Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen

Wolfgang Becker

Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR)

Abteilung Anwendersoftware

Fakultät für Informatik

Universität Stuttgart

Breitwiesenstr. 20-22, D 70565 Stuttgart

Telefon: 0711 7816 402
Telefax: 0711 7816 424
email: Wolfgang.Becker@informatik.uni-stuttgart.de

Erscheint in „Informatik Forschung und Entwicklung“, Springer Verlag, 1994.

Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen

Wolfgang Becker, Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR), Universität Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart

Eingereicht am 30. Dezember 1993 / Angenommen am 8. Juli 1994

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 werden für schwer vorabsehbare Größen und Effekte durch Beobachtung des realen Verhaltens Gewichtungen für Balancierungsentscheidungen dynamisch eingeregelt, und die Effizienz der Lastbalancierung wird durch angepaßte Reduktion des Zusatzaufwandes gewährleistet.

Der Artikel stellt die Grundideen vor und validiert die Konzepte durch Messung verschiedener Anwendungen auf einem heterogenen Netz von Arbeitsplatzrechnern. Dazu werden mehrere verschiedene parallelisierte Anwendungen konkurrierend auf dem Rechnersystem balanciert.

Schlüsselwörter: dynamische Lastbalancierung, datenintensive Anwendungen, parallele Anwendungen, Rechnernetze.

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 parameter values for load balancing decisions can be adjusted dynamically according to observations of the real behavior for quantities and effects that are hardly foreseeable. Efficiency of dynamic load balancing is guaranteed by situation-specific reduction of additional overhead.

This article introduces the main ideas and validates the concepts through measurement of different applications on a heterogeneous workstation network. Therefore several different parallelized Applications are balanced concurrently within the computing system.

Key words: dynamic load balancing, data intensive applications, parallel applications, workstation networks**.**

CR Subject Classifications: C.2.4, C.4, D.4.8

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 möglich 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 die 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 Arbeitsplatzrechnern 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, stati-

sche 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 [12, 24, 42]. Auf dem Gebiet der statischen Zuweisung von parallelisierten Anwendungen finden sich zahlreiche Ergebnisse: In [6, 9, 36] werden Aufträge zur Minimierung der Kommunikationskosten auf Prozessoren gruppiert; [7, 37] verteilen unabhängige Aufträge so, daß die Gesamtlaufzeit minimal wird, bzw. daß die Knotenauslastungen gleich sind [25, 33]. Zusätzlich werden in [10, 13] Reihenfolgebeziehungen zwischen Aufträgen berücksichtigt. Bei [30, 45] werden Transferraten für gegebenes statistisches Lastaufkommen ermittelt, um die mittlere Antwortzeit einzelner Aufträge zu minimieren.

In [28, 41] werden statische Lastverteilungsverfahren um dynamische Anpassungen erweitert. Zentral gesteuerte dynamische Lastbalancierungstechniken finden sich in [8, 35, 46, 47]. Viele Ansätze wurden zur dezentralen dynamischen Auftragszuweisung veröffentlicht; man unterscheidet, ob überlastete Knoten neue Aufträge weitergeben [18, 26, 29, 34, 38, 50], freie Knoten Aufträge anfordern [18, 38], oder explizite Absprachen zwischen benachbarten Knoten stattfindet [1, 22, 43, 44]. Die Projekte [16, 17, 19, 21, 31, 39, 51] beschäftigen sich mit der Migration laufender Prozesse zum Lastausgleich bzw. modifizieren die Auftragsgrößen zur Laufzeit [11]. Dynamische Adaption der Lastbalancierungsstrategien, d.h. Anpassung von Schwellwerten und Meßintervallen, wird in [20, 32] untersucht. Im Bereich der datenintensiven Anwendungen finden sich Ansätze in [14, 15, 23, 27, 40, 48, 49].

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 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.

Bild1>

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

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

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.

Bild2>

2 Der Lastbalancierungsansatz im HiCon-Modell

Die Untersuchungen im HiCon-Modell [2, 3] basieren auf dem Konzept einer Lastbalancierungsumgebung, die es ermöglicht, Effekte verschiedener Methoden zur Lastverteilung anhand verschiedenartiger Anwendungen auf einem heterogenen Netz von Arbeitsplatzrechnern 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 Verarbeitungsmodell 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 Struktur und Ablauf 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 Ebene (Bewertung und Auftragszuweisung) und eine Ebene, die dynamische Adaption der Lastverteilung ermöglicht (Justierung von Entscheidungsparametern). 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 (Abschnitt 2.3).

Bild3>

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 aufgelistet:

- Entstehung neuer Aufträge. Clients schicken Aufrufe ab, die durch einen Server einer bestimmten Klasse zu bearbeiten sind. Dabei können sie Vermutungen über die Größe des Auftrages und über Datenreferenzen mitgeben.
- Vorankündigung von Auftragsgruppen. Clients geben eine Abschätzung der Struktur und der Größen einer anstehenden Gruppe von Aufträgen.
- Änderungen im Arbeitszustand von Servern. Wie unten erläutert, haben Server lokale Auftragswarteschlangen. Jedesmal, wenn sie einen Auftrag abgeschlossen haben, geben sie der Lastbalancierung mit dem Ergebnis Informationen über den Bearbeitungsverlauf mit (Laufzeit, Datenwartezeiten etc.).
- Änderungen in der aktuellen Auslastung von Ressourcen. Als Ressourcen werden hier Prozessoren, Hauptspeicher, Platten und Netzverbindungen angesehen.
- 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 (Abschnitt 2.3) benachrichtigt diese ihre unmittelbaren Nachbarn.

Je nach momentaner Einstellung der Lastbalancierungsstrategie werden manche dieser Ereignisse berücksichtigt, andere ignoriert; denn das Ziel muß darin bestehen, stets eine möglichst simple, angepaßte Strategie zu verwenden. Im Regelfall werden daher nur wenige Ereignisse Balancierungs-Aktionen nach sich ziehen.

Bei Ankündigung einer Auftragsgruppe kann der Client die vermuteten Größen der Aufträge und die Reihenfolgebeziehungen zwischen ihnen angeben. Die Aufträge der Gruppe sind hier lediglich Namen. Die Lastbalancierung kann nun durch verschiedene Verfahren [5] Prioritäten zu den Aufträgen berechnen. Derzeit sind folgende Alternativen verfügbar:

- Keine Prioritätsberechnung. Vorankündigungen werden ignoriert.
- Die Priorität jedes Auftrags entspricht seiner Größe.
- Die Priorität ist proportional zur Anzahl der Aufträge auf dem längsten Folgepfad des Auftrags.
- Die Priorität ist proportional zur Summe der Auftragsgrößen des längsten Folgepfades.

Letzteres Verfahren entspricht dem Wertungskriterium des ‘Exit Path’-Scheduling bei statischer Lastverteilung. Als Verfeinerung können hier Aufträge, die eine hohe Parallelität an Folgeaufträgen aufweisen, stärker gewichtet werden. Im HiCon-Modell werden auf eine Gruppenankündigung hin nur die Prioritäten zu den Auftragsnamen gespeichert; es wird noch keine Zuweisung oder Reservierung vorgenommen. Die Priorität von Aufträgen wird in Instruktionen gemessen und soll die Größe des Auftrags samt den Größen des Folgepfades von Aufträgen darstellen.

Aufträge der Clients (Serverklassen-Aufrufe) werden zunächst in eine globale Warteschlange eingereiht. Dabei werden die neuen Aufträge normalerweise hintenan gestellt. Waren sie aber vorangekündigt, so können sie ihrer zugeordneten Priorität entsprechend eingereiht werden. Unangemeldete (Einzel-) Aufträge erhalten dann als Priorität ihre geschätzte Auftragsgröße. Schließlich wachsen die Prioritäten der Aufträge mit der Zeit, die sie in der zentralen Warteschlange stehen. Wird daher ein neuer Auftrag einsortiert, dann wird die Priorität der anderen Aufträge beim Vergleich um die Anzahl von Instruktionen erhöht, die sie in der Zwischenzeit in der Warteschlange ‘verpaßt’ haben (hier wird eine mittlere Systemleistung zugrundegelegt).

Bild4>

Auf jedes der oben genannten Ereignisse hin kann die Lastbalancierung versuchen, Aufträge an Server zuzuweisen. Die Zuweisung kann zu beliebigen Zeitpunkten erfolgen, denn die Server haben lokale Auftragswarteschlangen, die sie Stück für Stück abarbeiten (Abbildung 4). Folgender Algorithmus untersucht, ob momentan Aufträge an Nachbarsysteme (Abschnitt 2.3) oder an Server zugewiesen werden können:

- Zuerst wird die Gesamtlast des Systems mit den Nachbarsystemen verglichen. Als Lastmaß wird die mittlere Anzahl aktiver Server im System durch die Gesamtprozessorleistung dividiert. Ist das Nachbarsystem signifikant weniger belastet, so weist die Lastbalancierung diesem Nachbarsystem einige Aufträge zu, so daß die Auftragswarteschlangen gleich sind. Als Maß gilt die Länge der zentralen Warteschlange der Serverklasse zuzüglich der mittleren Schlangenlänge an den Servern. Man beachte, daß hier die neusten Aufträge (bzw. die Aufträge mit

geringster Priorität) an Nachbarsysteme vergeben werden. Sie werden ja dort nicht sofort bearbeitet, sondern erneut in eine zentrale Warteschlange eingereiht. Wann ein Nachbarsystem signifikant weniger belastet ist, wird durch die strategische Lastbalancierung festgesetzt.

Am Rande sei bemerkt, daß im HiCon-Modell auch implizite Verfahren evaluiert werden. Dort werden Nachbarsysteme einfach als besondere Server betrachtet und dieser Teil des Zuweisungsalgorithmus entfällt. Wir beschränken uns in diesem Artikel auf obiges Verfahren.

- Nun werden so viele Aufträge an Server zugewiesen, bis eine bestimmte mittlere Warteschlangenlänge bei den Servern erreicht ist oder keine Aufträge für die Klasse mehr anstehen. Diese Grenze für die mittlere Warteschlangenlänge wird durch die adaptive Komponente der Lastbalancierung geregelt (Abschnitt 2.4). Ein weiterer Parameter dieser Art beschränkt die Anzahl der Zuweisungen pro Lastbalancierungsereignis.
  Dazu wird immer für den ältesten Auftrag (bzw. den mit höchster Priorität) der bestgeeignete Server ausgewählt (wieviele Server jeweils betrachtet werden, wird durch eine weitere strategische Variable bestimmt). Ein Server wird danach bewertet, bis wann er den Auftrag vermutlich abschließen könnte, wenn er ihn jetzt zugewiesen bekäme. Dazu können die Rechenzeit für den Auftrag, die Wartezeit bis zur Beendigung der bereits wartenden Aufträge und die zu erwartenden Verzögerungen durch nicht lokal vorhandene Datensätze betrachtet werden. Welche Faktoren betrachtet werden, wird durch dynamische Adaption den Lastprofilen und der Lastsituation angepaßt.

  Grundlage der Abschätzung bildet die vom Prozessor des Servers momentan zu erwartende Rechenleistung. Dazu wird die theoretische Leistung des Prozessors - die MIPS-Zahl gemäß eines Benchmarks - durch die Anzahl der sonstigen aktiven Server auf dem Prozessor (plus eins) geteilt. Bei Multi-Prozessoren überläßt man das Scheduling innerhalb der Prozessoren besser dem Betriebssystem. Im HiCon-Modell wird daher ein Shared-Memory-Knoten mit vier Prozessoren wie ein einziger Prozessor betrachtet, auf dem aber die Fremdlast dreier anderer Server für den neuen Auftrag noch keine Leistungsreduktion bringt. Anstelle der Anzahl aktiver Server kann natürlich auch die sogenannte ‘cpu run queue length’, evtl. mit Berücksichtigung der Hauptspeicherüberlastung, betrachtet werden.

  So kann die Antwortzeit für den Auftrag auf diesem Server geschätzt werden. Man teilt die Auftragsgröße samt der Summe der Auftragsgrößen, die bereits beim Server warten bzw. bearbeitet werden, durch die momentane Prozessorleistung. Die Zeit seit dem letzten Ergebnis des Servers wird von der Antwortzeit abgezogen, da inzwischen der nächste Auftrag begonnen wurde.

  Die Wartezeit auf benötigte Daten, die sich der Server vermutlich von anderen Servern schikken lassen muß, wird wie folgt abgeschätzt. Pro angekündigtem Datensatz, der momentan beim Server nicht verfügbar ist, wird die Zeit addiert, die in letzter Zeit für einen Transport dieses Datensatzes benötigt wurde. Die adaptive Lastbalancierung führt darüber Statistik und regelt die Gewichtung durch exponentielle Glättung. Für vermutete Lesezugriffe genügt es, wenn der Server eine Kopie der Daten besitzt, für Schreibzugriffe braucht er das Original.

Die Lastbalancierung kann den Algorithmus nach jedem Ereignis durchlaufen, d.h. wenn Aufträge entstehen oder beendet wurden, wenn Daten verschoben wurden, wenn sich Ressourcenauslastungen ändern oder wenn neue Informationen über Nachbarsysteme verfügbar sind.

2.3 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 eines 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 aus und verschieben Aufträge an Teilsysteme, die weniger belastet sind (Abschnitt 2.2).

2.4 Die adaptive 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, der Situation angepaßte Heuri-

stiken 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 zum Einsatz 'optimaler' Verfahren nicht gegeben sind. Im HiCon-Modell werden drei der dynamischen Lastbalancierung inhärenten Probleme durch automatische Adaption angegangen:

Das Kosten-Nutzen-Verhältnis, d.h. die Effizienz der Lastbalancierung ist in dynamischen Ansätzen kritisch. Die adaptive Komponente beobachtet daher einerseits die Profilvorabschätzungen der Aufträge und andererseits die Gesamtbelastung des Systems. Sind die Aufträge sehr klein und die Ressourcen des Systems nicht voll ausgelastet, so sollten keine aufwendigen Entscheidungen getroffen werden, sondern die Aufträge möglichst unmittelbar auf das System verteilt werden, um eine volle Nutzung des Systems zu erreichen und den Balancierungsaufwand zu reduzieren. Die Adaption stellt in solchen Situationen die Entscheidungsparameter entsprechend ein. Umgekehrt sollten Zuweisungsentscheidungen sorgfältiger und möglichst spät getroffen werden, wenn Aufträge groß sind oder ihre Laufzeit stark von Datenaffinitäten abhängt, oder wenn das System voll ausgelastet ist.

Adaption ist weiterhin notwendig, um fehlende bzw. falsche Vorabschätzungen über das Anwendungsverhalten seitens der Anwendungen zu kompensieren. Im HiCon-Modell zieht die adaptive Komponente durch Vergleich der angekündigten und der tatsächlichen Auftragslaufzeiten sowie der Datenkommunikationskosten Rückschlüsse über den Grad der Fehleinschätzungen (bzw. den Fehler durch fehlende Abschätzungen) und kann zukünftige Profilabschätzungen entsprechend korrigieren, sofern einfache, beispielsweise lineare, Abweichungen beobachtet werden.

Schließlich kann adaptive Lastbalancierung Mängel in ihrem Berechnungsmodell, das für Zuweisungsentscheidungen verwendet wird, durch Beobachtung des tatsächlichen Verhaltens korrigieren. Im HiCon-Modell kann die Lastbalancierung beispielsweise den typischen Rechenzeitanteil für Aufträge bestimmter Serverklassen ermitteln und dadurch die Last, die pro arbeitendem Server auf einem Prozessor entsteht, entsprechend vorabschätzen.

Die Herausforderung dieses adaptiven Ansatzes liegt in der Analyse der Sensitivität des Durchsatzverhaltens bestimmter Lastmuster gegenüber diversen Entscheidungsparameter-Einstellungen. Es muß festgestellt werden, aufgrund welcher aggregierter Meßinformationen und Vorabschätzungen welche Parameter nachgeregelt werden können. Die Erkennung und Evaluierung dieser Zusammenhänge ist derzeitig Hauptgegenstand der Bemühungen im HiCon-Modell. Wir können daher in diesem Artikel noch keine Meßergebnisse und Erfahrungen vorstellen.

3 Leistungsbewertung des HiCon-Modells

Verschiedene Anwendungen sollen nun unter verschiedenen Konfigurationen und Lastbalancierungsstrategien evaluiert werden, um die Vorteile und Schwächen des vorgeschlagenen Lastbalancierungskonzeptes zu ermitteln. Als Rechnersystem wurde ein Netz heterogener Arbeitsplatzrechner verwendet (Abbildung 5). Darauf wurden jeweils eine zentrale und eine dezentrale Struktur (Abbildung 6) der Lastbalancierung konfiguriert. Auf jedem Knoten wurden drei Server der Serverklasse konfiguriert, während die Clients bewußt ungleich verteilt wurden. Die Ungleichverteilung soll für die dezentrale Balancierung ein Lastungleichgewicht bewirken. Bei zentraler Balancierung spielt ja der Entstehungsort der Aufträge keine Rolle. Wir beschränken uns bei den Messungen auf den Mehrbenutzerbetrieb paralleler Anwendungen bei hoher Gesamtsystemlast. Das ist, zusammen mit sehr heterogenen Auftragsgrößen und bezüglich der Leistung

heterogener Rechenknoten, der für dynamische Lastbalancierung relevanteste, aber auch komplexeste Einsatzbereich. In der dezentralen Konfiguration bekommt jede Lastbalancierungskomponente einen Knoten als Teilsystem.

Bild5>

Pro Anwendung werden die folgenden sechs verschiedenen Lastbalancierungsvarianten bzw. Konstellationen betrachtet. Die ersten beiden sind dezentral, die übrigen zentral balanciert. Bei allen Meßreihen wurden Aufträge so zugewiesen, daß an den Servern eine mittlere Auftragsschlangenlänge von 1 vorlag, mit Ausnahme der fünften, die eine mittlere Schlangenlänge von 7 anstrebte. Auftragsprioritäten wurden nicht berücksichtigt, sondern stets der älteste Auftrag zuerst vergeben.

Bild6>

1. *Unbalancierter Ablauf*. Hier werden die Anwendungen von einer dezentralen Lastbalancierungsstruktur verwaltet, wobei die Balancierer innerhalb ihres Knotens die Aufträge strikt reihum verteilen und zwischen den Teilsystemen nur bei Lastunterschieden ab einem Toleranzfaktor von 1000 Aufträgen austauschen. Dadurch werden die Aufträge fast ausschließlich auf dem Knoten berechnet, auf dem sie entstanden.

2. *Dezentral balancierter Ablauf*. Innerhalb eines Teilsystems (das hier einem Knoten entspricht) werden die Aufträge weiter reihum verteilt, aber zwischen Teilsystemen wird jegliche Lastdifferenz ausgeglichen (Toleranzfaktor 1). Neben der impliziten Information durch Resultate tauschen die Balancierungskomponenten alle 100 Millisekunden ihre Lastzustände aus.

3. *Zentrale Reihum-Verteilung*. Eine Balancierungskomponente verwaltet das gesamte System und weist Aufträge strikt reihum zu.

4. *Zentrale Balancierung unter Berücksichtigung der reinen Rechenzeit*. Als bestgeeigneter Server wird derjenige gewählt, der die kürzeste Antwortzeit verspricht. Dabei wird die Abschätzung aus Abschnitt 2.2 verwendet, nur die Datenreferenzen werden ignoriert. Man beachte, daß diese Strategie von der Genauigkeit des Modells bereits bei weitem komplexer ist als sämtliche in der Literatur vorgestellten dezentralen Verfahren.

5. *Zentrale Balancierung nach Datenaffinität*. Hier wird jeder Auftrag an den Server vergeben, der gemäß der Abschätzung aus Abschnitt 2.2 die geringsten Wartezeiten für nicht lokal vorhandene Datensätze verspricht.

6. *Zentrale Balancierung unter Berücksichtigung von Rechen- und Datenkommunikationszeit*. Dieser Algorithmus bestimmt den bestgeeigneten Server vollständig gemäß der Abschätzung aus Abschnitt 2.2. Da man hier eine weitreichende und stabile Prognose der Last erwarten könnte, versucht der Algorithmus, die Auftragswarteschlangen der Server auf einem Mittel von 7 zu halten. Er weist also Aufträge früher fest zu als die anderen Strategien, um Vollbeschäftigung der Server zu erreichen und die Zahl der zentral liegenden Aufträge gering zu halten (was aus mehreren Gründen den Balancierungs-Overhead verringert). Weitergehende Untersuchungen zur Einstellung dieses Parameters finden sich in [4].

Bisher wurden drei verschiedenartige Anwendungen unter dem vorgestellten Lastbalancierungsmodell realisiert. Die Anwendungen sind jeweils parallelisiert und können in der Lastbalancierungsumgebung mehrfach konkurrierend ablaufen. Zudem können auch verschiedene Anwendungen mehrfach nebeneinander ablaufen. Für die Messungen wurde je eine Anwendung durch einen Client gesteuert. Die Server sind in der Lage, zwischen Aufrufen verschiedener Clients und damit zwischen verschiedenen Anwendungen umzuschalten. Konkurrierende Anwendungen können sowohl private Daten (die nur innerhalb einer Anwendung gemeinsam genutzt

werden, z.B. Zwischeninformationen während einer Bilderkennung), als auch globale Daten benutzen (etwa Polygon-Dateien einer Datenbank). Die Anwendungen 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.

3.1 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 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.

Bild7>

Der Graph ist in Form mehrerer Dateien abgespeichert. Eine Datei enthält jeweils alle Kanten, die von einem bestimmten Knotenbereich ausgehen (Abbildung 7 zeigt links die Aufteilung eines Graphen an). 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 einer Partition zu einem Aufruf gebündelt werden. Abbildung 7 skizziert in der Mitte und rechts den Ablauf zweier Suchschritte.

Für die Messungen wurde ein Auftragsgranulat in der Größenordnung von 1 - 20 Sekunden Verarbeitungszeit (auf Arbeitsplatzrechnern) gewählt. Die 20 Dateien des Graphen haben im Mittel eine Größe von 15 KByte. Gesucht wurde auf einem Graphen mit 1000 Knoten und 50000 Kanten, der so generiert wurde, daß im Mittel 95% der Kanten zwischen 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), entsteht eine Vielzahl sehr kleiner Folgeaufträge.

Diese Anwendung ist problemlos parallelisierbar, da sie, abgesehen von der Synchronisation durch den Client, kaum Netzlast und Wartezeiten durch Änderungsoperationen auf gemeinsamen Daten enthält. Abbildung 8 zeigt die Laufzeiten der verschiedenen Meßkonfigurationen; es wurden jeweils fünf Wegesuchen konkurrierend durchgeführt. Der unbalancierte Fall (1) kann als Vergleichsmaßstab dienen. Dabei ist zu beachten, daß das verteilte Betriebssystem (hier als Laufzeitsystem auf UNIX gesetzt) trotzdem vorhanden ist und daß die Anwendungen, obwohl parallelisiert, nur auf dem Rechenknoten laufen, auf dem der zugehörige Client liegt. Die hohe Laufzeit rührt daher, daß mit zwei Anwendungen auf dem langsamsten und keiner Anwendung auf einem der leistungsfähigsten Knoten eine initiale Lastungleichverteilung vorliegt. Die dezentrale Lastbalancierung (2) ist in der Lage, diese Ungleichverteilung zu reduzieren. Sie ist damit der zentralen Reihum-Verteilung (3) überlegen, da Reihum-Verteilung nur unter der Annahme gleichgroßer Aufträge und homogener Rechenknoten ideal ist. Die Betrachtung der Datenaffinitäten (5) zahlt sich in dieser Anwendung nicht aus, da sich das Anlegen von Kopien für die Lesezugriffe länger-

fristig lohnt; so ist auch (6) etwas langsamer als Strategie (4), die auf die kürzeste Antwortzeit hin optimiert, ohne Datenzugriffe zu beachten und insgesamt am besten abschneidet.

Bild8>

3.2 Flächensegmentierung zur Bilderkennung

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 9): 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.

Bild9>

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. 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 10 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.

Bild10>

Im Vergleich der verschiedenen Lastbalancierungsstrategien und -strukturen (Abbildung 11) zeigt hier die zentrale Reihum-Verteilung (3) die schlechtesten Resultate. Bei dem hohen Datenkommunikationsaufkommen ist es sehr schlecht, die Aufträge ohne Rücksicht auf Datenaffinitäten zu verteilen. Auch die Beachtung der Rechenleistungen (4) alleine kann die Laufzeiten nicht wesentlich verbessern. Das relativ gute Abschneiden des unbalancierten Ablaufs (1) erklärt sich daraus, daß die Daten zumindest lokal auf dem Rechenknoten bleiben und über weniger Server zerstreut werden. Auch die dezentrale Balancierung (2), die ja keine Datenkommunikation berücksichtigt, verläuft ähnlich gut, weil die Aufträge zum Großteil lokal und ansonsten blockweise zwischen den Teilsystemen transferiert werden. Der Favorit für diese Anwendung ist eindeutig die zentrale Strategie (5), die lediglich nach Datenaffinitäten balanciert. Die komplexeste Strategie (6) ist

unterlegen, weil die Balancierungskomponente in den Phasen Square-Split & -Merge und Boundary-Search stark überlastet wird.

Bild11>

3.3 Verwaltung geometrischer Objekte mit Hilfe von R-Bäumen

Die dritte 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. 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 (Abbildung 12; die Rechtecke sind jeweils Daten-Partitionen). 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-Baumteile bzw. durch Anlegen von Kopien effizient parallelisiert werden.

Bild12>

Bei Einfügeoperationen müssen Zugriffs- und Datenpartitionen modifiziert werden. Im Gegensatz zu eindimensionalen B^+-Bäumen beschränken sich die Änderungen meist 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 (ein Verbund wird hier nicht in sich parallelisiert) relativ große Aufträge sind; die Auftragsgrößen schwanken zwischen 2 und 700 Sekunden.

Für die Messungen werden insgesamt 5 * 510 Polygone eingefügt und 5 * 3 Verbunde mit räumlichen Prädikaten berechnet (Abbildung 13 zeigt das Ablaufprofil, das jeder Client erzeugt). Die Polygone sind in 27 Dateien zu je ca. 10 KByte partitioniert, die Zugriffstrukturen belegen 350 Dateien einer Größe um 100 Byte.

Bild13>

Abbildung 14 stellt die Laufzeiten der R-Baum-Anwendungen gegenüber. Wegen des sehr feinen Auftragsgranulats der Einfügeoperationen und der hohen Empfindlichkeit gegenüber Datenaffinitäten kann der Lastausgleich der dezentralen Struktur (2) nur wenig Verbesserung gegenüber dem unbalancierten Fall (1) aufweisen. Eine Auftragsverschiebung durchläuft ja im dezentralen Fall mindestens zwei Balancierungskomponenten. Die zentrale Reihum-Verteilung (3) ist dadurch etwas schneller. Während die Lastbalancierung mit Berücksichtigung der Rechenkapazitäten (4), (6) signifikante Vorteile bringt, versagt die reine Betrachtung der Datenaffinitäten (5) deshalb, weil ein beträchtlicher Teil der Verbundoperationen auf dem langsamsten Knoten plaziert wird.

Bild14>

3.4 Lastbalancierung heterogener Lastgemische

Als letzte Meßreihe beobachten wir ein heterogenes Gemisch von Lasten auf dem parallelen System unter verschiedenen Balancierungsstrategien und -strukturen. Dazu werden die drei Anwendungen jeweils mehrfach konkurrierend abgewickelt. Abbildung 15 zeigt die Client-Server-Konfiguration; die zentrale bzw. dezentrale Lastbalancierungsstruktur sind wie bisher. Die

Anwendungen wurden dazu auf etwa gleichen Aufwand dimensioniert, d.h. die Wegesuchen verlief auf je 150000 Kanten und es wurden je R-Baum-Applikation 260 Einfüge- und 2 Verbundoperationen berechnet.

Bild15>

Die Laufzeiten in Abbildung 16 zeigen, daß zentrale Reihum-Verteilung (3) keine Verbesserung gegenüber dem unbalancierten Ablauf (1) erbringt, da die stark steigende Datenkommunikation die etwas bessere Prozessorausnutzung wieder kompensiert. Die reine Minimierung der Datenkommunikation (5) liefert sehr schlechte Ergebnisse, weil sie die Wegesuchen nicht genügend parallel ablaufen läßt. Während der dezentrale Lastausgleich (2) eine Leistungssteigerung von 10% gegenüber dem unbalancierten Fall erbringt, kann die zentrale Lastbalancierung unter Beachtung der Prozessornutzung (4), (6) den Systemdurchsatz verdoppeln.

Bild16>

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. Dabei werden vorrangig die vorgestellten Konzepte zur dynamischen Lastbalancierung plausibel gemacht. Die Verwendung des Client-Server-Modells für parallele und konkurrierende Verarbeitung auf vernetzten Rechnern ergibt, zusammen mit der Erweiterung um das Konzept der Arbeit auf virtuell gemeinsamen Daten, für dynamische Lastbalancierung ein erhebliches Potential zur anwendungsunabhängigen Optimierung des Systemdurchsatzes. Dabei können Vorabschätzungen von Seiten der Anwendungen gewinnbringend eingesetzt werden.

Da der Großteil der realisierten und simulierten Konzepte in Forschungsprojekten völlig dezentrale Verfahren verwendet, wurde hier nachgewiesen, daß es durchaus sinnvoll ist, Teilsysteme durch zentrale Balancierungsverfahren zu verwalten. Es wurde gezeigt, daß verschiedene Ressourcen, nicht nur die Systemgröße ‘cpu run queue length’, für Balancierungsentscheidungen relevant sind. Die Konzepte der zentralen und dezentralen Balancierungsstrategien unter Betrachtung verschiedener Ressourcen wurden mithilfe von Messungen verschiedener Anwendungen auf realen Systemen validiert. Da in den Messungen nur ein kleines System mit fünf Knoten und sechs Prozessoren verwendet wurde, tritt eine Überlastung der zentralen Balancierungskomponente nur in Abschnitt 3.2 sichtbar auf. Wie zuvor in Abschnitt 2.3 erklärt, muß man sich jedoch darüber im klaren sein, daß die zentrale Struktur nicht unbegrenzt skalierbar ist, sondern große Systeme eine gemischt zentral - dezentrale Struktur benötigen. Weiterhin wurden die Möglichkeiten zur automatischen Adaption der Lastbalancierung aufgezeigt, wodurch verschiedene Schwächen dynamischer Lastbalancierungsansätze verbessert werden können. Genauere Auswertungen stehen momentan im Mittelpunkt der Arbeiten.

Neben den hier vorgestellten Anwendungen werden noch ein exakter Löser für lineare Gleichungssysteme, ein Szenario komplexer, parallelisierter relationaler Datenbankanfragen sowie eine Spannungsberechnung in Körpern mit der Methode der finiten Elemente zum Ablauf unter dynamischer Lastbalancierung nach dem HiCon-Konzept realisiert. Ergebnisse liegen für die Datenbankoperationen bereits vor [5], wobei besonders die Betrachtung von Auftragsgruppen gute Resultate zeigte.

Durch weitere Untersuchungen soll einerseits die Einsatzfähigkeit des Konzeptes für Rechnersysteme in der Größenordnung von 100 Rechenknoten evaluiert werden, wobei neben Arbeitsplatzrechnern auch Parallelrechner mit schnellen Kommunikationskanälen einbezogen werden sollen;

andererseits soll das Modell für geographisch weiter verteilte Rechnernetze erprobt und ausgebaut werden.

Abschließend sei Herrn Prof. Reuter und den Gutachtern des Artikels herzlicher Dank für ihre wertvolle Unterstützung ausgesprochen.

5 Literaturverzeichnis

1. 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)

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

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

4. Becker, W., Pollak, R.: Efficiency of Server Task Queueing for Dynamic Load Balancing. Fakultätsbericht 1994 /8, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner (1994)

5. Becker, W., Waldmann, G.: Exploiting Inter Task Dependencies for Dynamic Load Balancing. IEEE 3rd Symposium on High Performance Distributed Computing (1994)

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

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

8. 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)

9. 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)

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

11. 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)

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

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

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

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

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

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

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

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

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

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

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

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

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

25. 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)

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

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

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

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

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

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

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

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

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

35. 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)

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

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

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

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

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

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

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

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

44. 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)

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

46. 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)

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

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

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

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

51. 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)

6 Abbildungen

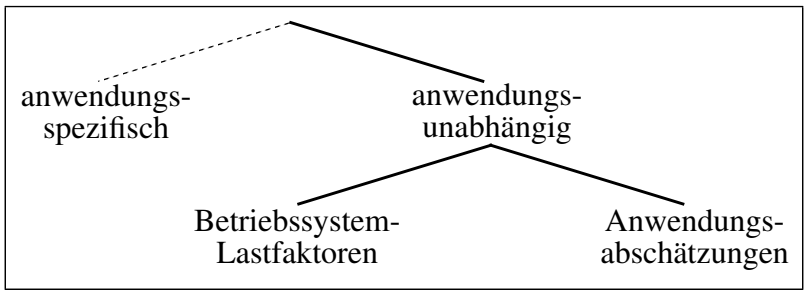

Abbildung 1: Klassifikation der genutzten Lastbalancierungsinformationen.

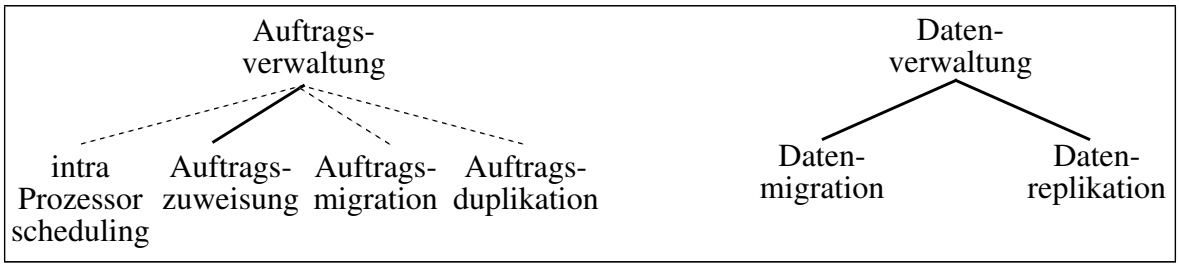

Abbildung 2: Klassifikation der Lastbalancierungsaufgaben.

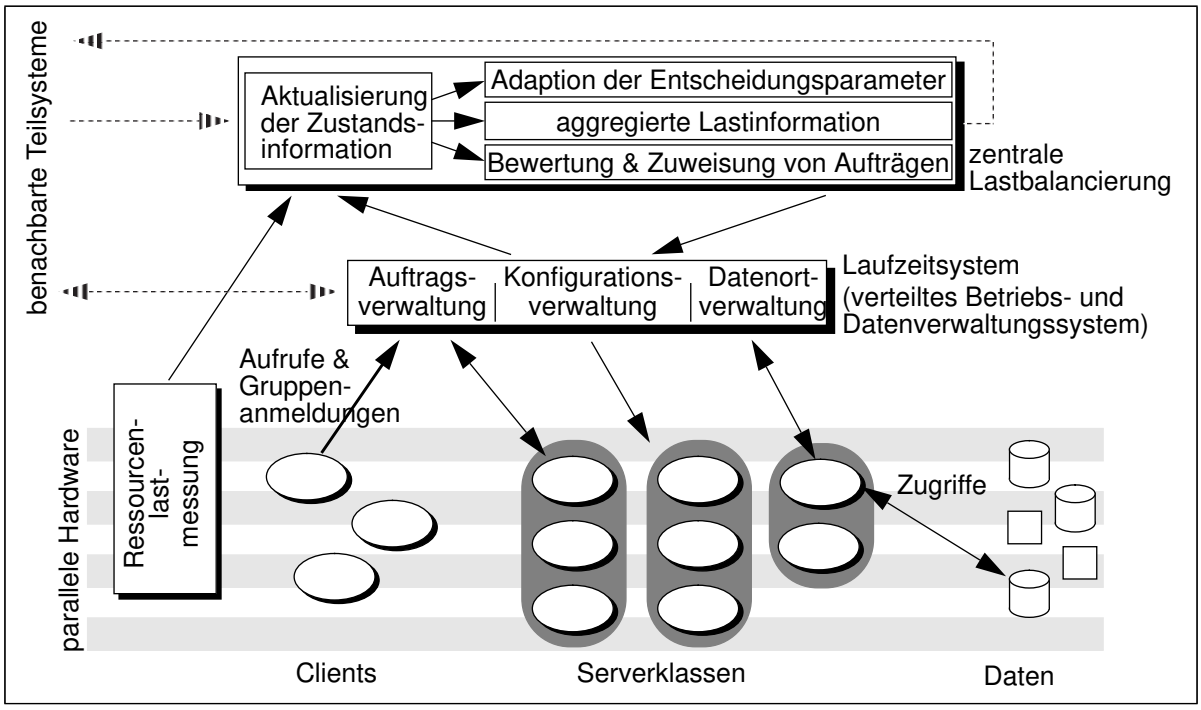

Abbildung 3: Struktur der Lastbalancierungsumgebung.

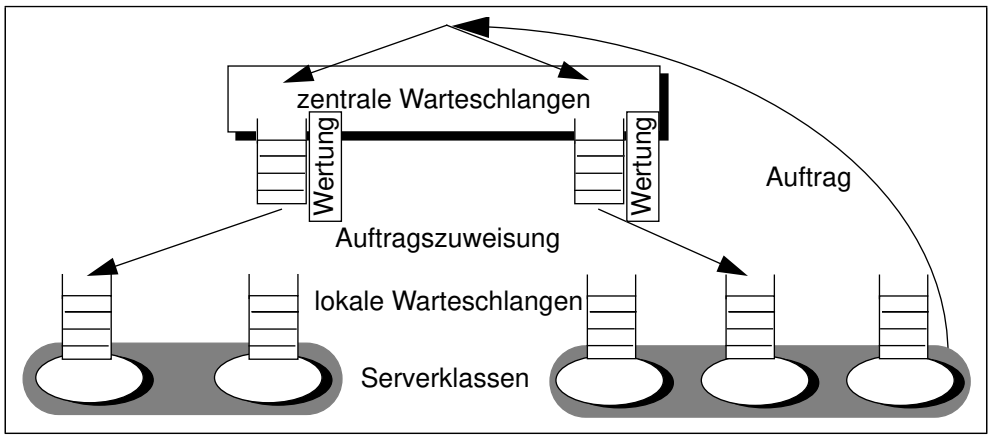

Abbildung 4: Auftragsverwaltung der Lastbalancierungsumgebung.

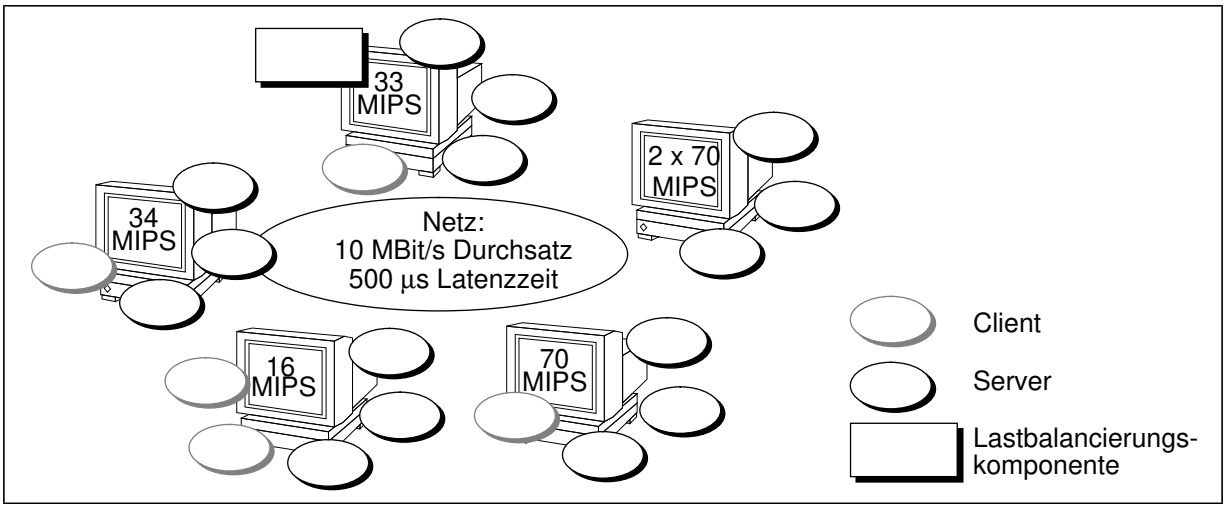

Abbildung 5: Die für die Messungen verwendete zentrale Konfiguration.

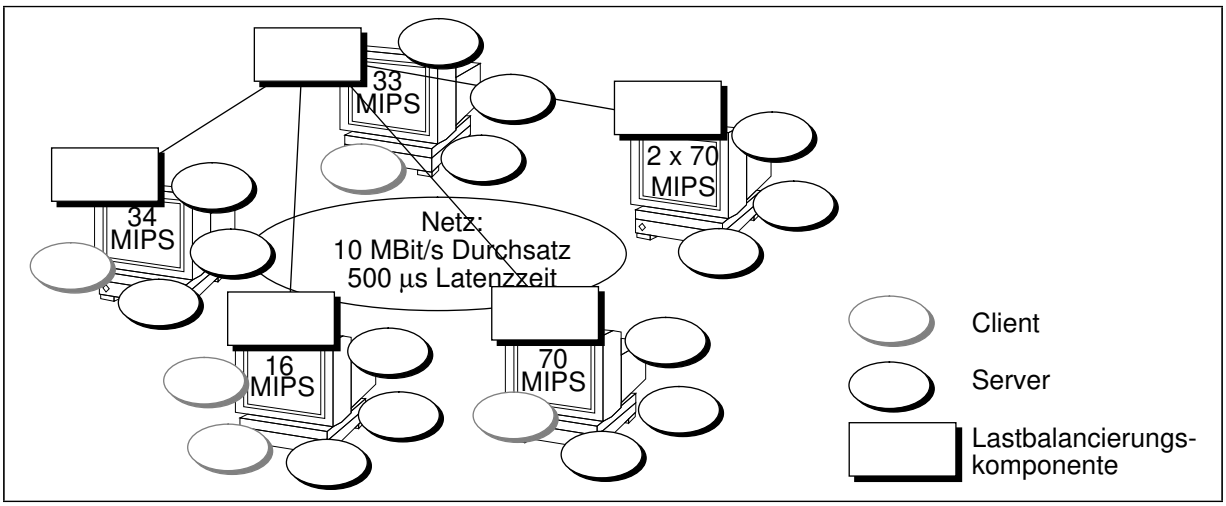

Abbildung 6: Die für die Messungen verwendete dezentrale Konfiguration.

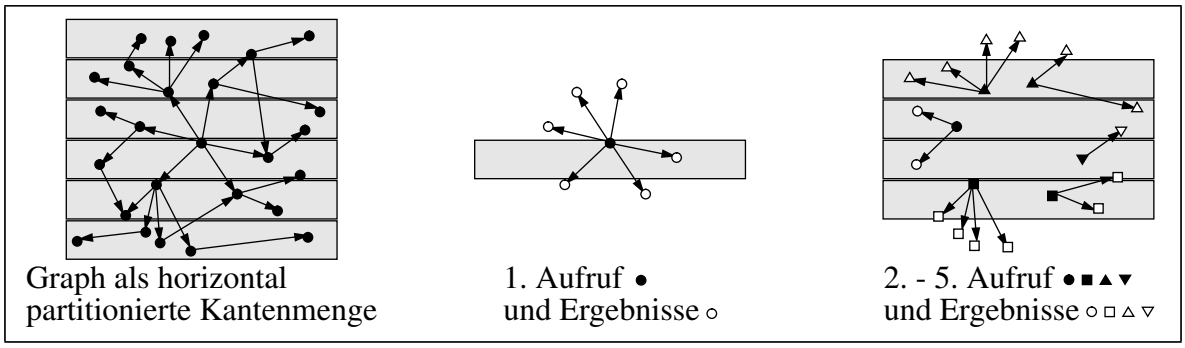

Abbildung 7: Aufteilung der Graphdaten und der Aufrufe in der Wegesuche.

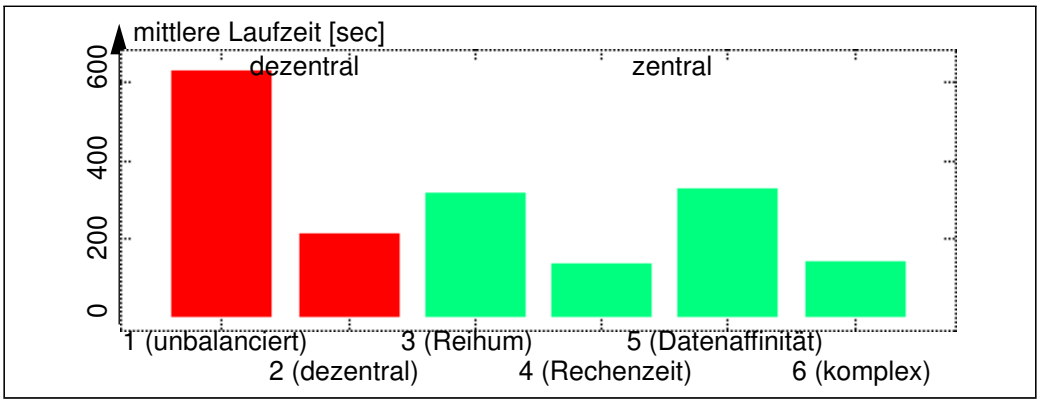

Abbildung 8: Laufzeiten der Wegesuche.

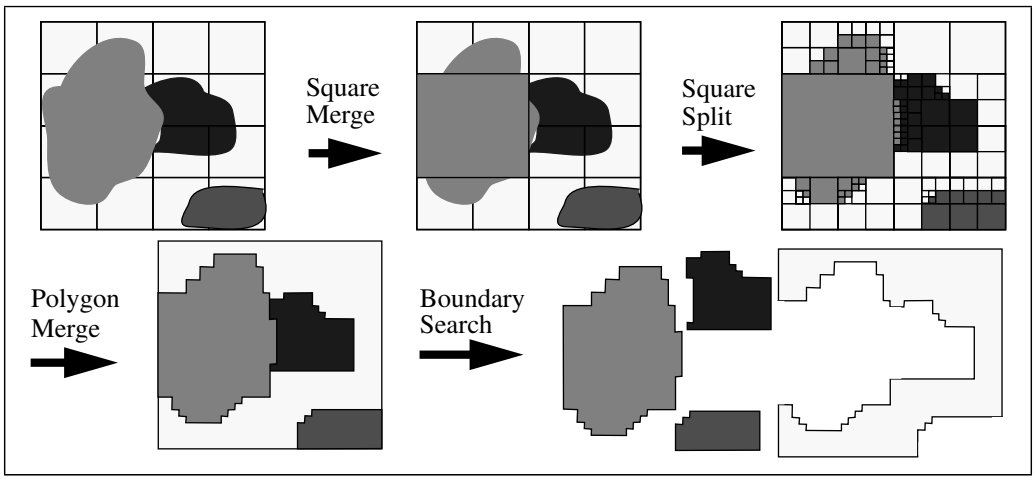

Abbildung 9: Schritte des Bildsegmentierungs-Algorithmus.

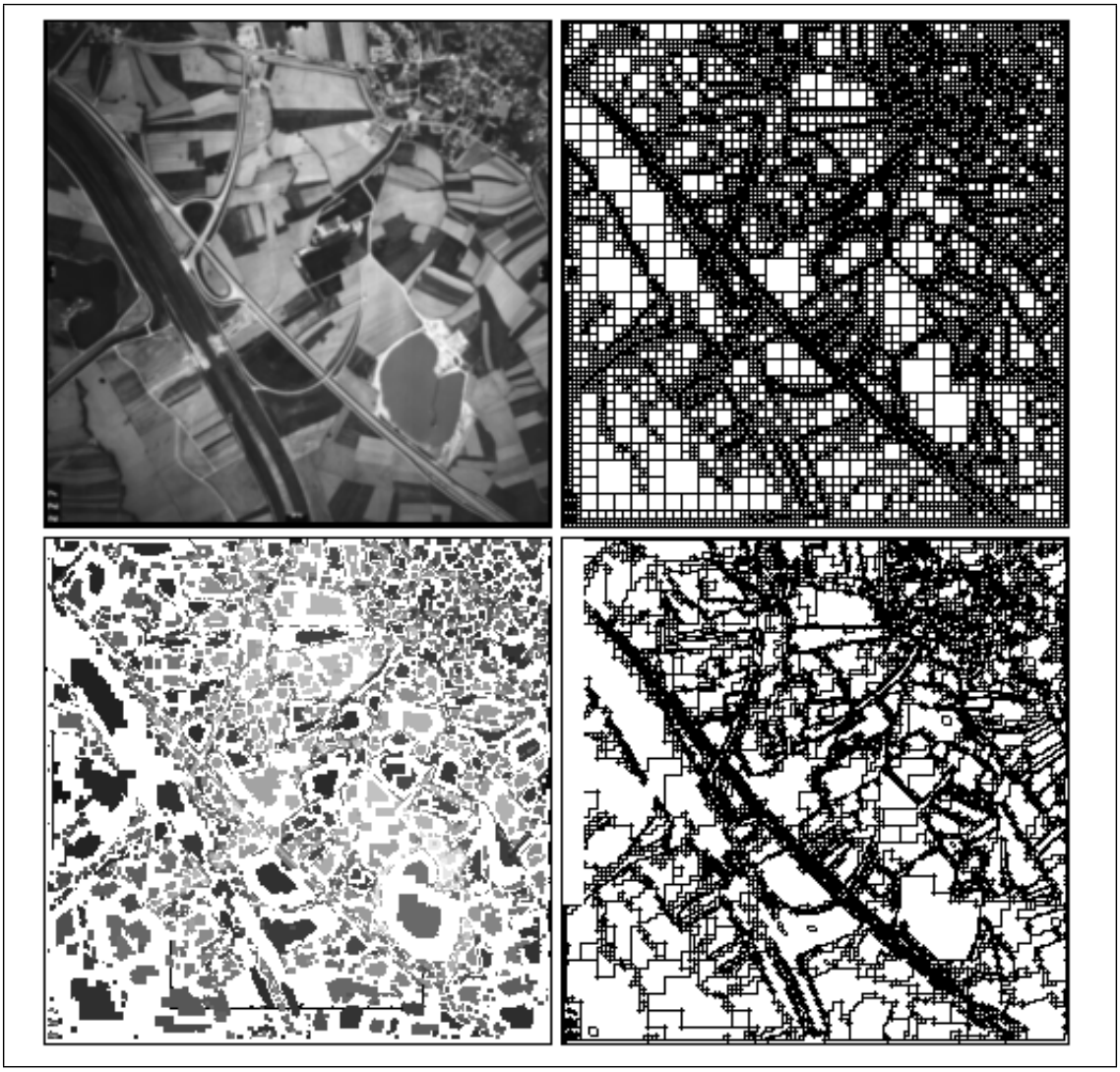

Abbildung 10: Zur Messung der Flächensegmentierung verwendetes Bild.

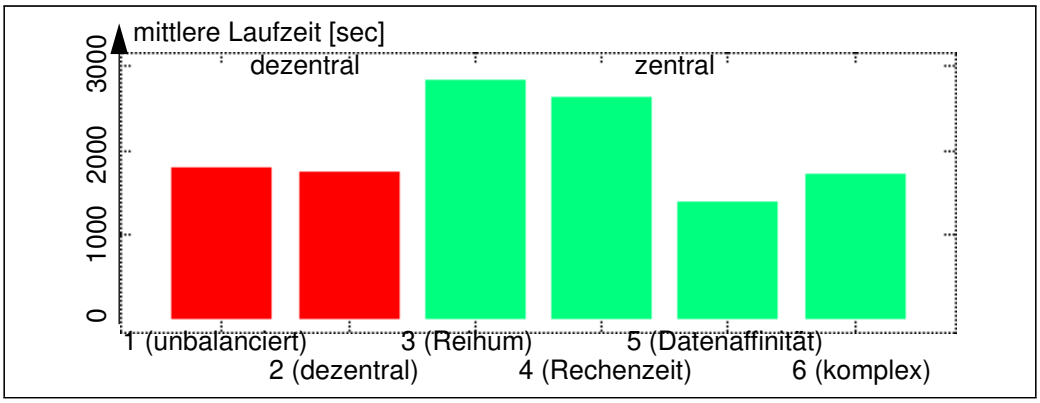

Abbildung 11: Laufzeiten der parallelen Flächensegmentierung.

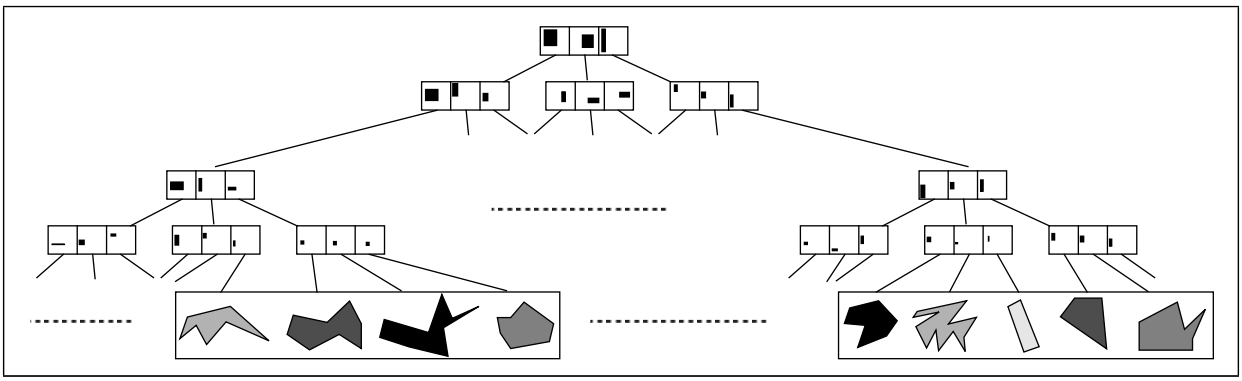

Abbildung 12: Verteilte Datenstrukturen zur Verwaltung der Polygone.

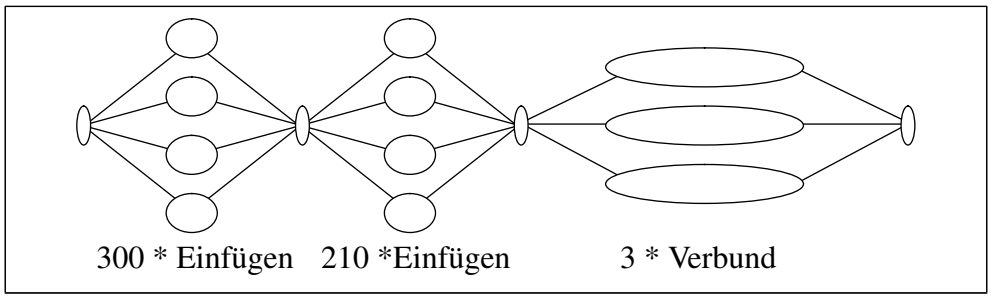

Abbildung 13: Ablauf einer Anwendung auf R-Bäumen für die Messungen.

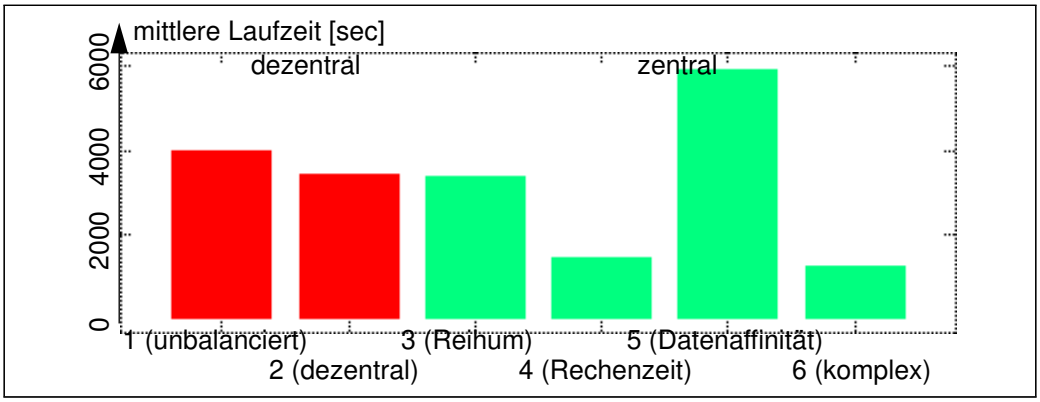

Abbildung 14: Laufzeiten der parallelen R-Baum-Operationen.

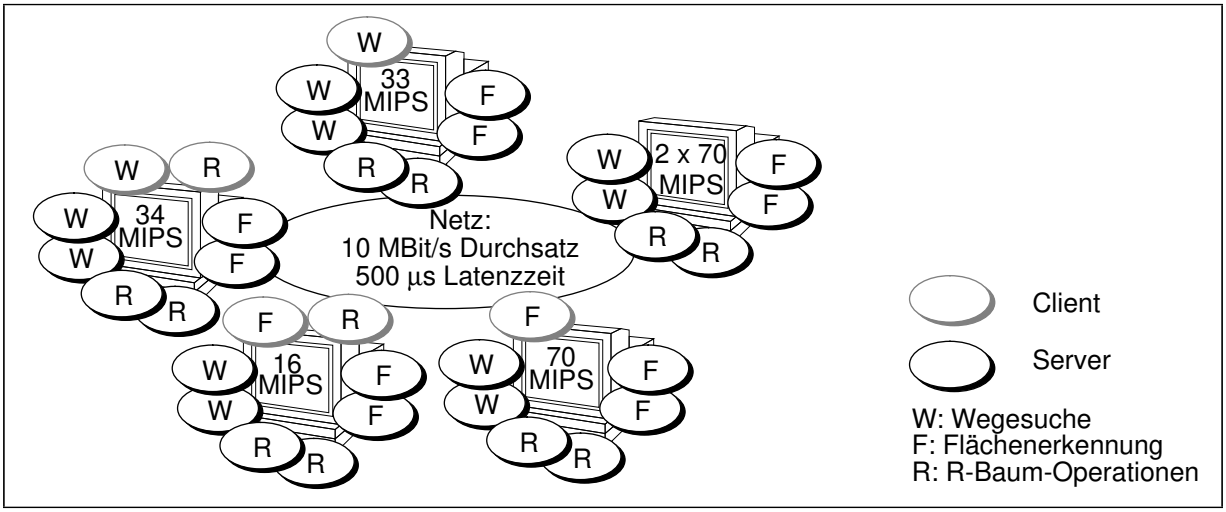

Abbildung 15:Konfiguration für das heterogene Lastszenario.

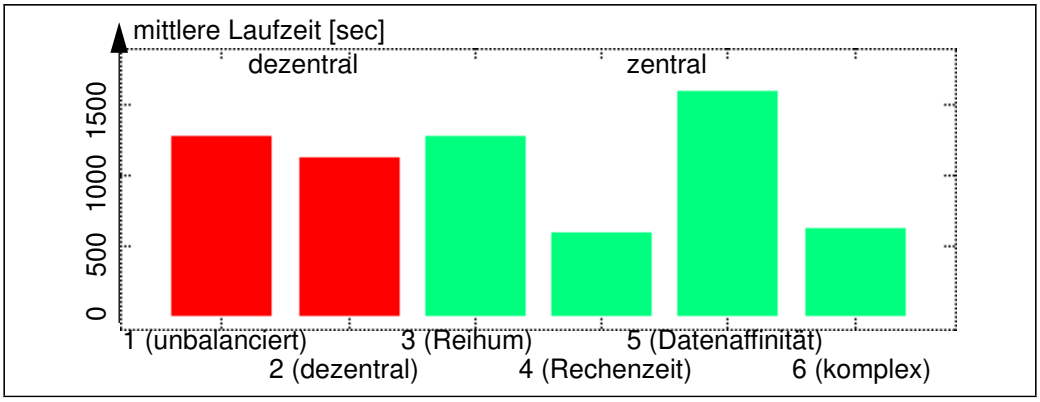

Abbildung 16: Laufzeiten des heterogenen Lastszenarios.

7 Liste der Abbildungsunterschriften

Abbildung1: Klassifikation der genutzten Lastbalancierungsinformationen.

Abbildung2: Klassifikation der Lastbalancierungsaufgaben.

Abbildung3: Struktur der Lastbalancierungsumgebung.

Abbildung4: Auftragsverwaltung der Lastbalancierungsumgebung.

Abbildung5: Die für die Messungen verwendete zentrale Konfiguration.

Abbildung6: Die für die Messungen verwendete dezentrale Konfiguration.

Abbildung7: Aufteilung der Graphdaten und der Aufrufe in der Wegesuche.

Abbildung8: Laufzeiten der Wegesuche.

Abbildung9: Schritte des Bildsegmentierungs-Algorithmus.

Abbildung10: Zur Messung der Flächensegmentierung verwendetes Bild.

Abbildung11: Laufzeiten der parallelen Flächensegmentierung.

Abbildung12: Verteilte Datenstrukturen zur Verwaltung der Polygone.

Abbildung13: Ablauf einer Anwendung auf R-Bäumen für die Messungen.

Abbildung14: Laufzeiten der parallelen R-Baum-Operationen.

Abbildung15:Konfiguration für das heterogene Lastszenario.

Abbildung16: Laufzeiten des heterogenen Lastszenarios.

8 Kurzbiographie

Wolfgang Becker, geb. 1966 in Aachen, lebt in der Schillerstadt Marbach, wo er auch bis 1985 die Schule besuchte. Nach Ableistung des Wehrdienstes studierte er zwischen 1986 und 1990 Informatik an der Universität Stuttgart mit dem Schwerpunkt Software und Abschlußarbeiten auf dem Gebiet der künstlichen Intelligenz. Seit 1990 arbeitet er dort am Institut für parallele und verteilte Höchstleistungsrechner als wissenschaftlicher Mitarbeiter im Bereich der Datenbanken und der Parallelverarbeitung. Freizeitausgleich findet er vor allem im aktiven Rudersport.