Lastdifferenzen auftreten, wird alles zentral und somit effizient abgewikkelt. Komponenten-übergreifende Aktionen sind dagegen mit zusätzlichem Aufwand verbunden und treten nur bei groben Auslastungsdifferenzen oder sehr großen Aufträgen bzw. Auftragsgruppen auf.

Die Lastbalancierungskomponenten sind im HiCon-Modell eine im Prinzip beliebig vernetzte Struktur. Direkt benachbarte Komponenten tauschen Lastinformationen und Aufträge untereinander aus. In der derzeitigen Realisierung ist eine Baumstruktur für die korrekte Verwaltung der Anwendungen sowie der Daten durch das Laufzeitsystem notwendig, denn an keiner Stelle im System liegen globale Informationen vor. Ansonsten kennt jede Komponente eine Menge von Nachbarn und jede Komponente kann gleichermaßen Ressourcen, Anwendungen und Daten verwalten.

Zwischen Teilsystemen werden dezentrale Lastbalancierungsverfahren zum Lastausgleich eingesetzt. Lastbalancierer tauschen periodisch Informationen (siehe Abschnitt 2.4) aus und verschieben Aufträge an Teilsysteme, die weniger belastet sind. In diesem Artikel werden jedoch keine weiteren Ergebnisse dazu vorgestellt. In [53] finden sich positive Ergebnisse für Lastbalancierungsverfahren, in denen benachbarte Teilsysteme als besondere Server miteinbezogen werden.

2.6 Die strategische Lastbalancierung

Im vorigen Abschnitt wurden grundlegende Probleme von Lastbalancierungsstrukturen angesprochen und potentielle Engpässe dadurch vermieden, daß die Menge von Informationen und Häufigkeit von Entscheidungen pro zentraler Lastverteilungskomponente in akzeptablem Rahmen bleibt.

Optimale Lastbalancierungsentscheidungen sind jedoch NP-vollständig, d.h. der Aufwand für die bestmögliche Verteilung der Arbeitslast auf das System unter Berücksichtigung aller relevanter Faktoren wächst exponentiell mit der Anzahl der Systemkomponenten und mit der Anzahl der Aufträge im System. Um Lastbalancierung zur Laufzeit durchführen zu können muß man daher auf optimale Lösungen verzichten und möglichst einfache Heuristiken einsetzen, die das theoretische Durchsatz-Optimum annähernd erreichen. Zudem sind die Informationen, auf denen die Balancierungsentscheidungen basieren stets unvollständig, ungenau und veralten sehr schnell, so daß die Voraussetzungen zur Anwendung ‘optimaler’ Verfahren nicht gegeben sind.

Heuristische Balancierungsstrategien sind nicht für jedes System und jede Lastsituation gleichermaßen geeignet. Im HiCon-Modell soll daher eine sogenannte strategische Komponente das System- und Anwendungsverhalten in groben Zügen beobachten und versuchen, jeweils die bestpassende Heuristik zur Lastverteilung zum Einsatz zu bringen. Dazu verfügt die Lastbalancierung über einen Vorrat an einfachen Strategien, die mit Lastkenngrößen versehen sind. Durch periodischen Vergleich des Systemverhaltens mit den Kenngrößen dieser Strategien kann die strategische Komponente erkennen, wann die momentan laufende Strategie durch eine bessere zu ersetzen ist.

Die Herausforderung dieses adaptiven Ansatzes ist weniger die Realisierung der Strategiewechsel als die Bestimmung der relevanten Lastindikatoren, welche die Anwendbarkeit der Strategien unterscheiden. Unten werden einige der bisher verfügbaren Strategien vorgestellt und in Messungen aufgezeigt, daß sie signifikant unterschiedliche Resultate aufweisen. Die Gewinnung und Evaluierung geeigneter Unterscheidungsfaktoren ist derzeitig Hauptgegenstand der Bemühungen im HiCon-Modell. Wir können daher in diesem Artikel noch keine Messergebnisse und Erfahrungen vorstellen.

3 Performance-Evaluierung des HiCon-Modells

3.1 Strategien zur dynamischen Lastverteilung

Die bisher im HiCon-Modell realisierten Strategien zur dynamischen Lastverteilung haben einige Eigenschaften gemeinsam. Zunächst kann jede der Strategien eine beliebige Methode zur initialen Datenverteilung vorgeschaltet werden. Derzeit sind verfügbar:

- *Do Nothing*. Neu entstandene Datensätze bleiben bei dem Server, der sie erzeugt hat.
- *Distribution*. Datensätze werden nach einer Hash-Funktion reihum auf die Server verteilt, d.h. bei *N* Servern wird ein Datensatz zum Server *h(Name) modulo N* migriert.
- *Blocking*. Hier werden Gruppen von Datensätzen reihum auf Server verteilt. Bei *B* Datensätzen pro Block wandert ein Datensatz zum Server $\lfloor h(Name) / B \rfloor$ *modulo N*.

Diese anfängliche Verteilung geschieht jeweils nur innerhalb der Server derselben Klasse und innerhalb einer Lastbalancierungskomponente. Durch Datenanforderungen der Anwendungen oder durch dynamische Lastbalancierungsentscheidungen können die Datensätze im Laufe der Zeit wieder völlig anders verteilt werden.

Weiterhin nutzen alle hier besprochenen Strategien die lokalen Auftragswarteschlangen der Server nur bis zu einer festen Grenze aus. Das muß nicht notwendigerweise so sein; der Einfluß auf den Erfolg der Strategien wird in [5] genauer untersucht. Die meisten Strategien reagieren nur auf zwei Ereignisse: der Ankunft eines neuen Auftrags und der Zustandsänderung eines Servers. Die Strategien *Data Locality* und *DLPS1* reagieren zusätzlich auf die Bewegung von Datensätzen. Bei allen anderen Ereignissen tragen die Strategien lediglich die Änderungen in der Zustandstabelle ein. Die Kooperation zwischen verschiedenen Lastbalancierungskomponenten wird bisher in keiner Strategie genutzt. Die im folgenden skizzierten Strategien versuchen jedesmal, wenn sie aktiviert werden, Aufträge aus der zentralen Warteschlange an geeignete Server zuzuweisen. Momentan führt keine Strategie zusätzliche Aktionen, wie etwa Änderungen der Server-Konfiguration oder Verlagerung von Datensätzen, durch.

- *Round Robin*. Die Server jeder Klasse bekommen reihum Aufträge zugewiesen. Für jeden ankommenden Auftrag steht also sofort fest, von welchem Server er bearbeitet wird. Die Strategie weist allerdings immer nur den ältesten Auftrag zu. Neuere Aufträge müssen daher evtl. warten, wenn der Server, der den ältesten Auftrag erhalten soll, gerade belegt ist (d.h. seine lokale Auftragsschlange bis zur fixen Grenze aufgefüllt ist).

- *First Free*. Im Unterschied zur obigen Strategie versucht diese, den ältesten Auftrag dem nächsten freien Server zuzuweisen; sie überspringt also belegte Server. Sie reagiert dadurch dynamisch auf das Systemverhalten, indem sie den Servern, die Aufträge schneller erledigen, mehr Aufträge zuweist.
- *Shortest Queue*. Ähnlich der *First Free* Strategie werden Aufträge hier jedoch nicht reihum, sondern jeweils dem Server mit der kürzesten lokalen Auftragsschlange zugewiesen. In Situationen hoher Parallelität verhält sie sich genau wie *First Free*, denn es warten stets Aufträge, und sobald ein Server wieder Platz bietet, bekommt er einen weiteren.
- *Data Locality*. Für jeden Auftrag wird der Server herausgesucht, der momentan den größten Teil der vermutlich benötigten Daten lokal vorliegen hat. Dazu werden die Abschätzungen verwendet, die der Client beim Aufruf mitgegeben hat. Wenn dieser beste Server (oder einer, der in einem Toleranzbereich dem besten nahekommt) noch nicht belegt ist, so erhält er den Auftrag. Bei jedem Ereignis wird, beginnend beim ältesten Auftrag, die zentrale Warteschlange soweit abgearbeitet, bis ein Auftrag nicht zugewiesen werden kann.
  Die Strategie versucht also hauptsächlich, dynamisch die Datenkommunikation zwischen den Servern zu minimieren (Nutzung der Datenaffinität). Durch den Toleranzbereich reagiert sie jedoch wie *First Free* auch auf die tatsächlichen Ausführungszeiten der Aufträge.
- *Processor Speed*. Diese Strategie wählt unter den verfügbaren Servern (deren Auftragsschlange nicht voll ist) denjenigen, dessen Prozessor die höchste Rechenleistung bietet. Dazu wird die volle Rechenleistung (durch Benchmarks zu bestimmen) durch die aktuelle sogenannte Run Queue Length dividiert. Die Run Queue Length ist die mittlere Anzahl laufbereiter Prozesse auf dem Prozessor.
  Diese Strategie ist eine andere Verfeinerung von *First Free*, die versucht, die gesamte Prozessorleistung im System voll auszunutzen. Viele Ansätze dezentraler Lastverteilung verwenden diese Metrik zur Bestimmung des geeigneten (Nachbar-) Prozessors oder als Kriterium zur Migration von Aufträgen.
- *DLPS1* (Data Locality & Processor Speed). Diese Methode kombiniert alle Faktoren, die in den letzten drei Strategien betrachtet wurden. Im Prinzip versucht sie jeden Auftrag, beginnend beim ältesten, demjenigen Server zuzuweisen, der ihn am ehesten fertigstellen wird. Auch sie bricht ab, wenn sie einen Auftrag nicht zuweisen kann, weil der beste Server bereits voll belegt ist.
  Der Zeitbedarf für einen Auftrag *A*, wenn er von Server *S* ausgeführt wird, wird folgendermaßen abgeschätzt: Zuerst müssen alle Aufträge in *S*'s Warteschlange bearbeitet werden. Man nimmt an, daß jeder soviel Zeit benötigt, wie der letzte Auftrag, den *S* bearbeitet hat. Dann muß *A* berechnet werden. Die vom Client angegebene Instruktionszahl wird durch die momentan verfügbare Prozessorleistung geteilt (siehe *Processor Speed*). Zuletzt müssen die Wartezeiten auf nicht-lokale Datensätze berücksichtigt werden. Für jeden fehlenden Satz wird die Zeit zuaddiert, die *S* beim letzten Heranschaffen des Satzes warten mußte. Dabei werden nur Änderungszugriffe betrachtet. Lesende Zugriffe gelten als kostenlos, um eine hohe Parallelität zu erreichen.
- *DLPS2*. Diese Strategie ist eine Abwandlung von *DLPS1*. Sie bewertet die Kosten für den Datentransport absichtlich schwächer, ignoriert aber die Kosten für Datenkopien bei Lesezugriffen nicht ganz.

Dieser Vorrat an Strategien ist weder umfassend noch das Optimum, was im Rahmen der hier vorgestellten Konzepte an Balancierungsmethoden möglich ist. Er soll nur zeigen, daß Unterschiede zwischen Strategien bestehen und eine strategische Ebene der Lastverteilung Nutzen bringen kann.

3.2 Realisierte Anwendungen

Bisher wurden drei verschiedenartige Anwendungen unter dem vorgestellten Lastbalancierungsmodell realisiert. Sie sind absichtlich aus verschiedenen Bereichen gewählt, die nicht zu den klassischen Datenbank-Anwendungen zählen. Dadurch wird ersichtlich, daß der in Abschnitt 1.2 erklärte Begriff 'datenintensive Anwendung' in der Praxis einen recht großes Anwendungsspektrum umfaßt. Es folgt die Beschreibung der verwendeten parallelen Algorithmen und der verteilten Datenstrukturen.

3.2.1 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 Serverklasse *GFind* auf mit einer Liste von Knoten, von denen aus weiter in Richtung Zielknoten gesucht werden muß. Zu Anfang ist diese Liste nur der Startknoten, im weiteren Verlauf besteht sie aus den Ergebnis-Knoten der *GFind* Aufrufe. Die Serverklasse *GFind* erhält als Eingabeparameter eine Liste von Knoten und gibt die Liste der Knoten zurück, die von dort aus in einem Schritt direkt erreichbar sind. Dabei verwaltet sie eine Liste der bisher erreichten Knoten samt den Kosten. Sie gibt nur diejenigen erreichten Knoten zurück, die bisher noch nicht oder nur teurer erreicht wurden. Abbildung 5 skizziert links die Aufteilung von Ergebnisknoten für Folgeaufträge. Die Startaufträge werden weitmöglichst so aufgeteilt, daß die Startpunkte in derselben oder in benachbarten Datenpartitionen liegen.

Der Graph ist in Form mehrerer Dateien abgespeichert. Eine Datei enthält jeweils alle Kanten, die von einem bestimmten Knotenbereich ausgehen (Abbildung 5 deutet rechts die Aufteilung des Graphen an). Jede Datei bildet einen Datensatz bezüglich der Lastbalancierung, ist also das Granulat der Verteilung und Synchronisation. Die temporäre Liste der bisher in der Suche erreichten Knoten wird als Datenfeld im Hauptspeicher gehalten. Auch dieses Feld ist für die Parallelarbeit nach Nummernbereichen der erreichten Knoten partitioniert.

Da der Graph während der Suche statisch ist, lohnt es sich, zum parallelen Suchen Kopien zu verteilen. Die Geschwindikeitssteigerung durch Parallelarbeit wird aber durch die Änderungsoperationen auf der gemeinsamen Liste der erreichten Knoten beschränkt. Neben der gleichmäßigen Ausnutzung der Prozessoren kann Lastbalancierung den Ablauf extrem beschleunigen, indem sie Aufträge an diejenigen Server vergibt, die den größten Teil der vermutlich benötigten Zeilen in der Liste lokal vorliegen haben.

Für die Messungen wurde ein Auftragsgranulat in der Größenordnung von 10 - 100 Sekunden Verarbeitungszeit (auf Workstations) gewählt. Die 20 Dateien des Graphen haben im Mittel eine Größe von 15 KByte. Die Liste der erreichten Knoten wurde - wegen der hohen Änderungsrate etwas feiner - in 50 Partitionen zu je 80 Byte aufgeteilt. Gesucht wurde auf einem Graphen mit 1000 Knoten und 50000 Kanten, der so generiert wurde, daß im Mittel 95% der Kanten zwischen

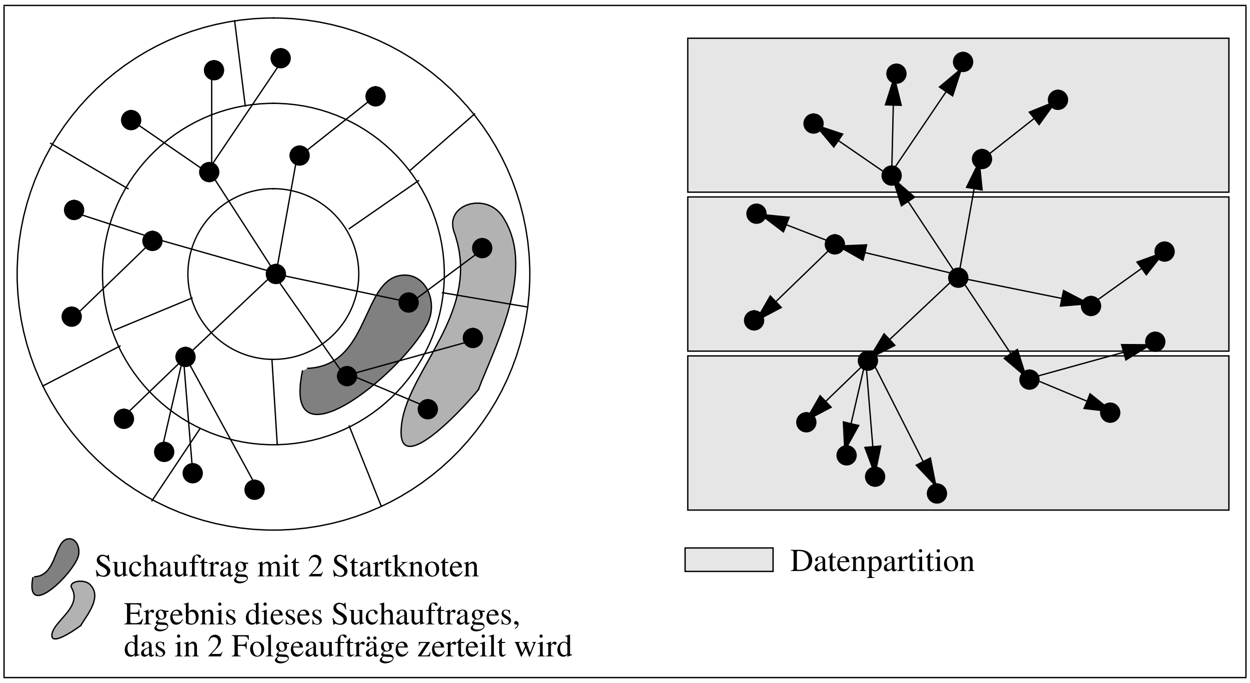

Abbildung 5: Aufteilung der Aufrufstartpunkte und der Graphdaten in der Wegesuche.

Knoten 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 beinhaltet Knoten aus vielen verschiedenen Partitionen), müssen die Folgeaufträge auf einer Vielzahl von Partitionen arbeiten oder das Granulat der Folgeaufträge muß sehr klein gemacht werden.

3.2.2 Flächensegmentierung zur Bilderkennung

Aufgabe der Flächensegmentierung ist es, ein gegebenes Punktrasterbild in eine Menge homogener Flächen (Polygone) zu konvertieren. Eine Fläche soll farblich beieinander liegende Punkte mit einem kleinen Anteil von Ausnahmen enthalten. Der verwendete Algorithmus [41] besteht aus vier Schritten (siehe auch Abbildung 6): 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). 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.

Split- und Merge-Operationen sind meist sehr feingranular (wenige Millisekunden). Daher werden sie, wo möglich, nach Bildsegmenten zusammengefaßt 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.

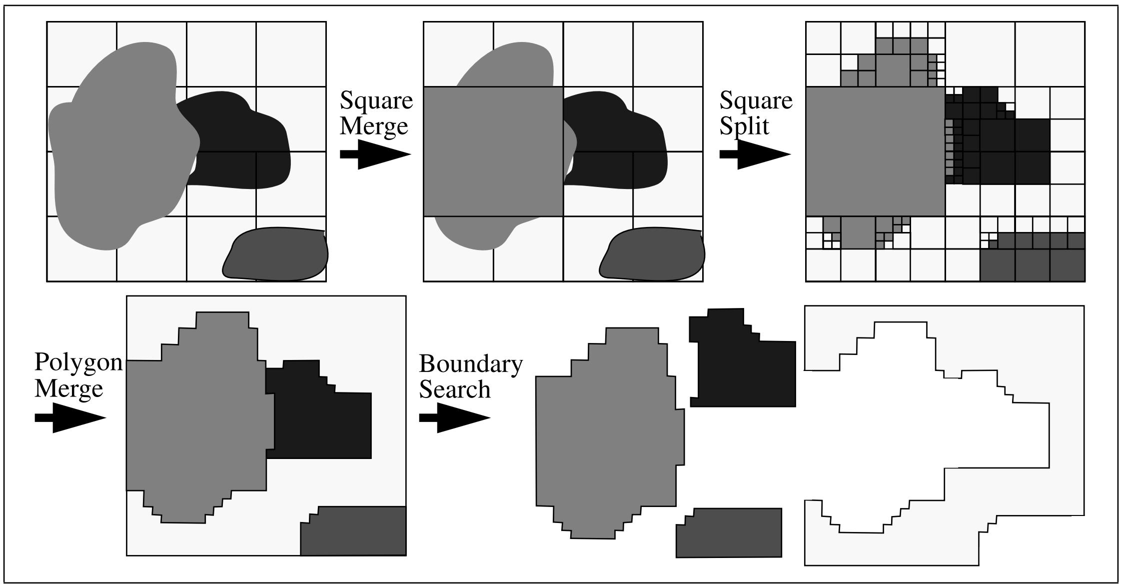

Abbildung 6: Schritte des Bildsegmentierungs-Algorithmus.

Das verursacht je nach Bildstruktur starke Datenkommunikation und schränkt die Parallelität ein. Dieselbe Beobachtung gilt auch für die anschließende Berechnung der Polygonränder.

Die hier vorgestellten Messungen basieren auf dem in Abbildung 7 links oben gezeigten Pixelmuster, der Luftaufnahme einer Landschaft. Rechts oben ist das Zwischenergebnis nach den ersten beiden Schritten, unten das Resultat (links die eingefärbten Polygone verkleinert, rechts nur die Umrandungen) gezeigt. Das Datenfeld der Bildpunkte besteht aus 900 Partitionen zu je 256 Byte, die Liste der ca. 3300 entstanden Flächen ist in 410 Blöcke der Größe 1400 Byte partitioniert. Die Bearbeitungszeit pro Auftrag liegt zwischen 0.1 und 20 Sekunden. Von der Gesamtrechenzeit entfallen etwa 15% auf die Phase Square-Merge und -Split, 75% auf Polygon-Merge und 10% auf die Phase Boundary-Search.

3.2.3 Verwaltung geometrischer Objekte mit Hilfe von R-Bäumen

Die dritte realisierte Anwendung führt geometrische Operationen auf Polygonen durch [1]. Polygone werden in Relationen abgespeichert; zu jeder Polygon-Relation wird zusätzlich ein R-Baum verwaltet, der inhaltsbezogene Zugriffe beschleunigt. Dies ist eine in Datenbank-basierten CAD-Systemen und geographischen Anwendungen übliche Struktur. Inhaltsbezogene Zugriffe sind Selektionen oder Verbundoperationen mit räumlichen Suchprädikaten: Überlappung / Enthaltensein / Höchstabstand zwischen geometrischen Objekten.

Sowohl die Polygone einer Relation als auch die Knoten des zugehörigen R-Baums sind partitioniert und auf verschiedene Dateien verteilt (siehe Abbildung 8; die Rechtecke sind jeweils Daten-Partitionen). Die Anwendung realisiert eine Mischung von Einfüge-, Selektions- 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-Baumteile bzw. durch Anlegen von Kopien effizient parallelisiert werden.

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

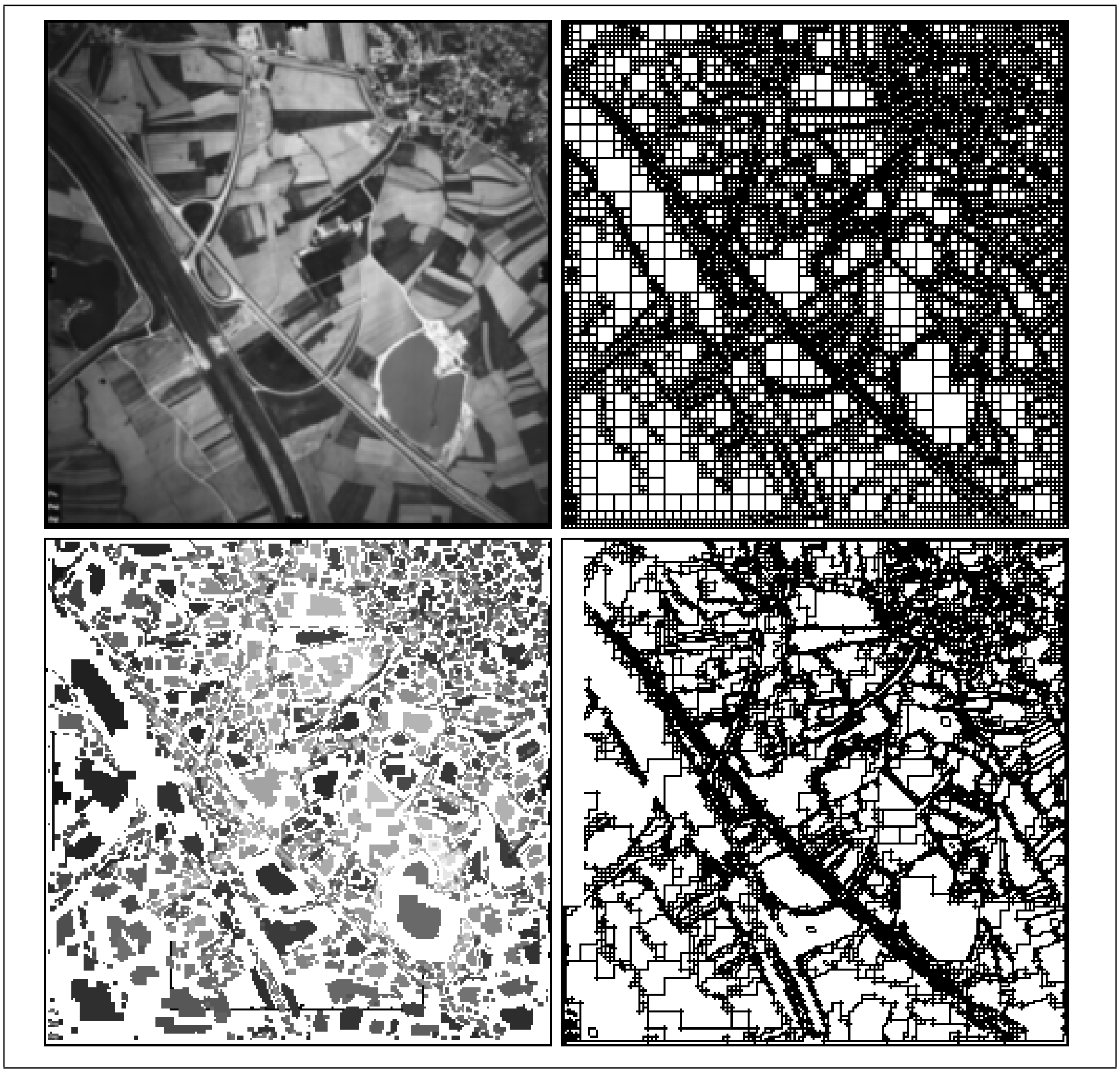

Abbildung 7: Zur Messung der Flächensegmentierung verwendetes Bild.

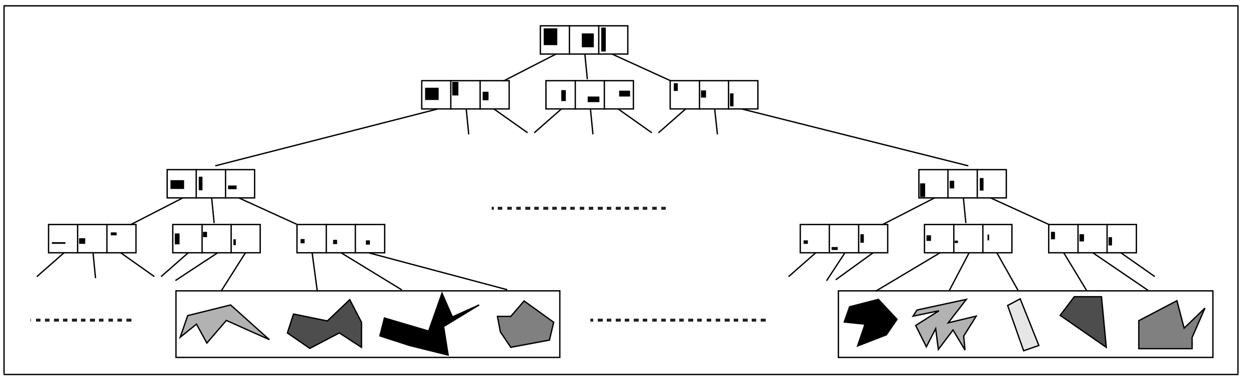

Abbildung 8: Verteilte Datenstrukturen zur Verwaltung der Polygone.

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 Selektionen und Verbunde (eine Selektion / ein Verbund wird hier nicht in sich parallelisiert) relativ große Operationen sind.

Für die Messungen werden insgesamt 3000 Polygone eingefügt und 600 Selektionen mit räumlichen Prädikaten durchgeführt (siehe auch Abbildung 13). Die Polygone sind in 325 Dateien zu je ca. 10 KByte partitioniert, die Zugriffstrukturen belegen 25 Dateien einer Größe um 1 KByte.

3.2.4 Weitere Anwendungen

Um das Spektrum der Awendungen zu erweitern, werden derzeit ein exakter Löser für lineare Gleichungssysteme, ein Szenario relationaler Datenbankoperationen sowie eine Spannungsberechnung in Körpern mit Methode der finiten Elemente auf die Lastbalancierungsumgebung portiert.

3.3 Messung und Beurteilung

3.3.1 Lastbalancierung zur Effizienzsteigerung einer einzelnen Anwendung

Alle in diesem Abschnitt vorgestellten Messergebnisse basieren auf dem in Abbildung 9 skizzierten System. Die Lastbalancierungskomponente sowie die Clients der Anwendungen wurden auf einem relativ langsamen Prozessor (A) angesiedelt, die Server jeweils auf die übrigen Prozessoren verteilt. Die Server der Wegesuche und der geometrischen Objektverwaltung arbeiten mit persistenten Daten und können daher nur auf die Rechner E .. H verteilt werden, die mit lokalen Platten ausgestattet sind.

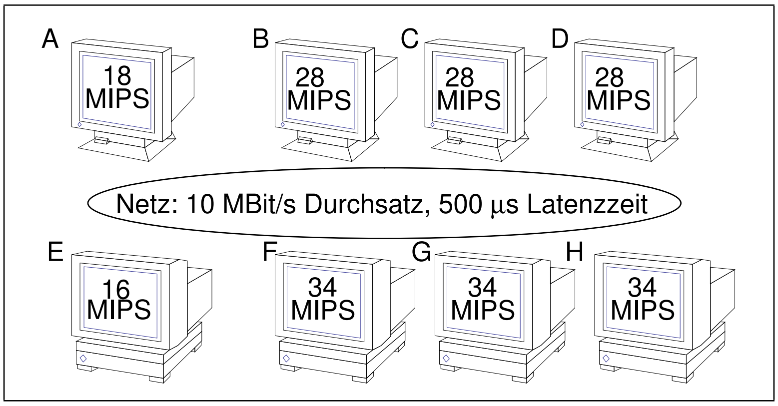

Abbildung 9: Die für die Messungen verwendete Konfiguration.

Abbildung 10 gibt einen Überblick des Potentials an Parallelität, das in den oben vorgestellten Algorithmen auf der Messumgebung enthalten ist. Man beachte dabei, daß diese Messungen alle unter derselben Lastverteilungsstrategie *First Free* mit der initialen Datenverteilung *Data Blokking* durchgeführt wurden (beschrieben in Abschnitt 3.1) und daher nicht das theoretische Parallelisierungspotental angeben. Die Werte sind jeweils Durchschnitte aus fünf identischen Meßläufen. Im Bild wurden die Meßreihen unter Verwendung eines, zweier, dreier ... Prozessoren als Balken aufgetragen. Sie sind auf der X-Achse nach der Gesamtrechenleistung der beteiligten Prozessoren positioniert.

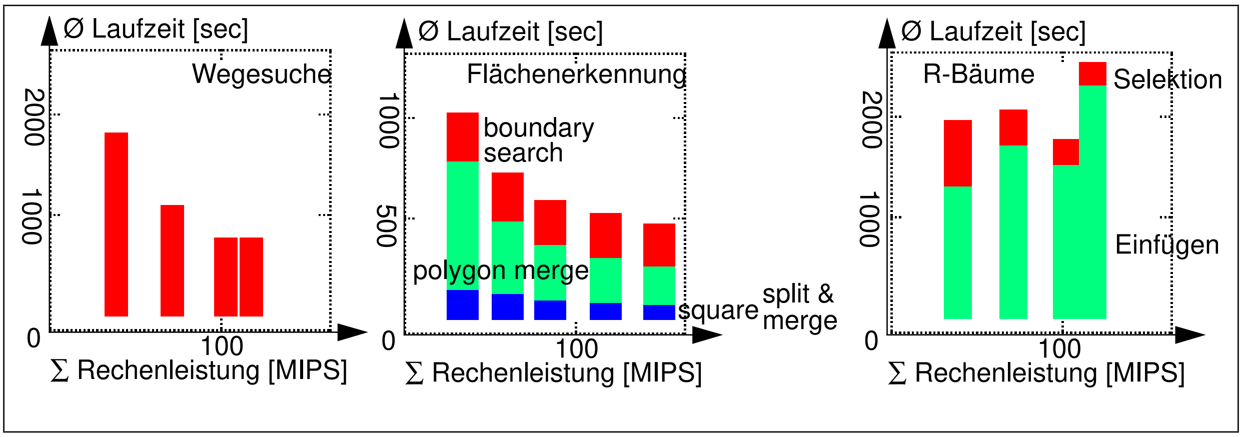

Abbildung 10: Speedup-Verhalten der Anwendungen bei einfacher Lastbalancierung.

Mit der relativ simplen Lastbalancierungsstrategie läßt sich die Wegesuche auf bis zu drei Prozessoren effizient parallelisieren. Darüber wird die Synchronisation der Änderungsoperationen auf der Erreichbarkeitsliste zu teuer. Die Flächenerkennung zeigt auch bei fünf Prozessoren noch Geschwindigkeitssteigerungen. Bemerkenswert ist, daß hauptsächlich die Merge-Phase beschleunigt wird, während die übrigen Phasen relativ konstant bleiben. Die R-Baum-Operationen lassen sich mit dem gewählten Operations-Mix insgesamt nicht gewinnbringend parallelisieren. Der Anteil der Einfügeoperationen ist so groß, daß die leichten Gewinne durch parallele Selektionen nicht genügend ausgleichen können.

Abbildung 11 vergleicht die Laufzeiten der parallelen Anwendungen unter Einsatz der Strategie *DLPS2* mir den obigen (*First Free*). Dies soll verdeutlichen, daß die Verwendung von unterschiedlichen Lastverteilungsstrategien signifikanten Einfluß auf die Laufzeiten hat.

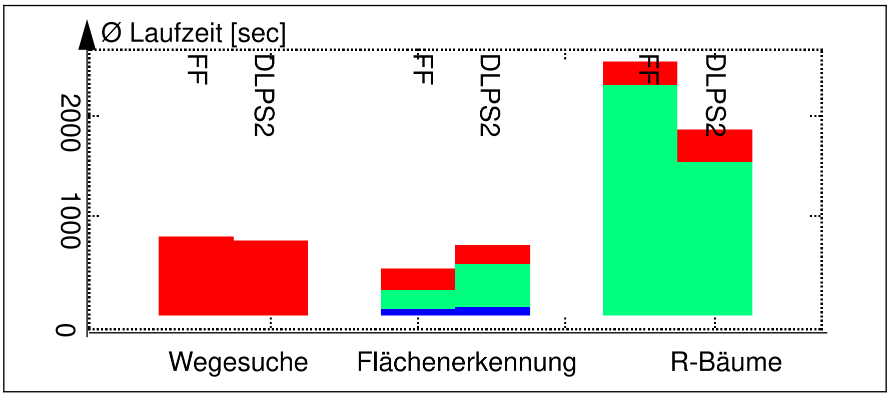

Abbildung 11: Vergleich der Strategien First Free und DLPS2.

Die Wegesuche läßt sich durch die Strategie DLPS2 auch noch auf vier Prozessoren effizient rechnen, denn durch die Berücksichtigung der Datenaffinitäten können mehr Updates der Erreichbarkeitsliste lokal abgewickelt werden. Abbildung 12 zeigt links das Laufzeitverhalten der Server. Über die Zeitachse sind die Zeitanteile aufgetragen, die die Server zum tatsächlichen Rechnen, beim Warten auf und Austauschen von Daten bzw. im unbeschäftigten Zustand verbracht haben. Die Flächenerkennung lief langsamer ab als unter *First Free*, weil in der Merge-Phase nicht alle Server eingesetzt wurden (in der Mitte von Abbildung 12 ist das Laufzeitverhalten gezeigt): Die Datenmigrationskosten wurden teilweise überschätzt bzw. der mögliche Gewinn durch das Anlegen von Kopien unterschätzt. Die R-Baum-Operationen wurden deutlich effizienter, da vor allem beim Einfügen von Polygonen verstärkt auf lokal vorliegenden Teilen der Bäume gearbeitet werden konnte (Abbildung 12 zeigt rechts das Laufzeitverhalten der Server).

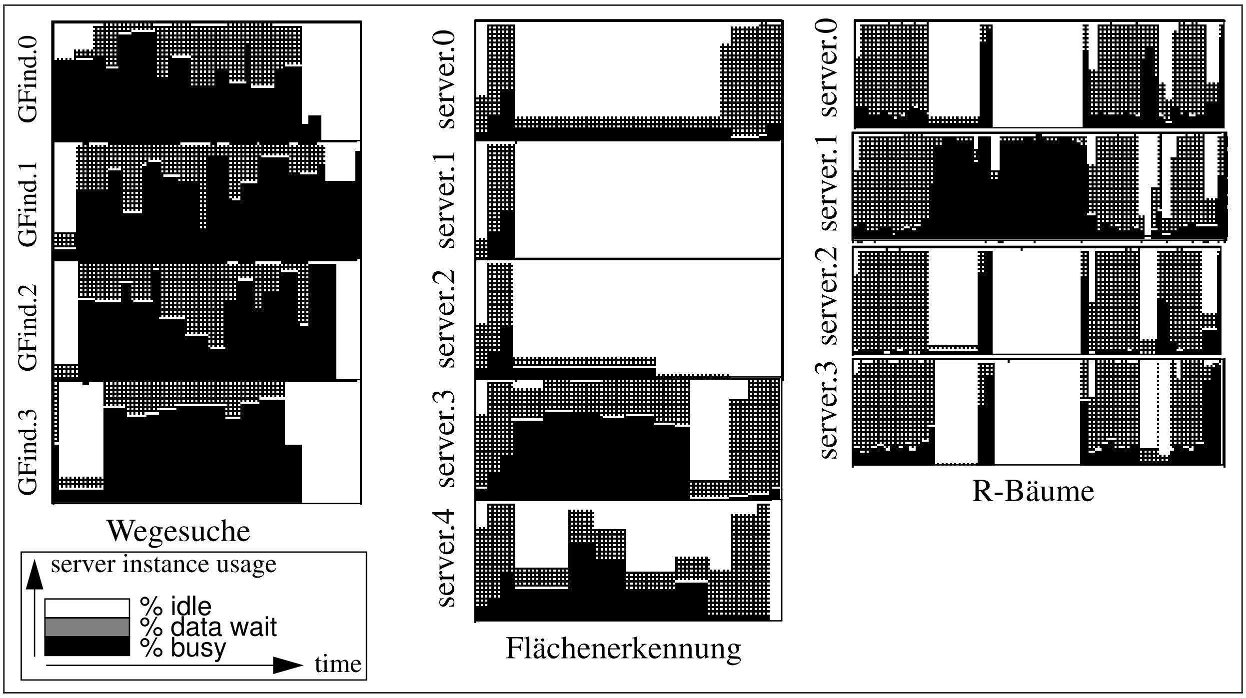

Abbildung 12: Arbeitsprofile der Server unter der Strategie DPLS2.

3.3.2 Lastbalancierung zur Durchsatzsteigerung bei homogenen Lasten

Um den Einfluß der Lastbalancierung im Mehrbenutzerbetrieb zu analysieren, zeigt Abbildung 14 Laufzeiten der Anwendungen, die jeweils mehrfach parallel abliefen. Die Wegesuche findet Pfade zwischen drei verschiedenen Start- und Zielpunkten (auf demselben Graph). Die Flächenerkennung analysiert drei verschiedene Bilder. Auf den R-Bäumen werden Einfüge- und Selektionsoperationen konkurrierend durchgeführt; im Unterschied zu Abschnitt 3.3.1 werden auch alle Selektions- und Einfügephasen zueinander parallel bearbeitet (siehe Abbildung 13). Um das Profil realistischer zu gestalten, wird die maximale Parallelität - wie auch in Abschnitt 3.3.1 - vom Client auf 40 begrenzt.

Der Speedup ist im Mehrbenutzerbetrieb weniger interessant, da mehrere unabhängige Anwendungen problemlos auf das System verteilt werden können. Hier sind jedoch auch die einzelnen Anwendungen in sich parallelisiert. Die Lastbalancierung muß also dieselben Probleme wie in Abschnitt 3.3.1 erfassen; als Zusatz kommen hier die höhere Systemlast und häufigere Lastbalancierungsereignisse.

Die Wegesuche zeigt bis auf Ausnahmen bessere Laufzeiten mit zunehmender Komplexität der Lastbalancierungsstrategie, während bei der Flächenerkennung die einzelnen Phasen sehr unterschiedlich auf die Strategien reagieren. Die erste Phase der Flächenerkennung läuft unter *Data Locality* und *DLPS1* deshalb lange, weil die Lastbalancierung für die vielen kurzen Aufträge zuviel Aufwand treibt, da diese viele benutzte Datenbereiche angeben. Bei den R-Baum-Operationen können, wie schon im Einbenutzerbetrieb beobachtet, durch verfeinerte Strategien vor

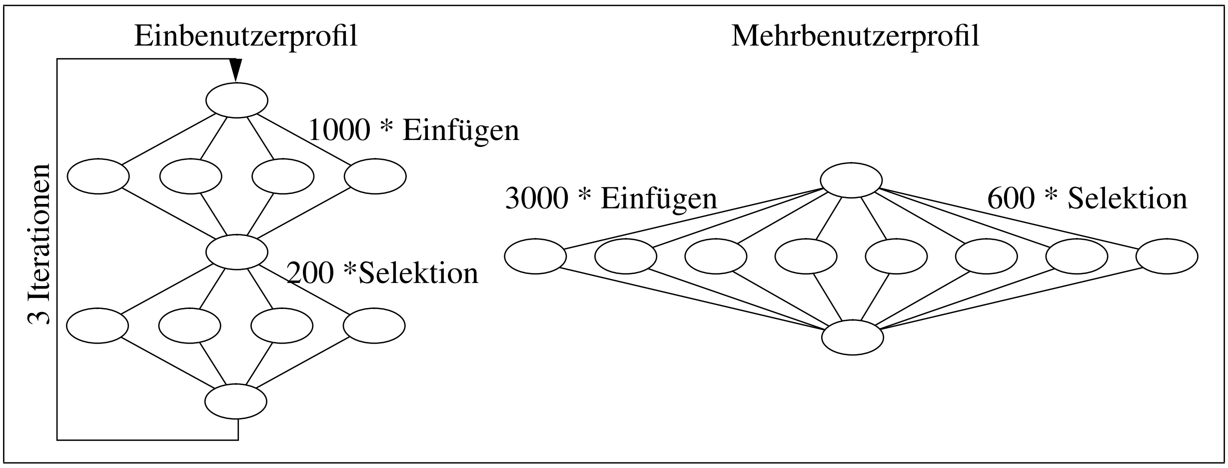

Abbildung 13: Ablauf der Operationen auf R-Bäumen für die Messungen.

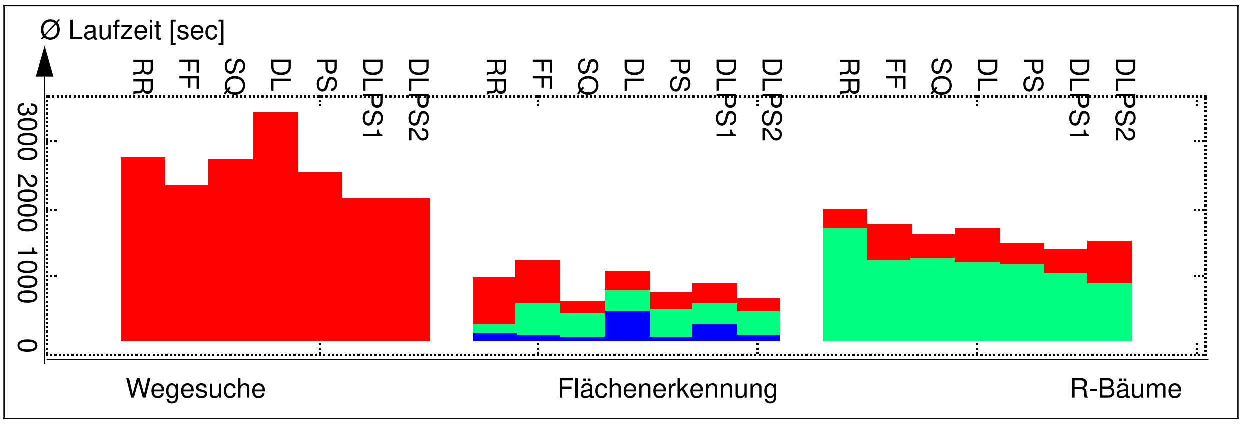

Abbildung 14: Laufzeitverhalten bei homogenen Lasten unter verschiedenen Strategien.

allem die kritischen Einfügeoperationen optimiert verteilt werden, was teilweise zu Lasten der Retrieval-Operationen geht.

Die Strategie *Data Locality* schneidet sichtbar schlecht ab, weil die alleinige Betrachtung der Datenaffinitäten dazu führt, daß einige Server mehr und mehr Datensätze auf sich konzentrieren und damit zunehmend viele Aufträge erhalten, während andere leerlaufen. Insbesondere bei starker Last im Mehrbenutzerbetrieb ist es für den Lastausgleich wichtig, unabhängige Aufträge zu verstreuen.

3.3.3 Lastbalancierung zur Durchsatzsteigerung bei Mischlasten

Als letzte Meßreihe beobachten wir ein heterogenes Gemisch von Lasten auf dem System unter verschiedenen Balancierungsstrategien. Dazu werden die drei Anwendungen konkurrierend abgewickelt. Als Rechnernetz wurden für die Server die Knoten E, F, G und H gewählt. Man beachte, daß nun pro Knoten mehrere Server, nämlich einer je Klasse, zur Verfügung stehen. Abbildung 15 gibt die Ausführungszeiten wieder. Alle Wegesuchen sind als letzte beendet, während es von der angewandten Strategie abhängt, ob die Bilderkennungsberechnungen oder die R-Baum-Operationen zuerst abgeschlossen sind. Im Bild werden die Abschlußzeiten der Anwendungstypen nebeneinander aufgezeichnet. Sie liefen aber tatsächlich nicht nacheinander, sondern vollständig parallel zueinander ab. Neben der höheren Last und häufigeren Balancierungsentscheidungen kommt hier das Problem hinzu, daß eine Strategie nicht auf alle Anwendungsprofile zugleich hin optimiert werden kann.

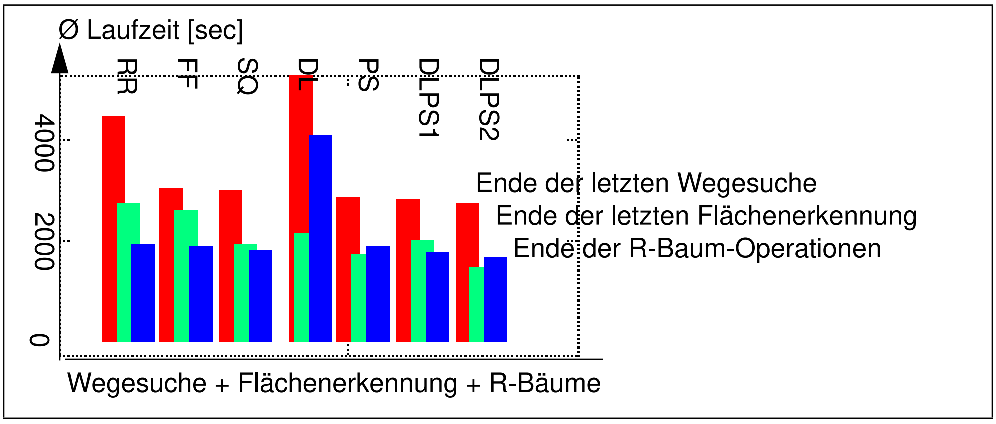

Abbildung 15: Laufzeitverhalten bei Mischlasten unter verschiedenen Strategien.

Insgesamt sind im Vergleich zur trivialen Balancierung (*Round Robin*) deutliche Leistungssteigerungen durch die meisten Strategien erkennbar. Die Lastbalancierung im HiCon-Modell wirkt sich also auch bei hoher, heterogener Systembelastung nicht störend aus, sondern ist in der Lage, den Gesamtdurchsatz zu erhöhen. Die detaillierten Ablaufprofile der einzelnen Anwendungen sollen hier nicht analysiert werden. Sie führen aber insgesamt zu dem Schluß, daß bei sehr heterogenen Lastprofilen das Wechseln zwischen kompletten Strategien wenig Erfolg verspricht. Vielmehr sollten dann Strategien eingesetzt werden, die jeden einzelnen Auftrag nach einer groben Einordnung passend behandeln. Beispielsweise sollten kurze Aufträge beliebig verteilt werden, kurze Aufträge mit hohem Anteil an Datenänderungen rein nach Datenaffinität und lange Aufträge nach gleichmäßiger Prozessorauslastung. Diese Beobachtung deckt sich mit dem in Abschnitt 2.6 vorgestellten Konzept, daß sehr heterogene Profile entsprechend komplexerer Strategien bedürfen, während homogene Phasen gut durch spezielle, einfache Strategien zu balancieren sind.

4 Zusammenfassung und Ausblick

Dieser Artikel gab eine Einführung in die Problematik der Lastbalancierung, gefolgt von einer Vorstellung der Konzepte, die im HiCon-Modell entwickelt und untersucht werden. Der Artikel will vorrangig die vorgestellten Konzepte zur dynamischen Lastbalancierung plausibel machen.

Da der Großteil der realisierten und simulierten Konzepte in Forschungsprojekten völlig dezentrale Verfahren verwendet, wurde hier gezeigt, daß es durchaus sinnvoll ist, Teilsysteme durch zentrale Balancierungsverfahren zu verwalten. Es wurde gezeigt, daß verschiedene Ressourcen, nicht nur die Systemgröße ‘CPU queue length’, für Balancierungsentscheidungen relevant sind. Die Konzepte der zentralen Balancierungsstrategien unter Betrachtung verschiedener Ressourcen wurden mithilfe von Messungen verschiedener Anwendungen auf realen Systemen evaluiert. Die Zweckmäßigkeit dezentraler Erweiterungen und adaptiver Strategiewechsel wurde lediglich plausibel gemacht; genauere Auswertungen sind momentan in Arbeit.

5 Literaturverzeichnis

1. Aldinger, K.: Lastbalancierte Verwaltung geometrischer Objekte mit R-Bäumen, Diplomarbeit Nr. 1040, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1993)

2. Baumgartner, K., Wah, B.: A Global Load Balancing Strategy for a Distributed Computer System, Workshop on the Future Trends of Distributed Computing Systems in the 1990's (1988)

3. Becker, W.: Lastbalancierung in heterogenen Client-Server Architekturen, Fakultätsbericht 1992 /1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1992)

4. Becker, W.: Globale dynamische Lastbalancierung in datenintensiven Anwendungen, Fakultätsbericht 1993 /1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1993)

5. Becker, W., Pollak, R.: Efficiency of Server Task Queueing for Dynamic Load Balancing, eingereicht zur Veröffentlichung (1994)

6. Becker, W., Waldmann, G.: Exploiting Inter Task Dependencies for Dynamic Load Balancing, eingereicht zur Veröffentlichung (1994)

7. Berger, M., Bokhari, S.: A Partitioning Strategy for Nonuniform Problems on Multiprocessors, IEEE Transactions on Computers, Vol. 36, No. 5 (1987)

8. Blazewicz, J., Drabowski, M., Weglarz, J.: Scheduling Multiprozessor Tasks to Minimize Schedule Length, IEEE Transactions on Computers, Vol. 35, No. 5 (1986)

9. Bonomi, F, Kumar, A.: Adaptive Load Balancing in a Nonhomogeneous Multiserver System with a Central Job Scheduler, IEEE Transactions on Computers, Vol. 39, No. 10 (1990)

10. Bowen, N., Nikolaou, C., Ghafoor, A.: On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems, IEEE Transactions on Computers, Vol. 41, No. 3 (1992)

11. Bruno, J.: On Scheduling Tasks with Exponential Service Times and In-Tree Precedence Constraints, Acta Inf. 22 (1985)

12. Cap, C., Strumpen, V.: The PARFORM - A High Performance Platform for Parallel Computing in a Distributed Workstation Environment, Technical Report, Institut für Informatik, Universität Zürich (1992)

13. Casavant, T. , Kuhl, J.: A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transactions on Software Engineering, Vol. 14, No. 2, 1988.

14. Chandy, K., Reynolds, P.: Scheduling Partially Ordered Tasks with Probabilistic Execution Times, Proceedings Operating System Principles, Operating Systems Review, Vol. 9, No. 5 (1975)

15. Ciciani, B., Dias, D., Yu, P.: Load Sharing in Hybrid Distributed - Centralized Database Systems, Proceedings Distributed Computing Systems (1988)

16. Copeland, G., Alexander, W., Boughter, E., Keller, T.: Data Placement in Bubba, Proceedings SIGMOD (1988)

17. Cybenko, G.: Dynamic Load Balancing for Distributed Memory Multiprocessors, Journal of Parallel and Distributed Computing, No. 7 (1989)

18. Douglis, F., Ousterhout, J.: Transparent Process Migration: Design Alternatives and the Sprite Implementation, Software-Practice and Experience, Vol. 21, No. 8 (1991)

19. Eager, D., Lazowska, E., Zahorjan, J.: Adaptive Load Sharing in Homogeneous Distributed Systems, IEEE Transactions on Software Engineering, Vol. 12, No. 5 (1986)

20. Eager, D., Lazowska, E., Zahorjan, J.: The Limited Performance Benefits of Migrating Active Processes for Load Sharing, ACM SIGMETRICS, Performance Evaluation Review (1988)

21. Efe, K., Groselj, B.: Minimizing Control Overheads in Adaptive Load Sharing, Proceedings 9th International Conference on Distributed Computing Systems (1989)

22. Ezzat, A.: Load Balancing in NEST: A Network of Workstations, Proceedings Fall Joint Computer Conference, Dallas, Texas (1986)

23. Ferguson, D., Yemini, Y., Nikolaou, C.: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems, Proceedings Distributed Computing Systems (1988)

24. Gavish, B., Sheng, O.: Dynamic File Migration in Distributed Computer Systems, Communications of the ACM, Vol. 33, No. 2 (1990)

25. He, X.: Eine Übersicht über die Lastverteilung in verteilten Systemen, Bericht 190/89, Universität Kaiserslautern, Fachbereich Informatik (1989)

26. Hosseini, S., Litow, B., Malkawi, M., Mc Pherson, J., Vairvan, K.: Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm, Journal of Parallel and Distributed Computing, No. 6 (1990)

27. Hsu, C., Liu, J.: Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems, Proceedings Distributed Computing Systems (1986)

28. Huang, Y., Wolfson, O.: A Competitive Dynamic Data Replication Algorithm, Proc. Data Engineering (1993)

29. Iqbal, M., Saltz, J., Bokhari, S.: A Comparative Analysis of Static and Dynamic Load Balancing Strategies, Proceedings Parallel Processing (1986)

30. Kale, L.: Comparing the Performance of two Dynamic Load Distribution Methods, Proceedings Parallel Processing (1988)

31. Kim, C., Kameda, H.: An Algorithm for Optimal Static Load Balancing in Distributed Computer Systems, IEEE Transactions on Computers, Vol. 41, No. 3 (1992)

32. Krueger, P., Livny, M: A Comparison of Preemptive and Non-Preemptive Load Distributing, Proceedings Distributed Computing (1988)

33. Kunz, T.: The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, IEEE Transactions on Software Engineering, Vol. 17, No. 7 (1991)

34. Li, K., Cheng, K.: Static Job Scheduling in Partitionable Mesh Connected Systems, Journal of Parallel and Distributed Computing, No. 10, October 1990.

35. Lin, F., Keller, R.: The Gradient Model Load Balancing Method, IEEE Transactions on Software Engineering, Vol. 13, No. 1 (1987)

36. Lin, H., Raghavendra, C.: A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC), IEEE Transactions on Software Engineering, Vol. 18, No. 2 (1992)

37. Lo, V.: Algorithms for Static Task Assignment and Symmetric Contraction in Distributed Computing Systems, Proceedings Parallel Processing (1988)

38. Martel, C.: A Parallel Algorithm for Preemptive Scheduling of Uniform Machines, Journal of Parallel and Distributed Computing, No. 5 (1988)

39. Mirchandaney, R., Towsley, D., Stankovic, J.: Adaptive Load Sharing in Heterogeneous Systems, Proceedings Distributed Computing Systems (1989)

40. Osser, W.: Automatic Process Selection for Load Balancing, Master Thesis, University of California, Santa Cruz (1992)

41. Pollak, R.: Lastbalancierte parallele Flächenerkennung, Diplomarbeit Nr. 974, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1993)

42. Rahm, E.: Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen, Angewandte Informatik 4/86 (1986)

43. Ross, K., Yao, D.: Optimal Load Balancing and Scheduling in a Distributed Computer System, Journal of the ACM, Vol. 38, No. 3 (1991)

44. Schabernack, J.: Lastenausgleichsverfahren in verteilten Systemen - Überblick und Klassifikation, Informationstechnik, Vol. 34, No. 5 (1992.)

45. Shen, S.: Cooperative Distributed Dynamic Load Balancing, Acta Informatica, Vol. 25 (1988)

46. Smith, R.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers, Vol. 29, No. 12 (1980)

47. Tantawi, A., Towsley, D.: Optimal Static Load Balancing in Distributed Computer Systems, Journal of the ACM, Vol. 32, No. 2 (1985)

48. Theimer, M., Lantz, K.: Finding Idle Machines in a Workstation-Based Distributed System, IEEE Transactions on Software Engineering, Vol. 15, No. 11 (1989) bzw. Proc. 8th Conf. on Distributed Computing Systems (1988)

49. van Tilborg, A., Wittie, L.: Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers, IEEE Transaction on Computers, Vol. 33, No. 9 (1984)

50. Varadarajan, R., Ma, E.: An Approximate Load Balancing Model with Resource Migration in Distributed Systems, Proceedings Parallel Processing (1988)

51. Waldmann, G.: Dynamische Lastbalancierung unter Ausnutzung von Reihenfolgebeziehungen, Studienarbeit Nr. 1280, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1993)

52. Yu, P., Leff, A., Lee, Y.: On Robust Transaction Routing and Load Sharing, ACM Transactions on Database Systems, Vol. 16, No. 3 (1991)

53. Zedelmayr, J.: Hierarchische dynamische Lastbalancierung in datenintensiven Anwendungen, Studienarbeit Nr. 1293, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1994)

54. Zhou, S., Ferrari, D.: An Experimental Study of Load Balancing Performance, Report No. 86.8, Computer Science Division, University of California, Berkeley (1987)

55. Zhou, S., Zheng, X., Wang, J., Delisle, P.: Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems, Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Canada (1992)