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-7000 Stuttgart 80

Globale dynamische Lastbalancierung in datenintensiven Anwendungen

Wolfgang Becker

Fakultätsbericht Nr. 1993 / 1

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

Wolfgang.Becker@informatik.uni-stuttgart.de

Globale dynamische Lastbalancierung in datenintensiven Anwendungen

Inhaltsverzeichnis

1 Das HiCon Projekt

Dieser Bericht stellt einen Ansatz zur dynamischen Lastbalancierung paralleler und verteilter Programme vor. Im ersten Kapitel wird das Projekt anhand der betrachteten Anwendungen, Rechnersysteme und Lastbalancierungsverfahren charakterisiert. Die Systemstruktur sowie das Balancierungsprinzip werden in Kapitel 2 und 3 ausführlich erklärt, Kapitel 4 stellt Realisierungen und Ergebnisse des Modells vor und Kapitel 5 gibt einen abschliessenden Ausblick.

1.1 Motivation

In der Vielzahl in der Literatur existierender Untersuchungen zur Lastbalancierung zeichnen sich einige grundsätzliche Probleme ab, von denen wir einige durch den in diesem Bericht vorgestellten Ansatz bewältigen wollen. Die Realität stellt Lastbalancierungsverfahren vor ein bestehendes System, dessen Struktur und Schnittstellen meist für eine automatische Lastverteilung schlecht geeignet sind. Es besteht kaum Information über das zu erwartende Verhalten von Anwendungen, Betriebssystemschnittstellen schränken die Funktionalität und Effizienz der Lastbalancierung ein und zwischen den einzelnen Schichten der Software findet kein Informationsaustausch statt. Die transparente Verteilung von Abläufen und Daten ist ein wesentlicher Fortschritt für Programmierer, Lastbalancierungsverfahren müssen jedoch um den damit verbundenen Synchronisations- und Nachrichtenaufwand wissen.

Wir haben daher ein Verarbeitungsmodell entwickelt, das dem Programmierer transparente Verteilung und Replikation paralleler Ausführungen und Daten in heterogenen Systemen ermöglicht, dem Lastbalancierungsmechanismus aber eine Kostenabschätzung erlaubt. Die explizite Integration von Datenbeständen im Konzept erlaubt nunmehr eine Balancierung aufgrund der drei relevanten Faktoren - des Rechenaufwandes, der Datenlokalität und des Kommunikationsbedarfs.

Lastbalancierung im HiCon Projekt wird in mehrere Ebenen unterteilt. Die unteren Ebenen sind für die Ausführung einer bestimmten Balancierungsstrategie zuständig. Das umfaßt Aufgaben wie die Verwaltung einer geeigneten Serverkonfiguration, die Zuweisung von Aufträgen, die Verteilung der Daten sowie das Nachrichten-Routing. Die explizite Verwaltung statischer Datensätze ermöglicht sowohl die dynamische Migration und Replikation von Daten als auch die scheinbare Migration von Servern bzw. laufenden kooperierenden Teilausführungen in heterogenen Architekturen. Die obere Ebene des Lastbalancierungsmechanismus verwaltet einen Katalog möglicher Strategien und bringt je nach Profil der laufenden Anwendungen geeignete Methoden zum Einsatz. Dieses Konzept ermöglicht durch bessere Adaption an verschiedenartige Anwendungsprofile die Balancierung eines größeren Anwendungsspektrums und kann für bestimmte Profile sehr einfache, spezialisierte Strategien nutzen.

Das vorgestellte Lastbalancierungskonzept berücksichtigt sowohl eine vollständigere Menge relevanter Faktoren und Größen und verfügt über einen komplexeren Zuweisungsmechanismus als bisher bekannte Studien. Dadurch kann eine breitere Klasse von Applikationen und Systemkonfigurationen durch ein anwendungsunabhängiges automatisches Lastbalancierungsverfahren effizient betrieben werden.

An dieser Stelle wollen wir anhand einer kurzen Entwicklung von Lastbalancierungsverfahren verdeutlichen, daß in der Tat ein Bedarf an fortgeschrittenen Methoden besteht:

Primitive Balancierungsverfahren arbeiten ohne Messung des Systemverhaltens und ohne Vorwissen über die Aufträge. Die wichtigsten Beispiele sind das Random- sowie das Round Robin-Verfahren, die sich in der Realität als relativ gut erwiesen (siehe etwa [Trippner92]). Sie basieren auf der Annahme, Aufträge und Rechner seien völlig homogen. Sie enthalten weitere implizite Annahmen, wie wir im folgenden noch sehen werden. Derartige Verfahren versagen leider bereits bei einer schwankenden Grundlast auf dem System (Mehrbenutzerbetrieb).

Anspruchsvollere Verfahren berücksichtigen das dynamische Systemverhalten. Üblicherweise mißt man die Run Queue Length der Prozessoren (siehe [Ferrari86]). Sind mehr Prozessoren (bzw. Server) frei als Aufträge anstehen, so bevorzugt man Prozessoren mit kurzer Run Queue, andernfalls gibt man den Auftrag dem ersten Server, der sich frei meldet. Ein solches Verfahren wurde etwa im *Supervisor* [Schiele91], auf dem das *HiCon*-Projekt aufbaut, erfolgreich realisiert. Allerdings geraten Strategien, welche die Prozessorwarteschlange betrachten, in Schwierigkeiten, sobald man etwa datenintensive Berechnungen betrachtet. Der erhebliche Aufwand zum Heranschaffen der Daten bzw. die Beauftragung des dortigen Servers zerstört oft die geplante Lastverteilung. Neben der Run Queue Length sind also für manche Applikationen auch andere Warteschlangenlängen von Interesse - die Zahl der auf Plattenzugriffe oder vor Datensperren wartenden Prozesse (siehe Studien wie [Zhou87]). Doch auch in dieser Kategorie von Balancierungsverfahren unterstellt man unter anderem allen Aufträgen denselben Aufwand und scheitert, sobald die einzelnen Aufträge signifikant unterschiedliche Profile zeigen.

Zieht man außer der Systemlast auch noch Wissen über die Aufträge in Betracht, so erhält man theoretisch günstigere Ergebnisse. Das HiCon Modell kann neben der Länge der Server-Warteschlangen und der Sekundärlast auch den Rechenumfang und Datenbedarf der Aufträge einbeziehen (vgl. etwa [Yu86], [Yu91]). In der Praxis stößt man jedoch auf massive Probleme: die im voraus verfügbaren Angaben sind sehr unzuverlässig und der Aufwand für die Balancierung selbst wächst prohibitiv. Dennoch ignoriert man selbst in dieser Strategien-Klasse noch Reihenfolgeabhängigkeiten und Kommunikation zwischen den Aufträgen. Die Verfahren zeigen deutliche Schwächen etwa bei Anwendungen, die nebst Parallelität einen kritischen sequentiellen Pfad enthalten oder auch in Anwendungstypen mit intensiv kooperierenden parallelen Teilaufträgen, wie der Pipeline-Verarbeitung.

Balancierungsalgorithmen, welche auch diese Situationen meistern, findet man in der Praxis nicht. Jedoch existieren, vor allem im Bereich der statischen Planung - dem sogenannten Scheduling, Ansätze für einzelne Teilprobleme. Relevant sind beispielsweise die Arbeiten von [Chandy75], [Lam75], [Bokhari81], [Bryant81], [Ma82], [Bruno85], [Blazewicz86], [Iqbal86], [Thomasian86], [Towsley86], [Li90] oder [Kanet91]. Hier zwingt der große, exponentiell mit Aufträgen und Rechnern wachsende Balancierungsaufwand zu sehr einfachen, anwendungsspezifischen Heuristiken. Der HiCon-Ansatz sieht hierfür ein zweistufiges Balancierungsmodell vor, in dessen oberer Ebene aufgrund gegebener oder gemessener Anwendungsprofile passende - möglichst einfache, spezialisierte - Strategien erwählt werden, die von der unteren Ebene effizient durchgeführt werden können.

1.2 Anwendungsgebiete

Im HiCon Projekt untersuchen wir im wesentlichen datenintensive Berechnungen. Darunter fallen der Bereich der Verwaltung großer Datenmengen (Informationssysteme) sowie Applikationen, die umfangreiche Transformationen auf relativ stabilen, globalen Datenstrukturen durchführen, an. Diese Charakterisierung soll die betrachtete Domäne einerseits von Datenflußanwendungen abgrenzen, bei welchen man typischerweise kleine Datenmengen durch eine Reihe von Bearbeitungsfunktionen schleust, und andererseits von stark dialogorientierten Applikationen trennen, die viele sehr kurze, unvermittelt auftretende Verarbeitungsschritte aufweisen.

Als relevante Arbeiten in diesem Bereich betrachte man etwa [Yu86], [Ciciani88] und [Yu91], die ein Verfahren vorstellen, um Transaktionen an verteilte Datenbank-Server zuzuweisen. Die eigentliche Datenzugriffe werden vom Server am Ort der Daten durchgeführt. Lastbalancierung besteht hier in der Abwägung zwischen Ausnutzung der Rechenkapazitäten und dem Aufwand für Remote-Datenzugriffe. [Varadarajan88] untersuchen Datei-Migration in verteilten Datenbanken. Anhand der Datenverteilung und Zugriffsmuster wird erwogen, ob Datenmigration oder Ausführung des Auftrages am Ort der Daten günstiger ist. Dabei werden Daten nur zwischen und Aufträge nur innerhalb von sogenannten Regionen bewegt. Innerhalb einer Region wird der Auftrag bzw. der Datenzugriff am Ort der Daten durchgeführt. Optimierungskriterium ist die Minimierung der Migrationskosten, wobei die Ausführungszeiten ein Zeitlimit einhalten müssen. [Copeland88] betrachten die Partitionierung und Verteilung von Daten über Platten in Anwendungen, die von Datenzugriffen dominiert werden. Sie gehen davon aus, daß Operationen direkt am Ort der Daten durchzuführen sind. [Gavish90] geben einen Überblick diverser Datei-Allokationsverfahren.

Weiterhin sollen hauptsächlich mäßige bis grobe Parallelisierungen betrachtet werden. Die sequentiellen Teiloperationen haben einen gewissen Rechenumfang und bearbeiten eine größere Menge von Daten. In diesem Umfeld kann es nicht Ziel sein, die maximal mögliche Parallelität eines Problems zu nutzen; oft ist es sinnvoller, Algorithmen anzuwenden, die ein relativ grobes Granulat aufweisen und somit besser auf die Fähigkeiten der betrachteten Hardware und auf die Methoden der Lastbalancierung abgestimmt sind.

Die Auswahl des Anwendungsgebietes bzw. der Algorithmenklasse hat Einfluß auf das gewählte Ausführungsmodell sowie auf die Klasse der zu untersuchenden Lastbalancierungsmethoden. So sind neben der Rechenkapazität die Verfügbarkeit und der Lagerort von Daten relevante Faktoren. Aufwendige, zentralisierte Ansätze zur Lastbalancierung sind erfolgversprechend.

1.3 Die Rechnerumgebung

Heutige Rechensysteme verfügen typischerweise über sehr starke zentrale Rechenleistung und vergleichsweise langsame und aufwendige Kommunikationsmöglichkeiten. Massiv parallele Architekturen sind (Software-seitig) äußerst schwer zu beherrschen und werden sich langfristig in Richtung der zentralen Prozessoren mit leistungsfähiger Vektor- und Pipeline-Unterstützung bewegen. Für SIMD-Systeme sowie Vektor- und Pipeline-Einheiten ist automatische dynamische Lastbalancierung weniger relevant, da hier weder Mehrbenutzerbetrieb noch funktionale Parallelität als wesentlicher Gesichtspunkt erscheint. Die auftretende massive Datenparallelität wird derzeit am besten durch einfache, statische Algorithmen balanciert.

Diese Überlegungen und der oben gewählte Fokus betrachteter Applikationen lassen es sinnvoll erscheinen, ein Rechnermodell bestehend aus relativ wenigen, sehr starken und über eigene Ressourcen verfügenden, lose gekoppelten Prozessoren zugrunde zu legen. Die Entscheidung für dieses Modell schließt die Einbindung anderer Architekturen in HiCon nicht aus, jedoch wird die Lastbalancierung andere Systeme, z.B. SIMD oder Shared Memory Systeme, nicht optimal nutzen.

1.4 Das Ausführungsmodell

Applikationen nutzen die Parallelverarbeitung, indem der Algorithmus auf mehrere kooperierende Prozesse verteilt wird. Das Granulat der Aufteilung in Teilberechnungen sowie die Kooperationsform zwischen diesen wird durch die jeweilige Anwendung eingeschränkt und sollte außerdem auf die verwendete Rechnerumgebung angepaßt werden.

Eine automatische Lastbalancierung benötigt aber neben dem Wissen über das Verhalten der Hardware auch Wissen über die Struktur und den Ablauf der Anwendungen. Wir wollen uns bewußt nicht darauf beschränken, Prozesse unbekannten Verhaltens bezüglich ihrer Rechenarbeit, des Kommunikationsverhaltens und ihrer Datenzugriffe zu balancieren. Vorabschätzungen können über die Beschreibung einzelner sequentieller Einheiten hinaus auch die Kooperationsform, gegenseitige Abhängigkeiten zwischen diesen oder Ablaufhäufigkeiten der Einzelberechnungen umfassen.

Im HiCon Projekt entwickeln wir ein Programmier- und Ablaufmodell, das eine auf die Hardware angepaßte, flexible Implementierung von Algorithmen erlaubt und der Lastbalancierung entscheidende Informationen zur günstigen Abwicklung zur Verfügung stellen kann. Eine Anwendung gliedert sich in Teilfunktionen, die durch je eine Serverklasse modelliert werden. Eine Serverklasse stellt also eine Funktion bzw. einen Satz von Operationen auf gewissen Datenobjekten dar. Instanzen dieser Serverklassen können sich beliebig gegenseitig Nachrichten bzw. Aufrufe und Resultate zusenden. Die Funktionalität einer Serverklasse wird durch mehrere Serverinstanzen realisiert. Im HiCon-Modell wird funktionale Parallelität durch verschiedene Serverklassen, Datenparallelität durch Einsatz mehrerer Instanzen pro Serverklasse oder multi-threaded Instanzen modelliert.

Serverinstanzen können zur Laufzeit generiert und gelöscht werden können, jedoch liegt hier der Gedanke einer relativ statischen Server-Konfiguration zugrunde. Neben den auf heutigen Betriebssystemen hohen Prozeßstartkosten basiert diese Entscheidung auf folgenden Beobachtungen: bei relativ groben Granulat an Parallelität ist es sinnvoll, nicht für jeden Auftrag unverzüglich eine neue Instanz bzw. einen neuen Thread zu erzeugen, sondern den Auftrag in eine zentrale und später in die lokale Warteschlange einer Instanz einzureihen. Dadurch limitiert man die entstehende Parallelität auf ein Maß, welches das System in einem vernünftigen Arbeitsbereich hält. Zu häufige Prozesswechsel, Überlauf des Hauptspeichers, eine hohe Kollisionsrate auf dem Netzwerk und Blockierungen durch überfüllte Puffer bringen bekanntlich extreme Leistungseinbrüche. In datenintensiven Anwendungen kommt der Aspekt des nicht zu vernachlässigenden Datenbestandes der Serverinstanzen hinzu, der jeder neu kreierten Instanz zuerst geschickt, kopiert bzw. zugänglich gemacht werden muß.

Die Anzahl und Verteilung der Serverinstanzen ist für den Programmierer nicht sichtbar, wohl aber die Existenz einzelner Instanzen. So können Nachrichten an eine Serverklasse oder an eine spezielle Instanz gerichtet werden. Diese Möglichkeit wird hauptsächlich zur Rückgabe von Resultaten oder in stehenden Verbindungen zum Zwecke intensiver Kommunikation, d.h. kontextsensitiven Aufrufen, genutzt.

Wir betrachten derzeit keine Prozessmigration auf Betriebssystem-Ebene, obwohl dies in einigen Projekten erfolgreich angewandt wird (siehe etwa [Ezzat86], [Douglis91]), da unser Lastbalancierungsmodell zum einen für heterogene Systeme ausgelegt ist, zwischen denen es in absehbarer Zeit noch keinen Standard geben wird, der das Verschicken der Prozesskontrollblöcke sowie die transparente ‘Verlängerung’ von I/O-Verbindungen erlaubt; zum anderen ist Prozeßmigration hauptsächlich in Modellen mit dynamischer Prozeßerzeugung und sehr langlaufenden sequentiellen Aufträgen (etwa kompletten Anwendungen) interessant, d.h. in Systemen, die für jeden Aufruf einen eigenen Server kreieren. Migrationskosten bewegen sich heute in der Größenordnung eines Prozeßstarts. In Client-Server Umgebungen haben einzelne Aufrufe vergleichsweise kürzere Bearbeitungszeiten und man kann prinzipiell für jeden Serverklassen-Aufruf eine andere Instanz gewählt werden, sodaß die Migration einer arbeitenden Serverinstanz nicht wirtschaftlich ist. Das Zufügen und Löschen von Instanzen ist hier eine längerfristige Rekonfigurationsoperation und wird durchgeführt, wenn ein Anwachsen bzw. Abfallen der Parallelität oder Veränderungen in den Datenzugriffsprofilen längerfristig erwartet wird oder beobachtet wurde.

Durch die Möglichkeit kontextsensitiver Aufrufe ist jedoch eine der Migration vergleichbare Flexibilität erreichbar. Die Instanzen einer Serverklasse verfügen über einen gemeinsamen Datenbestand, der anwendungstransparent verteilt und konsistent gehalten wird. Wenn man den Kontext einer ‘Session’ innerhalb dieses Datenbestandes verwaltet, so kann eine Session problemlos mit einer anderen Serverinstanz fortgesetzt werden. Die Aufspaltung langlaufender Funktionen in mehrere Client-Server Kooperationsschritte verlangt also keine feste Bindung an eine Serverinstanz. Es ist allein eine Entscheidung der Lastbalancierung, wann sich ein Wechsel und die damit verbundene Bewegung des Kontextes lohnt.

Jede Serverklasse verfügt im HiCon Modell über statische Datenbestände, die von der Anwendung in mehrere Partitionen beliebig unterteilt werden können. Hier kann der Programmierer die datenparallele Ausführung von Funktionen ermöglichen, indem Serverinstanzen nur einzelne Partitionen zum Zugriff sperren und nach Abschluß eines kritischen bzw. zusammengehörigen Abschnittes wieder freigeben. Die Verteilung, Duplizierung und Konsistenthaltung der Datenpartitionen ist für den Programmierer nicht sichtbar, sondern wird vom Laufzeitsystem unter dem Einfluß der Lastbalancierung abgewickelt. Der Datenbestand einer Serverklasse umfaßt dabei beliebige Datenstrukturen im Hauptspeicher und auf dem jeweils lokalen Plattenspeicher, d.h. sowohl flüchtige als auch nichtflüchtige Daten, und wird von den Instanzen einer Klasse als gemeinsamer Speicher betrachtet.

In diesem Ansatz sind mehrere Formen der Kooperation zwischen Clienten und Servern erfaßt und die zur Lastbalancierung relevanten Informationen können auf einheitliche Weise beschrieben werden. Ein SQL-Server verwaltet beispielsweise eine Menge von Relationen als statische Datensätze. Da das Resultat eines jeden Aufrufs an ihn vom aktuellen Inhalt der Datenbank abhängt, kann man diese Relationen als den Kontext des SQL-Servers betrachten. Ein Anwendungs-Server hingegen, der einen Teil der Datenbank kapselt und Operationen auf diese Daten zur Verfügung stellt, hat keinen eigenen Kontext. Dennoch hängen die Aufrufe an ihn vom Zustand der Datenbank, d.h. vom Kontext des SQL-Servers ab. Das Szenario ist in Abbildung 1 skizziert. Aus dem Blickwinkel der Lastbalancierung interessiert zum einen, welche Datenbestände von einer Serverklasse verwaltet werden und wie diese Datenbestände verteilt sind. Es ist günstig, Aufrufe an eine Serverinstanz zu senden, die den Großteil der benötigten Daten lokal vorliegen hat. Zum anderen interessiert bei Aufrufen an Serverklassen, die nur mittelbar mit großen Datenmengen arbeiten, wie nah sie den Serverinstanzen stehen, welche die datenintensiven Operationen ausführen. Dadurch kann die Lastbalancierung versuchen, den Kommunikationsaufwand gering halten.

In [Becker92] haben wir verschiedene Programmiermodelle in Bezug auf die Lastbalancierung verglichen. Ein prozeduraler Programmierstil mit Remote Procedure Calls wird im HiCon Modell

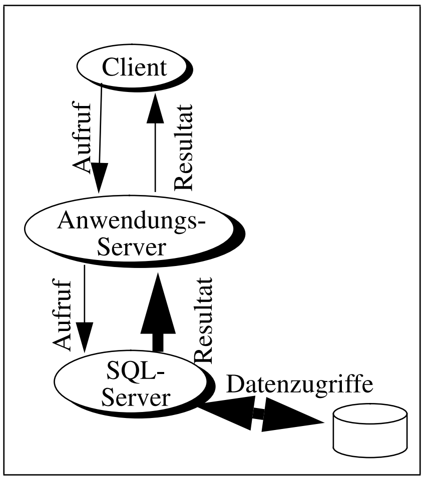

Abbildung 1 : Lokalität von Datenzugriffen und kurze Kommunikationswege als Lastbalancierungskriterien.

durch Serverklassen-Aufrufe unterstützt, die Interaktion zwischen Client und Server ist jedoch im Prinzip nicht auf die Aufrufparameter und das Resultat beschränkt. In *Fortran* ist es üblich, daß die Teilfunktionen einer Anwendung auf einem globalen Satz von Daten arbeiten. Ähnlich, jedoch in strukturierter Weise operieren im Datenbankbereich verschiedene Anwendungen auf globalen Datenstrukturen. Diese Vorgehensweisen sind im HiCon Modell nur eingeschränkt möglich, da statische Datenbestände nur jeweils den Instanzen einer Klasse zugeordnet sind. Die zunehmend allgemein akzeptierten objektorientierten Konzepte verlangen, daß Datenstrukturen durch Objekte gekapselt werden und nur über die dort angebotenen Methoden vom Objekt selbst bearbeitet werden können. Eine Serverinstanz kann also nicht auf die statischen Daten einer anderen Serverklasse zugreifen, sondern muß sie dort bearbeiten lassen oder sich die Daten als Aufrufparameter zusenden lassen. Parallelarbeit wird durch Replikation von Serverinstanzen erreicht, auf die auch die statischen Klassendaten verteilt sind.

1.5 Der Lastbalancierungsansatz - Konzept und Klassifikation

Angesichts der Vielzahl veröffentlichter Studien zur Lastbalancierung im weiten Sinne wenden wir uns sofort komplexeren zentralen Verfahren im Bereich datenintensiver Anwendungen zu. Für eine allgemeinere Einführung in den Bereich der Lastbalancierung bzw. Lastverteilung sei auf [Becker92], [Casavant88] oder [He89] verwiesen.

Im HiCon Projekt untersuchen wir globale zentrale Lastbalancierungsverfahren. Dabei können die Entscheidungsträger der Balancierung durchaus in Form einer hierarchischen Struktur über das Rechnersystem verteilt sein. Unter den oben umrissenen Vorgaben und Bedingungen erscheint eine Zentralstelle zur Verwaltung der Zustandsinformation und der darauf basierenden Entscheidungsfindung sinnvoll. Um zu vermeiden, daß diese einen Engpaß bildet, ist eine Einschränkung ihres Wirkungsbereiches notwendig. Daher wird die Lastbalancierungsaufgabe an Instanzen verteilt, die jeweils ihre lokale Umgebung verwalten. Diese Lastbalancierungsinstanzen werden wiederum von übergeordneten Balancierungskomponenten verwaltet. Hinter dieser Struktur steckt die Idee, Lastbalancierungsentscheidungen vorrangig lokal abzuwickeln und nur größere Objekte oder in Fällen stärkerer Ungleichverteilung der Systemlast auf höherer Ebene zu operieren. Das Ziel der global optimalen Lastverteilung kann ja offensichtlich in unserer Umgebung aufgrund des Aufwandes für Kommunikation und für die Lastbalancierung selbst nicht vollständig erreicht werden.

Ein zentraler Balancierungsansatz ist nicht selbstverständlich, zumal sich in der Literatur viele erfolgversprechende dezentrale Verfahren finden (siehe etwa [Barak85], [Eager85], [Eager86], [Hsu86], [Lin87], [Kale88], [Cybenko89], [Hosseini90] und [Kuchen90]). Als weitere Alternative beschreibt [Smith80] ein komplexes Verfahren der Interaktion zwischen Clienten und Servern zur optimalen Lastverteilung, eine Variante des allgemeinen Bidding-Ansatzes (siehe [He89], Kapitel 2.2.A und 2.2.B). Im MIMD-Bereich werden jedoch Nachrichten durch schnell wachsende lokale Prozessorleistung und sehr beschränkt steigende Kommunikationsgeschwindigkeit immer teurer. Das erfordert eine eher grobgranulare Parallelisierung der Anwendungen. Bei der Balancierung größerer Einheiten lohnt es sich jedoch, genauere Informationen über Aufträge und Systemzustand einzusetzen, was durch einen (logisch) zentralisierten Ansatz ermöglicht wird (siehe auch Kapitel 3.5.4).

Die Aufgabe der Lastbalancierung besteht im wesentlichen darin, Teilaufgaben bestimmten Instanzen zur Ausführung zuzuordnen. Die Verteilung von Daten steht nicht unter dem unmittelbaren Zugriff der Balancierung, wird aber bei der Zuweisungsentscheidung berücksichtigt und damit durch die Plazierung der Teilaufgaben in vorhersehbarer Weise beeinflußt (siehe Kapitel 3.4). Die zu verteilenden lasterzeugenden Objekte sind hier sowohl einzelne Aufrufe zwischen Servern als auch ganze Gruppen von Aufrufen, d.h. die Ausführung komplexer Funktionen.

Das Optimierungskriterium der Lastbalancierung ist die im Mittel möglichst schnelle Ausführung aller im System ablaufenden Aufträge, also die Minimierung der gesamten Ausführungszeit. Das bedeutet, daß ein neuer Auftrag nicht isoliert betrachtet wird, sondern so in das aktuelle Systemgeschehen eingebunden wird, daß seine Laufzeit einschließlich der Laufzeitzunahme der übrigen Aufträge kleinstmöglich ausfällt. Diese Sichtweise wird oft als soziale Lastbalancierung bezeichnet.

Besteht ein komplexer Auftrag aus einer Menge von Teilaufgaben, so wird im HiCon-Modell lediglich seine Gesamtausführungszeit minimiert. Die Laufzeiten der einzelnen zugehörigen Subaufträge interessieren nicht primär, sie werden vielmehr auf die schnellstmögliche Bearbeitungszeit des Hauptauftrages hin ausgelegt. Dieser Aspekt ist für Aufgaben, deren Teilaufträge gewissen Reihenfolgebeziehungen unterliegen - also nicht völlig unabhängig sind -, von großer Bedeutung (siehe etwa die am Ende des Kapitels 1.1 angegebenen Referenzen).

Das betrachtete Balancierungsverfahren agiert dynamisch, d.h. es mißt zur Laufzeit Ressourcenbelastung und Antwortzeitverhalten und berücksichtigt diese im weiteren Vorgehen. Durch die zweistufige Architektur gesellt sich die HiCon-Lastbalancierung zu den adaptiven Verfahren, welche aufgrund von Laufzeitauswertungen dynamisch ihre Balancierungsstrategie, d.h. nicht nur die Parameter der Entscheidungsfunktion sondern den Algorithmus selbst, der Lastsituation anpassen. Die Grenzen dieser Aufgliederung sind jedoch fließend.

In den folgenden Kapiteln werden wir den Aufbau und die Funktionsweise des im HiCon Projekt entwickelten Lastbalancierungsverfahrens detailliert beschreiben.

2 Die Struktur des HiCon-Systems

Abbildung 2 zeigt die logische Struktur des HiCon-Ansatzes. Auf die einzelnen Komponenten sowie die Kooperation zwischen ihnen wird im Verlaufe des Kapitels genauer eingegangen.

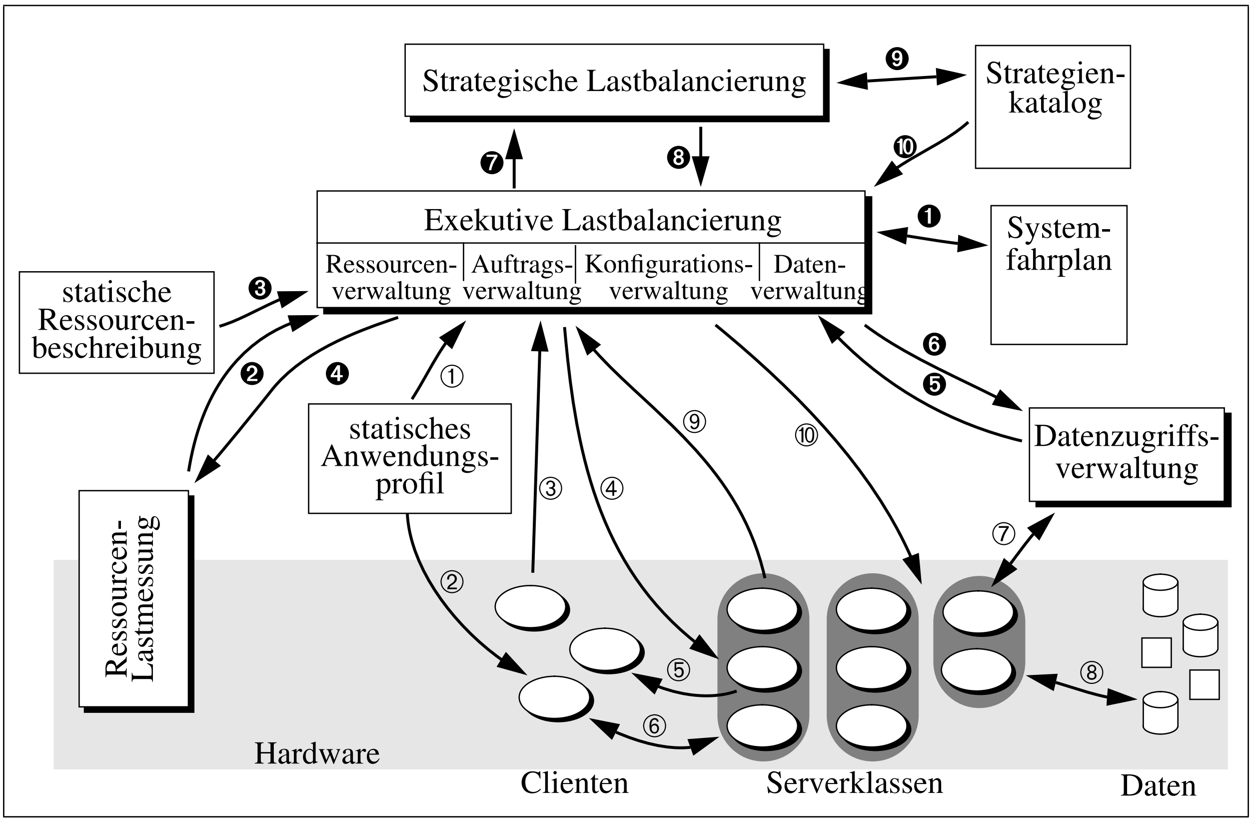

Abbildung 2 : Systemkomponenten des HiCon Ansatzes.

Eine Anwendung besteht aus einer Menge von Serverklassen, die in Abbildung 2 als graue Rahmung der Serverinstanzen eingezeichnet sind. Die Anwendung wird statisch durch ein Profil charakterisiert; die Elemente einer solchen Beschreibung werden unten erläutert. Aufgrund des statischen Profils ① kann die exekutive Lastbalancierung bereits sowohl Konfigurationsentscheidungen für Server-instanzen und Daten treffen ⑩❻ als auch der strategischen Balancierung Hinweise zur Auswahl geeigneter Strategien liefern ❼.

Der Ablauf einer Anwendung besteht - neben lokalen Berechnungen innerhalb der Clienten und Serverinstanzen - aus Aufrufen an Serverklassen, Rückgabe von Resultaten, direkter Kooperation zwischen einer Client- und einer Serverinstanz und Datenzugriffen der Serverinstanzen. Die Einteilung der Instanzen in Client oder Server ist dabei dynamisch und beschreibt lediglich die momentane Rolle der betreffenden Instanz. Jede Instanz kann beispielsweise gleichzeitig als Server einen Auftrag bearbeiten und als Client Teilaufträge, die dabei anfallen, an andere Server delegieren. Beim Aufruf einer Serverklasse ③ kann der Client die Lastbalancierung aufgrund statischen Vorwissens ② und aktueller Laufzeitparameter durch Abschätzung des durch den Auftrag entstehenden Aufwandes unterstützen. Die exekutive Lastbalancierung wird den Auftrag irgendwann einer der Instanzen der betreffenden Serverklasse zuweisen ④. Die Entscheidung beruht auf unterschiedlichen Informationen, wird aber im Prinzip aus dem Systemfahrplan ❶ hergeleitet.

Nachdem die Serverinstanz den Auftrag bearbeitet hat, sendet sie evtl. Resultate an den Aufrufer zurück ⑤. Sie kann die Lastbalancierung dann - oder auch während der Bearbeitung - durch Informationen über ihren Zustand unterstützen ⑨. Bei kontextsensitiven Aufrufen kommt es zu direkter Kooperation zwischen Client und Server-Instanz ⑥. Hier ist es besonders wichtig, die Lastbalancierung über den Fortgang der Bearbeitung zu informieren ⑨.

Im HiCon-Konzept werden globale, statische Datensätze explizit berücksichtigt. Statisch bedeutet, daß sie über die Bearbeitung eines Auftrags hinweg gültig bleiben. Solche sind etwa globale Datensätze, Serverkontext-Information und Dateien. Die Datenzugriffsverwaltung dient der Synchronisation konkurrierender Zugriffe, der Verwaltung von Kopien und der Auslösung einer Migration solcher Datensätze. Bevor Serverinstanzen auf statische Datensätze zugreifen ⑧, melden sie dies bei der Datenzugriffsverwaltung an ⑦, welche den Datensatz entsprechend sperrt und der Instanz über ihre lokalen Ressourcen zugänglich macht. Die Datenzugriffsverwaltung informiert die Lastbalancierung über Ort und Zustand der Datensätze ❺ und erhält von dort Hinweise zur günstigen Verteilung ❻.

Aus der Server- und Datenkonfiguration, der momentanen Auslastung der Ressourcen ❷ und deren Charakteristika ❸ kann die exekutive Lastbalancierung erkennen bzw. absehen, wodurch welche Last erzeugt wird und welche Auslastung und Laufzeit ein Auftrag an welcher Stelle mit sich bringen würde. Diese Aufzeichnungen führt sie im Systemfahrplan ❶. Je nach derzeit angewandter Strategie ❿ stellt sie die benötigten Größen und Intervalle für die Ressourcen-Messung ein ❹. Sie liefert der strategischen Lastbalancierung regelmäßig längerfristig aggregierte Informationen über das Profil der Anwendung, Verhalten der Ressourcen und die erreichten Laufzeiten ❼.

Die strategische Lastbalancierung verfügt über eine Sammlung diverser Lastbalancierungsstrategien ❾, aus denen sie jeweils die derzeit günstigste zum Einsatz bringt ❽. Neben den eigentlichen Strategien, die aus Methoden für die verschiedenen Entscheidungspunkte der exekutiven Balancierung bestehen, verwaltet sie jeweils Kriterien, unter denen die Strategien gut verwendet werden können. Durch Beobachtung des Systems ❼ und Vergleich mit den für die Balancierungsmethoden gegebenen Voraussetzungen stellt sie fest, wann es sinnvoll ist, auf eine andere, speziellere oder allgemeinere Strategie umzuschalten. Darüberhinaus hat die strategische Balancierung auch die Möglichkeit, durch Beobachtung des erreichten Durchsatzes die Einsatzfähigkeits-Kriterien der verwendeten Methode im Strategienkatalog zu verbessern. Diese Fähigkeit wollen wir jedoch im HiCon Modell nicht weiterverfolgen.

Diese Systembeschreibung zeigt die logische Struktur des HiCon-Konzepts, die von der physischen Aufteilung der Komponenten unabhängig ist. Clienten und Serverinstanzen werden, mit Ausnahme von multi-threaded Servern, als je ein Prozeß realisiert. Die Datenzugriffsverwaltung ist teils in den Prozessen der Serverinstanzen verteilt und teils in der Lastbalancierung verwirklicht, sodaß einerseits lokale Zugriffe auf die über das System verstreuten Daten effizient sind und andererseits die Lastbalancierung einen Überblick und Einfluß auf die aktuelle Verteilung der Daten hat. Die exekutive Lastbalancierungskomponente wird gemeinsam mit der strategischen, abgesehen von wenigen unterstützenden Funktionen, die in den Serverinstanzen liegen, als zentraler Prozeß realisiert. Für große Systeme besteht die Möglichkeit, diese Komponente als eine Hierarchie von autonomen, kooperierenden Lastbalancierungseinheiten zu gestalten. Die Ressourcen-Lastmessung geschieht in Form eines Prozesses je Rechner. Dabei kann die Ressourcenbelastung relativ eng gekoppelter Parallelrechner meist durch einen einzigen Prozeß abgelesen werden.

3 Einzelne Komponenten im Detail

Wir wollen nun ausgewählte Funktionen des HiCon Modells genauer betrachten.

3.1 Anwendungs- / Auftragsprofile

Der Ablauf einer Anwendung gliedert sich im HiCon-Modell in eine Anzahl von Funktionsausführungen durch Serverinstanzen. Die Anwendung kann prinzipiell allein aufgrund der Programme der Serverinstanzen, d.h. ohne weitere statische oder dynamische Angaben, ausgeführt werden. Die statische und dynamische Charakterisierung der Anwendung ist redundant und dient im HiCon-Projekt nur zur Unterstützung der Lastbalancierung.

Ohne statische oder dynamische Voreinschätzung des Verhaltens von Aufträgen muß die Lastbalancierungsstrategie ein festes Standardprofil für alle Aufträge annehmen, was oft zu gravierenden Fehlentscheidungen führt. Dennoch begnügen sich nahezu alle bekannten Verfahren damit, weil in vielen Anwendungen keine exakten Vorhersagen über das Laufzeitverhalten möglich sind: sie hängen stark von aktuellen Laufzeitdaten ab.

3.1.1 Statische Anwendungsprofile

Die statische Beschreibung einer Anwendung macht Aussagen über Auftragsbearbeitungen, Server-Aufrufe und Datenzugriffe ①②. Diese Abschätzungen gelten für den gesamten Verlauf der Bearbeitung. Im HiCon Projekt folgen wir dabei folgendem Konzept: für jede Serverklasse kann man einige Auftragstypen herauskristallisieren. Ein Aufruf eines Clienten läßt sich dadurch aufgrund seiner Parameter und aufgrund des Zustandes der Serverinstanz einem bestimmten Auftragstyp zuordnen. Beispielsweise würde man bei einer Serverklasse, die eine Liste verwaltet, die Auftragstypen ‘Suche ein Element in der Liste’, ‘Füge ein Element ein’ und ‘Lösche ein Element’ unterscheiden.

Für jeden Auftragstyp kann man den mittleren Aufwand an Rechenzeit und externen Datenzugriffen pro Ausführung, die Anzahl von Unteraufrufen an verschiedene Serverklassen und das Schreib- / Leseverhältnis bei Zugriffen auf globale Daten abschätzen. Global kann man Angaben über die Häufigkeit und potentielle Parallelität von Server-Aufrufen bestimmter Auftragstypen sowie die dadurch entstehende Kommunikationslast machen. Genauere Ausführungen finden sich in [Becker92] und [Staib92].

Statische Profile werden bisher vor allem in Scheduling-Verfahren für Produktionsplanung und Batch-Betrieb verwendet, können jedoch auch für dynamische Balancierung eine wesentliche Hilfe darstellen.

3.1.2 Dynamische Auftragsprofile

Für dynamische Lastbalancierung sollte das Profil eines Auftrages so spät wie möglich, d.h. erst unmittelbar bei seiner Zuweisung, festgestellt und genutzt werden. Zu diesem Zeitpunkt sind relativ verläßliche Abschätzungen über die Ressourcenbedürfnisse möglich. Im HiCon Modell kann beim Absenden eines Auftrags ③ das Profil dieses Auftrages durch folgende Größen charakterisiert werden:

- Ein Maß für die benötigte Prozessor-Rechenzeit sowie die Anzahl der notwendigen Plattenzugriffe. Unser Modell kennt dazu keine weiteren Aufgliederungen, da eine Serverinstanz nur auf einem Prozessor arbeitet, nicht migriert und ausschließlich auf lokal angeschlossene Platten zugreift. Wir verzichten auch auf eine explizite Unterscheidung zwischen mehreren lokalen Platten, da solche auf längere Sicht in Form von Disk-Arrays als eine logische Platte organisiert werden.

  Anhand der beiden Größen kann man neben absoluten Laufzeitabschätzungen anhand des Verhältnisses zwischen Rechenzeit und Zugriffswartezeit auch die Auslastung der Platte sowie die prozentuale Prozessorauslastung absehen. Dabei sind allerdings noch synchronisationsbedingte Wartezeiten zu beachten.
- Eine Auflistung der vorraussichtlich zur Auftragsbearbeitung benötigten globalen Daten. Dabei gibt man an, mit welcher Wahrscheinlichkeit Änderungsoperationen auf die jeweiligen Daten geschehen bzw. die Wahrscheinlichkeit, daß nur lesend zugegriffen wird. Während es in einer allgemeinen Programmierumgebung nahezu unmöglich ist, dies in einheitlicher Form zu beschreiben, können solche Angaben im HiCon Modell relativ einfach ermittelt werden, da jede Serverklasse über einen separaten ‘globalen’ Datenbestand verfügt und nur auf diesen zugreift (siehe Kapitel 1.4 und 3.4). Der Datensatz einer Klasse ist in eine ‘überschaubare’ Anzahl von Partitionen aufgeteilt, die einzeln spezifiziert werden können und gleichzeitig im Ausführungsmodell die Einheiten der Synchronisation darstellen. Die Betrachtung einzelner Partitionen setzt ein relativ grobes Granulat voraus, bei feinerer Aufspaltung der Daten sind statistische Angaben sinnvoller. Wenn beispielsweise der statische Datenbestand einer Serverklasse in 10 Partitionen zerfällt, so können diese einzeln genannt werden; wird er in 100.000 Teile gespalten, so ist nur noch relevant, wie viele Teile aus welchem Bereich bei einem Aufruf betroffen sind. Man beachte, daß die Partitionierung der Daten allein durch die Anwendung bestimmt wird und von Verteilung und Anzahl der Serverinstanzen völlig unabhängig ist.
- Eine Abschätzung der Unteraufträge, die während der Auftragsbearbeitung entstehen werden. Dabei ist nicht allein interessant, wieviel Aufrufe an welche Serverklassen gehen, sondern auch die zeitliche Verteilung der Unteraufrufe und die zu erwartenden Profile. Bei synchronen Unteraufrufen ist die dadurch entstehende Abhängigkeit zu vermerken.

Bei allen Angaben ist im Prinzip die Sicherheit bzw. statistische Relevanz mit anzugeben.

Es gibt diverse Möglichkeiten, die dynamische Beschreibung eines Auftrages zu generieren, und keine ist unmittelbar zu favorisieren. Für eine allgemeine Lastbalancierung ist es schwierig, Formeln zur Erstellung eines Auftragsprofils aufgrund der Aufrufparameter und Wissen über den Zustand der Serverinstanzen zu finden. Für die Serverinstanzen selbst ist es am einfachsten, verläßliche Angaben über den anstehenden Auftrag zu erzeugen. Zum Aufrufzeitpunkt steht die Serverinstanz jedoch noch nicht fest, sodaß in einer Art Ausschreibung alle Instanzen befragt werden müßten. Das ist ein erheblicher Zeit- und Nachrichtenaufwand (siehe etwa [Smith80]). Im HiCon Modell haben wir uns daher für die Alternative entschieden, in welcher der aufrufende Client aufgrund seiner Aufrufparameter und der statischen Auftragsbeschreibung ein Profil generiert. Dieses Profil übergibt er zusammen mit den Aufrufparametern der Lastbalancierungskomponente ③.

3.1.3 Profile für Auftragsgruppen

Mit den oben vorgestellten Größen lassen sich Reihenfolgebeziehungen oder Schleifen (Wiederholungen) innerhalb eines Aufrufes nur schlecht ausdrücken. Solche Angaben müssen jeweils für ganze Auftragsgruppen spezifiziert werden. Beziehungen zwischen Aufträgen können sehr wesentlich sein und werden daher im HiCon Modell berücksichtigt:

Zu einem Auftrag können die bei der Abarbeitung involvierten Unteraufträge in Form einer Menge von Teilaufträgen angegeben werden. Wir verwenden dazu im HiCon Modell keine hierarchische Aufrufkette sondern eine ‘flache’ Menge kooperierender Teilaufträge. Zu jedem Teilauftrag wird daher die Intensität der Kooperation (die Anzahl und Häufigkeit der Nachrichten) zwischen den verschiedenen Teilaufträgen sowie die entstehende Parallelität abgeschätzt. Zusätzlich können Reihenfolgeabhängigkeiten zwischen den Teilaufträgen spezifiziert werden.

Wie bereits am Anfang des Kapitels erwähnt, sind all diese Angaben abgekoppelt von der tatsächlichen Synchronisation durch die Serverinstanzen zur Laufzeit; es sind lediglich Vermutungen, welche die Lastbalancierung unterstützen.

Die Probleme dieses Ansatzes liegen zum einen bei der Beschreibung der Auftragsgruppe, die möglichst kompakt und einfach sein, aber dennoch komplexen Anwendungen verschiedenen Granulats gerecht werden soll, zum anderen in den hohen Laufzeitkosten für den Vorgang der Einplanung an sich. Diese Grenzen sind aus dem Gebiet des statischen Scheduling hinreichend bekannt. Man muß sich daher auf die Spezifikation weniger Zusammenhänge, die von besonderem Interesse für die Lastbalancierung sind, beschränken. Programmiersprachen-ähnliche filigrane Ablaufbeschreibungen, wie sie etwa für parallelisierende Compiler verwendet werden, sind für dynamische Lastbalancierung nicht geeignet. Daher betrachten wir zunächst, welche Charakteristiken einer Anwendung, die über einzelne, isolierte Auftragsprofile hinausgehen, für eine Vorausplanung wesentlich sein können:

- Die in Petrinetz- und Datenflußbeschreibungen bekannten Reihenfolge-Abhängigkeitsgraphen erscheinen im prozeduralen Modell als Wartezeiten auf synchrone Aufrufe. Interessant ist zu wissen, in welchem Maße die Ausführungszeit des (Haupt-) Auftrages von der Bearbeitungszeit eines Unteraufrufes bzw. von der Geschwindigkeit eines Kooperationspartners abhängt. Daraus kann der Balancierer die notwendigen relativen Verarbeitungszeiten - und damit Prioritäten, sobald der Auftrag zugewiesen wurde - abschätzen. In der statischen Lastbalancierung sind der Bottleneck-Path-Algorithmus und der Level-Algorithmus bekannte Verfahren (siehe dazu die Referenzen am Ende des Kapitels 1.1). Diese Reihenfolgebeziehungen können freilich nur für kleinere Gruppen diskreter Aufträge genutzt werden, nicht für gesamte komplexe Anwendungen, die zumeist Schleifen, Rekursionen und Verzweigungen enthalten.
- Für größere Mengen anstehender Aufträge kann Lastbalancierung die Anzahl von und das Verhältnis zwischen Lese- und Änderungszugriffen auf Kontext (-Partitionen) ausnutzen, um einen Kompromiß zwischen Parallelität und Kontextverwaltungsaufwand zu finden. Genauere Datenzugriffs-Angaben können jedoch meist -wenn überhaupt - erst zum Aufrufzeitpunkt gemacht werden.
- Wenn für eine Serverklasse innerhalb der Auftragsgruppe der vermutlich entstehende Grad an möglicher Parallelität abgeschätzt werden kann, so hat der Balancierer die Möglichkeit, die Server-Konfiguration diesen Anforderungen anzupassen. Mit möglicher Parallelität ist damit die Anzahl der Aufträge einer Klasse gemeint, die tatsächlich zugleich ausführungsbereit sind.

- Die Angabe des Nachrichtenaufkommens (Dauer und Intensität) zwischen einzelnen Teilaufträgen bzw. zwischen zwei Serverklassen schlechthin erlaubt die Berücksichtigung von schnellen Kommunikationsverbindungen bei der Instanzenauswahl bzw. bei der Server-Konfiguration (siehe [Ma82], [Berger87], [Bowen88], [Lo88] und [Lo88/2]).

Um die Beschreibung derartiger Informationen über Auftragsgruppen einfach zu halten, verzichten wir auf hierarchisch verschachtelte Auftragsgruppen sowie auf die explizite Behandlung von Schleifen. Eine Auftragsgruppe ist eine flache Gruppe von Auftragstypen, zu denen die Anzahl gegenseitiger Aktivierungen (Aufrufe) samt Kommunikationsintensität und Reihenfolgebeziehungen angegeben ist.

Kapitel 3.5.2 betrachtet, wie obige Informationen zur Lastbalancierung genutzt werden können.

3.2 Ressourcen-Lastmessung

Dynamische Lastbalancierungsverfahren verfolgen zur Laufzeit die Auslastung ihrer Ressourcen (Prozessoren, Platten und Verbindungsnetze bzw. -Busse) und benutzen diese Information zur Umverteilung laufender, zur Zuweisung neuer Aufträge und zur Verteilung der Daten ❷. Im HiCon Modell werden prinzipiell keine bereits in Bearbeitung befindlichen Aufträge mehr migriert. Daten werden in der Regel implizit durch Auftragszuweisung verschoben (siehe Kapitel 1.4, 3.4).

Für die Lastbalancierung ist weder die maximale Leistung einer Ressource noch die Systemlast von primärem Interesse, sondern die Zeit, die eine Instruktion bzw. eine Nachricht oder eine I/O-Operation derzeit auf der Ressource in Anspruch nehmen würde.

Der Aufwand zur Messung des aktuellen Zustandes während des laufenden Betriebes ist eine Zusatzbelastung sowohl für die Ressourcen als auch für den zentralen Balancierungsagenten, sodaß man sich auf die notwendigen Messungen beschränken muß. So verzichten die meisten Verfahren auf die Messung der durch einzelne Serverinstanzen induzierten Ressourcenbelastung, d.h. die Messung einzelner Prozesse. Üblicherweise wird die Zahl der ausführbaren Prozesse auf einem Prozessor, die Run Queue Length, als einzig entscheidender Lastfaktor angesehen (siehe [Ferrari86]), während die ‘CPU Busy Time’, die Auslastung der Plattencontroller sowie die Belastung des Netzwerkes für dynamische Lastbalancierung weniger relevant scheint.

Für eine Lastbalancierungsentscheidung ist die Leistung interessant, die ein Prozessor dem neuen Auftrag zur Verfügung stellen kann. Bei CPU-gebundenen Aufträgen schätzt man diese ab, indem man die Prozessorleistung durch die Anzahl der aktiven Aufträge (Run Queue Length) dividiert. Bei sozialer Lastbalancierung ist zusätzlich die Behinderung anderer Aufträge zu berücksichtigen (siehe Kapitel 3.5.1).

Üblicherweise läuft auf jedem Prozessor bzw. auf jedem Parallelrechner ein Messprozeß, der den lokalen Systemzustand in Form eines Lastvektors periodisch zum Lastbalancierer sendet. Um das Nachrichtenaufkommen zu minimieren sendet er nur im Falle einer signifikanten Laständerung. Welche Änderungen als signifikant angesehen werden, hängt dabei von der verwendeten Balancierungsstrategie und vom Anwendungsprofil ab. Daher kann die Lastbalancierung zur Laufzeit sowohl die Elemente des Lastvektors als auch die Messperioden variieren. Als Alternative zu separaten Messprozessen ist denkbar, daß die Serverinstanzen jeder Nachricht lokale Lastinformationen beifügen, was aber komplexer ist und starke Schwankung der Messintervallänge zur Folge hat.

3.3 Serverklassen und Instanzen

Im HiCon Modell verfügen wir über eine relativ statische Konfiguration von meist ‘single threaded’ Serverinstanzen. Es wird also nicht für jeden Serverklassen-Aufruf eine neue Instanz, d.h. ein Prozeß oder ein Thread kreiert, sondern im Normalfall wird der Auftrag an eine bestehende Serverinstanz vergeben ⑩. Da diese Instanz eventuell gerade einen Auftrag bearbeitet und noch weitere zur Bearbeitung in ihrer Warteschlange liegen, ist bei der Auftragszuweisung neben der momentanen und der zukünftig erwarteten Ressourcenauslastung auch der Zustand der Serverinstanzen zu berücksichtigen.

Jede Serverklasse hat einen statischen Datenbestand, der in Partitionen auf die Instanzen verteilt ist und zur Laufzeit durch Zugriffsanforderungen zwischen den Instanzen ausgetauscht bzw. kopiert wird. Diese Verteilung und Synchronisation wird vollständig durch die Datenzugriffsverwaltung gemeinsam mit der Datenverwaltung übernommen, sodaß die Serverinstanzen einer Klasse einen virtuellen gemeinsamen Datenspeicher sehen. Die statischen Daten einer Serverklasse können flüchtig oder nicht-flüchtig sein.

Der Algorithmus einer Serverinstanz hat im allgemeinen folgende Form:

Initialisierung

repeat {

Warte_auf_Auftrag
lokale Rechenoperationen
Unteraufrufe an Serverklassen
Warte_auf_Ergebnisse_von_Unteraufrufen
Zugriffe_auf_Partitionen_statischer_Klassendaten
Sende_Zwischenresultate_zurück
Warte_auf_weitere_Parameter
Sende_Resultat_zurück

} until Anwendung beendet

Die kursiv gedruckten Aktionen während der Durchführung eines Auftrages können dabei in beliebiger Reihenfolge und Wiederholungszahl auftreten. Unteraufrufe an andere oder an die eigene Serverklasse sind prinzipiell asynchron. Die aufrufende Instanz (Client) kann zu beliebigen Zeitpunkten auf die Resultate spezieller oder irgendwelcher Unteraufrufe warten. Weiterhin sind Serverklassen-Aufrufe ohne Resultat möglich. Dies bezeichnet man üblicherweise als vollständig asynchron.

Zwischen Client und Serverinstanz kann eine Session bestehen, indem die Serverinstanz Zwischenergebnisse sendet und der Client weitere Parameter an diese Instanz schickt ⑥. Solche Aufrufe werden gegenüber der Lastbalancierung explizit gekennzeichnet.

3.4 Datenzugriffsverwaltung

Im HiCon Modell besitzt jede Serverklasse statische Datenbestände, die von der Anwendung in Partitionen aufgeteilt werden (siehe auch Kapitel 1.4). Die Datenstrukturen und Einteilung in Partitionen wird durch die Anwendung bestimmt; der Datenbestand einer Serverklasse kann sich beispielsweise gemischt aus Hauptspeichervariablen und Dateien zusammensetzen. Die

Konvertierung der Daten von einem bzw. in ein über Nachrichten verschickbares Format geschieht durch von der Anwendung zur Verfügung gestellte Rückrufprozeduren.

Eine Partition ist zugleich die Einheit der Synchronisation, der Migration und der Replikation. Zugriffen auf eine Datenpartition muß stets eine Sperranforderung vorangehen ⑦. Dabei wird unterschieden, ob die Serverinstanz den Datensatz lesen oder modifizieren will. Zugleich mit dem Erwerb der Sperre wird bei Bedarf die Partition bzw. eine Kopie besorgt. Die Verwaltung und damit Zuständigkeit für Datenpartitionen wechselt dynamisch zwischen den Serverinstanzen. Das entspricht dem dynamischen Algorithmus in Shared Virtual Memory Systemen, wobei im HiCon Modell der Lastbalancierung die momentan zuständige Serverinstanz stets bekannt ist ❺. Im derzeitigen Prototyp übernimmt eine Instanz mit Anforderung einer Exklusivsperre auch die Verwaltung der Partition, bei Anforderung einer Lesesperre erhält sie lediglich eine Kopie. Ohne Lastbalancierung, welche die Kopien und den Aufwand für diese Kontextverwaltung miteinbezieht, ist diese Entscheidung nicht generell vorteilhaft; siehe etwa die Untersuchungen von [Weinmann92]. Im HiCon Modell kann das Wissen über die Verwaltungsstrategie erfolgreich zur Lastbalancierung genutzt werden.

Da der Freigabezeitpunkt von Sperren auf Datenpartitionen durch die Anwendung bestimmt wird, sind verschiedene Stufen der Konsistenzerhaltung möglich, wie z.B. Operationssperren, Cursor-Stabilität oder serialisierbare Transaktionen.

Das HiCon Konzept zur Synchronisation des Zugriffs auf globale Daten und der Verteilung globaler Daten in heterogenen Systemen ermöglicht die Balancierung datenintensiver Anwendungen und erfüllt die von der Objektorientierung geforderte Kapselung von Daten durch Serverklassen.

3.5 Die exekutive Lastbalancierung

Bevor wir die einzelnen funktionellen Komponenten der exekutiven Lastbalancierung betrachten, wollen wir den prinzipiellen Mechanismus mit den zugehörigen Datenstrukturen vorstellen.

Der Lastbalancierungsmechanismus benutzt eine Tabelle, den sogenannten Systemfahrplan, um die durch laufende und geplante Aufträge erwartete bzw. gemessene Systembelastung und Zuordnung von Aufträgen zu Instanzen und Ressourcen zu verwalten ❶. Dies ist freilich nur eine konzeptionelle Sichtweise, in Wirklichkeit ist diese Information über diverse Prozesse im System verteilt. Das Hauptproblem einer solchen Datenbasis besteht darin, die unterschiedlichen, aber nicht unabhängigen Objekte wie Aufträge, Serverinstanzen, Datenbestände und Ressourcen geeignet darzustellen sowie die verschiedenen Zustandsübergänge wie Zeitintervalle, Auftragsbearbeitungsdurchlauf und logische Reihenfolgeabhängigkeiten auf möglichst wenige Dimensionen abzubilden.

Im HiCon Modell wird der Systemfahrplan primär über den Achsen Server, Ressourcen und Aufträgen aufgespannt. Abbildung 3 soll die Struktur veranschaulichen.

Für jede Serverklasse verwaltet die Lastbalancierung eine zentrale Auftragswarteschlange. Sie enthält diejenigen Serverklassen-Aufrufe, die von Clients abgesandt aber noch keiner Serverinstanz endgültig zugewiesen wurden. Zu jedem Auftrag in dieser Schlange existieren Abschätzungen über seinen Ablauf und Präferenzen für bestimmte Serverinstanzen. Zu gegebener Zeit weist die Balancierung diesen Auftrag einer festen Serverinstanz zu und verlagert ihn dabei in die Warteschlange der Instanz. In den Instanzenwarteschlangen werden zusätzliche Informationen wie Zustand der Bearbeitung und gemessene Ressourcenauslastung sowie Zeitabschätzungen verwal-

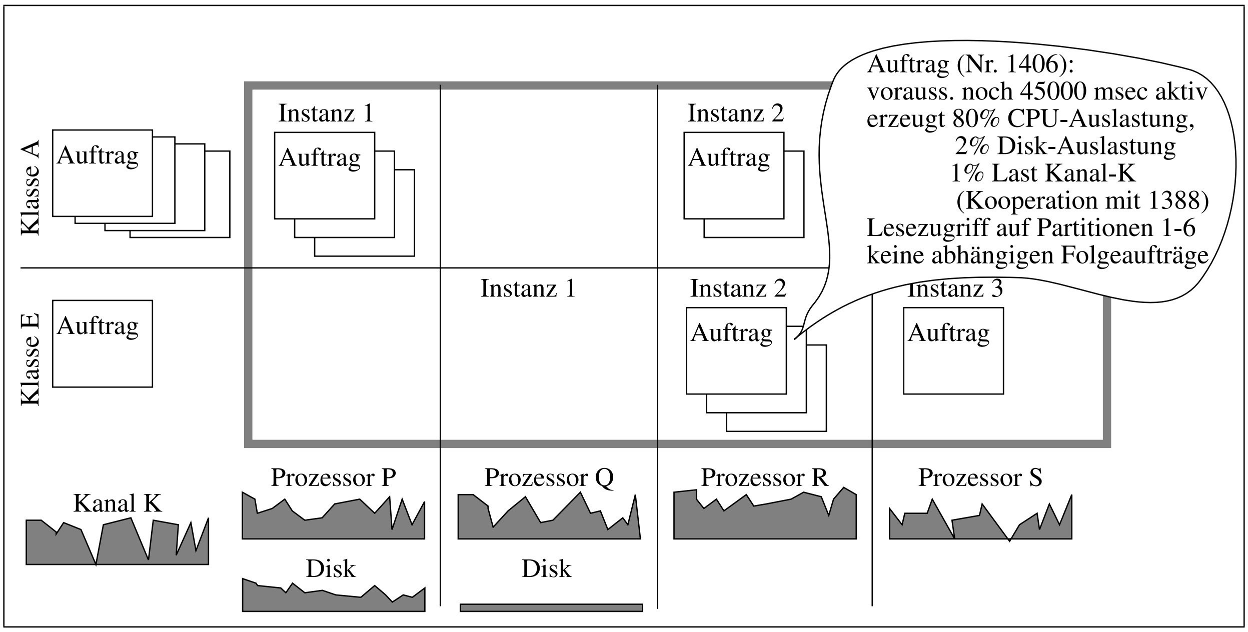

Abbildung 3 : Auftrags- und Systemsicht der exekutiven Lastbalancierung.

tet. Die Reihenfolge und der Zeitpunkt der Bearbeitung von Aufträgen in den Klassen- und Instanzen-Warteschlangen hängt dabei ebenso wie Auswahl relevanter Informationen von der derzeit aktiven Lastbalancierungsstrategie ab.

Zugewiesene Aufträge können im HiCon-Modell bei der Serverinstanz selbst gelagert werden (siehe auch Abbildung 4). So können die Instanzen nach Ausführung eines Auftrags unmittelbar mit der Bearbeitung des nächsten beginnen. Serverinstanzen können allerdings der Lastbalancierer das Ende einer Bearbeitung als - freiwillige - Zustandsmitteilung senden, damit die Diskrepanz zwischen Systemfahrplan und tatsächlichem Systemverhalten nicht allzu groß wird.

3.5.1 Auftragsverwaltung - Einzelaufträge

Die Auftragsverwaltung der exekutiven Lastbalancierung besteht im wesentlichen aus vier Methoden, die jeweils aus dem Strategienkatalog bezogen werden. Die erste Methode, vergleichbar mit den in der Literatur vorgestellten Sender-initiierten Balancierungsstrategien, nimmt neue Serverklassen-Aufrufe entgegen ③, bewertet sie und weist sie evtl. vorläufig oder endgültig einer Serverinstanz zu ④. Die zweite Methode, den sogenannten Receiver-initiierten Verfahren verwandt, agiert auf Änderungen von Serverinstanz-Zuständen hin, beispielsweise wenn eine Serverinstanz einen oder alle Aufträge beendet hat ⑨. Sie versucht dann, vorrätige Aufträge an diese und andere geeignete Serverinstanzen zu vergeben. Die dritte Methode wird bei signifikanten Änderungen in der Systemauslastung ❷ oder Datenverteilung ❺ sowie nach Ablauf einer Zeitschranke angewandt und versucht vorrätige Aufträge zuzuweisen bzw. neu zu bewerten. Zeitschranken können beispielsweise gesetzt werden, wenn die Zuordnung eines Auftrages nur eine gewisse Zeit hinausgezögert werden soll. In Kapitel 3.5.2 werden wir die vierte Methode vorstellen.

Wir wollen daher in diesem Abschnitt nur die prinzipielle Struktur der Auftragsverwaltung betrachten (siehe Abbildung 4).

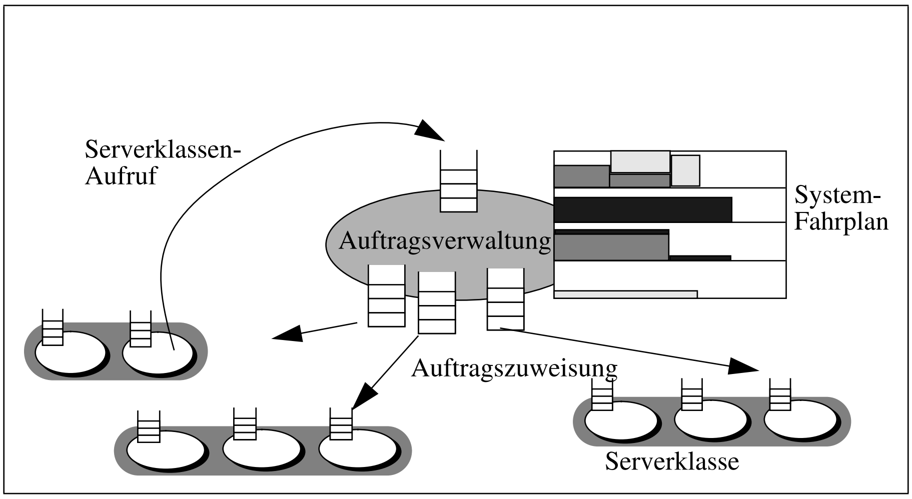

Abbildung 4 : Modell der Auftragsverwaltung.

Die Auftragsverwaltung bekommt durch ihren Eingangspuffer neue Aufträge, Auftragsgruppen, Fertigmeldungen und Server-Zustandsmeldungen, Datenverschiebungen und Änderungen der Ressourcenauslastung mitgeteilt. Sie paßt daraufhin den Systemfahrplan an und aktiviert eine Lastbalancierungsmethode.

Viele Lastbalancierungsumgebungen benutzen ein Modell, bei dem sich eine Serverinstanz frei meldet und daraufhin einen neuen Auftrag bekommt. Das hat den Vorteil der möglichst späten Bindung des Auftrags an eine Serverinstanz, bringt aber das Problem mit sich, daß die Serverinstanz nach Abarbeitung des Auftrages stets solange Leerlauf hat, bis die Lastbalancierung mit Sendung eines weiteren Auftrages reagiert. Es ist erträglich, solange auf jedem Prozessor ständig mehrere Aufträge parallel bearbeitet werden, da diese in der Wartezeit der einen Instanz die Knotenleistung nutzen können. In jedem Falle aber müßte der Lastbalancierer die Leerlaufzeiten berücksichtigen.

Die Warteschlangen an den einzelnen Instanzen bieten weiterhin die Möglichkeit, Aufträge gruppenweise zu behandeln. Der Lastbalancierer kann jeweils einen Puffer voll von Aufträgen am Stück an eine Serverinstanz senden. Diese Optimierung wird oft bei hoher Parallelität und feinem Auftragsgranulat verwendet (siehe etwa [Schiele91]), wenn die Nachrichtenkosten sowie der Verwaltungsaufwand pro Auftrag durch den zentralen Scheduling-Prozeß relativ hoch sind. Man nimmt hier offensichtlich eine Vergröberung des von der Anwendung vorgegebenen Granulats vor. Da im HiCon Konzept das Granulat vollständig dem Anwender überlassen bleibt, wird diese Optimierung derzeit nicht angewandt.

Zentrale Klassen-Auftragswarteschlangen ermöglichen es dem Lastbalancierer weiterhin, die Parallelität in der Bearbeitung zu kontrollieren: Rekursive Anwendungen erzeugen oft eine exponentiell wachsende Anzahl parallel ablauffähiger Aufträge. Die Bereitstellung bzw. dynamische Erzeugung einer entsprechenden Anzahl von Serverinstanzen wäre ineffizient und unrealistisch (siehe Kapitel 1.4).

Die sofortige Zuweisung entstehender Aufträge an Serverinstanzen führt zu langen lokalen Warteschlangen, was dem Balancierungsprinzip der möglichst späten Zuweisung widerspricht. Bis nämlich der Auftrag zur Bearbeitung an die Reihe kommt, ist die Serverinstanz möglicherweise nicht mehr die für ihn bestgeeignete.

Jeder Auftrag in der Warteschlange bekommt eine Bewertung, welche Instanz zu seiner Bearbeitung in welchem Maße geeignet ist. Diese Bewertungsfunktion hängt von der konkreten Balancierungsstrategie ab; prinzipiell wird der Auftrag einer Serverinstanz zugewiesen, sobald sichergestellt ist, daß sie aus derzeitiger Sicht den Auftrag am 'besten' bearbeiten wird. 'Sicher sein' bedeutet, ausreichend genaue Angaben zu haben, daß die Serverinstanz innerhalb eines Toleranzbereiches die bestgeeignete ist. Als Bewertungskriterium gilt zweifellos die minimale Antwortzeit, man muß jedoch unterscheiden, ob die Lastbalancierung den Auftrag isoliert betrachtet oder die Gesamtheit der unter ihrer Regie laufenden Aufträge berücksichtigt. Letzteres wird auch als soziale Balancierung bezeichnet. Im ersten Fall liefert die Bewertungsfunktion die vermutete Antwortzeit für diese Instanz zurück, im zweiten Fall eine Linearkombination der Antwortzeit auf dieser Instanz und der dabei entstehenden Verzögerungen, die anderen Aufträgen durch Benutzung derselben Ressourcen entstehen (wir sprechen hier noch nicht von Reihenfolgeabhängigkeiten). Die Summanden können nach der Priorität der Aufträge gewichtet werden.

Die Fähigkeit zur Migration laufender Aufträge erlaubt es, Aufträge mit relativ grober Vorplanung zuzuweisen und Fehlplanungen später durch Migration zu korrigieren. Das würde obige Probleme entschärfen; Auftragsmigration wird jedoch im HiCon Modell nicht unterstützt, da sie in heterogenen Systemen und bei Serverinstanzen mit statischen Datenbeständen sehr aufwendig ist (siehe Kapitel 1.4).

Zuletzt soll noch die Möglichkeit der ungleichen Lastaufteilung innerhalb einer Ressource erwähnt werden, welche wir im HiCon Modell noch nicht explizit nutzen. Serverinstanzen können je nach Wichtigkeit ihres aktuellen Auftrages mit unterschiedlichen Prozessprioritäten ablaufen. Um einen eiligen Auftrag durchzusetzen, auf den etwa viele andere warten (siehe Kapitel 3.5.2), kann es sinnvoll sein, andere laufende Aufträge zu bremsen bzw. warten zu lassen. Bislang teilen sich im HiCon Konzept alle laufenden Aufträge die gemeinsamen Ressourcen gleichmäßig untereinander auf. Der Lastbalancierer kann jedoch wichtigere Aufträge vorrangig bzw. an 'schnellere' Instanzen vergeben, oder er kann Prozessoren, welche wichtige Aufträge abwickeln, von weiteren Aufträgen verschonen.

3.5.2 Auftragsverwaltung - Auftragsgruppen

Neben der oben vorgestellten Aufgabe der Lastbalancierung bietet das HiCon System die Möglichkeit, zur Laufzeit zusammenhängende Gruppen von Aufträgen (Anwendungen) anzumelden ③. Durch Reservierung und Bereitstellung von Ressourcen und einer vorläufigen Zuweisung der einzelnen Aufträge unter Ausnutzung des Wissens über den Gesamtablauf der Gruppe können wir wichtige Aspekte wie wiederholte bzw. parallele Aufrufe einer Serverklasse, Lokalität in aufeinanderfolgenden Datenzugriffen sowie Reihenfolgeabhängigkeiten zwischen Einzelaufträgen in die Balancierung einbeziehen, was bei der isolierten Betrachtung einzelner Aufrufe noch nicht möglich ist (siehe auch Kapitel 3.1.3).

Diese Angaben bewirken lediglich eine vorläufige Einplanung der zu erwartenden Aufträge sowie eventuell Änderungen der Server-Konfiguration. Die endgültige Zuweisung der Aufträge geschieht nach wie vor erst zum tatsächlichen Aufrufzeitpunkt. Wir folgen damit dem in der dynamischen Balancierung wichtigen Prinzip der möglichst späten Zuordnung und können außerdem Fehler und Ungenauigkeiten in den Profilangaben tolerieren.

Das Optimierungskriterium des hier vorgestellten Balancierungsmechanismus ist die möglichst schnelle Ausführung eines Gesamtauftrages. Dies geschieht unter Berücksichtigung anderer unabhängige Aufträge, da wir soziale Lastbalancierungsmethoden einsetzen. Bei Auftragsgruppen ist also nur die Antwortzeit des Hauptauftrags - und damit des Gesamtauftrags - zu minimieren; alle Teilaufträge sind zunächst unkritisch und ihre Dringlichkeit ergibt sich allein aus den Abhängigkeitsbeziehungen zum Hauptauftrag.

Die Anmeldung von Auftragsgruppen kann zur Laufzeit vor Absendung der einzelnen Aufträge geschehen. Daraufhin kann der Balancierungsmechanismus entsprechende Kapazitäten reservieren - etwa durch Rekonfiguration der Serverinstanzen - und eine vorläufige grobe Zuweisung der Einzelaufträge vornehmen. Verfahren, die auf dem Bottleneck-Path Algorithmus aufsetzen, werden die Aufträge entlang des kritischen Pfades markieren und alle anderen Aufträge so verteilen, daß sie die wichtigen Aufträge möglichst wenig stören. Aus dem Level-Algorithmus lassen sich Methoden ableiten, die darauf achten, daß Aufträge mit höherem Level vor denen mit niedrigeren bearbeitet werden.

In praktisch allen Arbeiten über dynamische Lastbalancierung werden Information über Abhängigkeiten, Kooperation und Parallelität innerhalb von Auftragsgruppen nicht genutzt, um die Komplexität und die Selbstkosten der Balancierung gering zu halten. Derlei Angaben sind ja ohnehin nur wertvoll, wenn sie eine gewisse Genauigkeit und Wahrscheinlichkeit aufweisen. Da die HiCon-Lastbalancierung jedoch durch die strategische Ebene in der Lage ist, für bestimmte Anwendungsklassen spezialisierte Balancierungsmethoden einzusetzen, sollten auch solche Beschreibungsformen im Modell möglich sein.

Anhand eines einfachen Beispiels wollen wir die Wichtigkeit der Berücksichtigung von Reihenfolgebeziehungen demonstrieren. In Abbildung 5 sind vier vom Aufwand her identische Teilaufträge (Laufzeit t Sekunden auf leerem Prozessor) in einem Präzedenzgraphen dargestellt, die auf zwei gleichartigen Prozessoren ausgeführt werden sollen. Wenn B und C auf einen, A auf den anderen Prozessor gelegt werden, erreicht man die Laufzeit 2*t. Führt man hingegen A und B auf einem, C auf dem anderen aus, so wird die Applikation 3*t Sekunden benötigen.

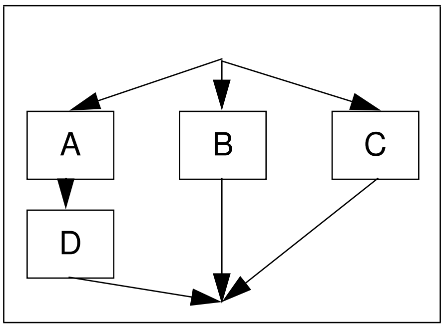

Abbildung 5 : Reihenfolgebeziehungen zwischen Teilaufträgen.

3.5.3 Konfigurations- und Datenverwaltung

Die exekutive Lastbalancierung kann zur Laufzeit neue Serverinstanzen kreieren ⑩, bestehende terminieren und die Datenbestände, welche ja grundsätzlich auf die Serverinstanzen verteilt sind, umverteilen ❻⑦. Neben der Plazierung der (Original-) Datensätze kann sie auch Kopien verteilen und Kopien ungültig machen. Bei solchen Entscheidungen spielen nicht nur Profile von Auftragsgruppen eine Rolle. Konfigurationsänderungen können auch durch längerfristige Beobachtungen aggregierter Daten ausgelöst werden. Wir wollen es hier nur anhand weniger Beispiele motivieren:

- Wenn eine Klassen-Auftragswarteschlange ständig sehr lang ist, sollten eventuell neue Instanzen aufgesetzt werden. Umgekehrt können Instanzen gelöscht werden, wenn bei den Instanzen einer Serverklasse die mittlere Wartezeit auf einen neuen Auftrag groß ist.
- Wenn die Wartezeit auf Daten im Verhältnis zur eigentlichen Rechenzeit der Instanzen zu groß ist, so scheint das Anlegen von Lesekopien aufgrund des hohen Anteils an Schreibzugriffen unwirtschaftlich (siehe [Weinmann92]) oder das Balancierungsverfahren achtet bei der Auftragszuweisung zu wenig auf die Datenlokalität. Im Zweifelsfall empfiehlt sich eine Reduktion der Serverinstanzen.
- Wenn eine Serverinstanz eine Datenpartition lesen möchte, so genügt es im Prinzip, ihr eine Kopie zu senden. Wenn aber die Wahrscheinlichkeit für einen nachfolgenden Schreibzugriff hoch ist, mag es sinnvoller sein, ihr das Original zu geben. Hat andererseits eine Instanz auf einem Datensatz gearbeitet und kaum Aussichten, ihn in naher Zukunft noch einmal zu verwenden, so kann man ihr die Daten bzw. die Kopie sofort entziehen.

3.5.4 Hierarchische Verteilung der Lastverwaltung

In völlig dezentralen Lastbalancierungsverfahren sprechen die Server die Auftragsverteilung selbstständig untereinander ab. Der Server, welcher neue Unteraufträge hat oder überlastet ist, versucht Aufträge an andere Server abzugeben. Unterbelastete Server sind bestrebt, von anderen Aufträge zu übernehmen. Man unterscheidet oft (siehe [Casavant88], [Lin92]), ob nur diese Entscheidung dezentral ist oder auch die Zustandsinformationen über die Server verteilt sind, ein Server also nur die Lastsituation seiner Nachbarn kennt. Im HiCon Modell wird sowohl die Zustandsinformation als auch die Entscheidung in einer Komponente zentral verwaltet (siehe auch Kapitel 1.5).

Eine zentrale Informations- und Entscheidungsstelle bietet den Vorteil einer global optimalen Balancierung. Dezentrale Verfahren optimieren nur lokal und basieren auf relativ einfachen Meß- und Entscheidungsgrößen; die einzelnen Instanzen sind nicht in der Lage, komplexere Zusammenhänge wie Reihenfolgebeziehungen oder soziale Lastverteilung zu erfassen. Die Verwaltung zentraler Auftragsschlangen ermöglicht weiterhin die späte Zuweisung von Aufträgen ohne zugeteilte oder laufende Aufträge migrieren zu müssen.

Zentrale Balancierung hat aber den Nachteil, selbst zum Engpaß zu werden. Die Balancierungskomponente muß sehr schnell viele Informations- und Auftragsnachrichten bearbeiten; wenn sie überfordert ist, arbeitet sie mit veralteten Informationen bzw. zögert die Auftragsbearbeitungen hinaus, da sie nicht schnell genug Zuweisungsentscheidungen treffen kann. Auch aus Gründen der Fehlertoleranz ist eine zentralisierte Lösung bedenklich.

In großen Systemen ist also, sofern man einen logisch zentralen Ansatz favorisiert, eine hierarchische Abstraktion der Lastbalancierung notwendig. Lokale Komponenten balancieren autonom ein - sinnvollerweise physisch zusammenhängendes - Teilsystem. Sie betrachten den übergeordneten Balancierer als eine weitere Komponente ihres Teilsystems, der eine abstrakte Sicht seines Zuständigkeitsbereichs zeigt. Eine Balancierungskomponente kann also von ihrer übergeordneten wie von einer Unterkomponente Aufträge und Lastinformationen erhalten und an diese abgeben.

Im HiCon Modell erfolgt die Abstraktion auf Ebene der Instanzen: Ein Balancierer bietet Anderen eine Instanz pro Serverklasse, deren Leistung dem Mittel- oder Bestwert seiner lokalen

Instanzen entspricht. Sein Hardware-Teilsystem sowie seine lokalen Aufträge verbirgt er vollständig vor den anderen Balancierungskomponenten.

Hierarchische Balancierung funktioniert nur dann effektiv, wenn eine gewisse Lokalität vorliegt und von der Balancierung berücksichtigt wird. Lastbalancierer verteilen Aufträge - und damit auch Daten - möglichst lokal. Die 'Pseudo'-Instanz des übergeordneten Balancierers muß daher um einiges besser sein, damit sie gewählt wird. Wenn aufgrund der Anwendung keinerlei Lokalität in den Datenzugriffen vorhanden ist und sich auch nicht durch Datenmigration oder Replikation einstellt, so ist die hierarchische Balancierung ungünstig. Abbildung 6 zeigt einen solchen Fall, bei dem Serverinstanzen ständig auf Daten zugreifen, die bei weit entfernten Instanzen lagern; die grau gefüllten Ellipsen stehen für Lastbalancierungskomponenten. Allgemein lassen sich Änderungsoperationen auf einem gemeinsamen Datensatz nicht beliebig verteilen und die Lastbalancierung sollte dies auf höherer Ebene feststellen.

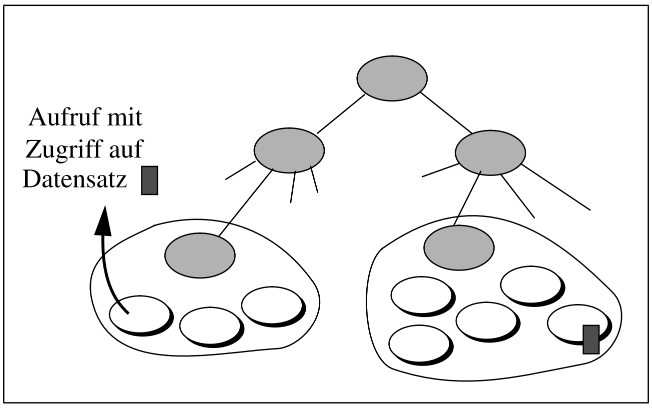

Abbildung 6 : Änderungsoperation auf nichtlokalen Daten.

Alternativ zur oben vorgestellten homogenen Hierarchie könnte man sich vorstellen, einzelne Funktionen der Lastbalancierung grundsätzlich dezentral zu gestalten. Beispielsweise könnten die Instanzen die Datenverwaltung direkt untereinander, ohne den Umweg über eine Balancierungsinstanz, regeln. Der für Auftragszuweisung verantwortliche Balancierer hätte dann aber kein vollständiges Wissen mehr über die Datenverteilung und kann sie entsprechend schlecht einbeziehen. Zudem legt man sich durch diese funktionale Trennung auf eine bestimmte Systemgröße fest, während die homogene Hierarchie - eine gewisse Lokalität in der Anwendung vorausgesetzt - beliebig skalierbar bleibt. Das Problem solcher funktionalen Trennung in Ebenen wurde ja bereits in Kapitel 1.1 angesprochen.

4 Simulation und Realisierung des HiCon Modells

Eine Lastbalancierungsumgebung nach dem HiCon Modell wurde als Prototyp realisert; außerdem wurde ein Simulator entwickelt, der uns eine schnellere und einfachere Evaluierung der vorgeschlagenen Konzepte ermöglicht. Wir werden diese Implementierungen im folgenden kurz betrachten.

4.1 Simulation des HiCon Modells

Um die prinzipielle Tauglichkeit der entwickelten Lastbalancierungsverfahren zu untersuchen und ein Gefühl für die verschiedenen Faktoren, ihr Zusammenspiel und ihre Auswirkungen zu bekommen, wurde ein Simulator zur Nachbildung des HiCon-Ausführungsmodells entwickelt (siehe [Staib92]). Für eine Simulation definiert man die Ressourcen seines Systems, bestehend aus Prozessoren, externen Speichermedien und Verbindungskanälen. Dazu gibt man eine anfängliche Server-Konfiguration vor. Die eigentliche Anwendung wird durch Auftragsprofile für die Serverklassen definiert. Diese Profile sind bisher statisch, eine Erweiterung auf dynamische Auftragscharakterisierung ist im Entstehen. Nun kann man aus einer Reihe von Lastbalancierungsverfahren dasjenige wählen, welches eingesetzt werden soll. Im derzeitigen Simulator besteht jede Strategie aus Methoden für zwei Eingriffspunkte: die Entstehung eines Auftrags und die Zustandsänderung einer Serverinstanz.

Der Benutzer kann die Bearbeitung graphisch mitverfolgen und erhält eine abschließende Gesamtauswertung. Während der Simulation wird der Verlauf der Anwendung, d.h. die Abarbeitung der Funktionsausführungen in Form von Serverklassen-Aufrufen, Unteraufrufe und Datenzugriffe sowie die Auslastung aller Ressourcen detailliert dargestellt. Abbildung 7 zeigt die Bildschirmansicht während eines Simulationslaufs. Im linken Teil wird für jeden Prozessor, jede Platte und jeden Kanal die gemittelte prozentuale Nutzungszeit (busy time) aufgetragen. Der rechte obere Teil zeigt den Verlauf von Serverklassen-Aufrufen über der Zeitachse. Jeder zusammenhängende horizontale Balken steht für einen Serverklassen-Aufruf, wobei die Schraffierungen verschiedene Phasen der Bearbeitung, wie Rechenzeit, Wartezeit auf Plattenzugriffe und synchronisationsbedingte Wartezeiten auf globale Datensätze bzw. auf Unteraufrufe darstellen. Im unteren Teil von Abbildung 7 geben diese Schraffierungen den entsprechenden momentanen Zustand der - durch Ellipsen symbolisierten - Serverinstanzen an. Der senkrechte Balken in der Mitte einer Serverinstanz gibt an, welcher Anteil des momentanen bearbeiteten Auftrages bereits erledigt ist. In dem Kasten der rechten Hälfte jeder Instanz werden die derzeit lokal vorhandenen Datenpartitionen bzw. Kopien dargestellt. Schließlich wird die aktuelle Länge der Klassen-Auftragswarteschlange durch eine Reihe von Rechtecken über den Instanzen der Klasse symbolisiert.

In einem Anwendungsbeispiel mit relativ feinkörniger Parallelität, der Suche nach einem kürzesten Weg in einem Graphen, zeigte [Staib92] bereits positive Resultate für einfache Lastbalancierungstechniken nach dem HiCon-Konzept. Wir wollen jedoch nicht hier, sondern im Rahmen des *Loadman*-Prototyps in Kapitel 4.2.4, näher auf diese Experimente eingehen.

4.2 Loadman - Prototyp eines Lastbalancierers nach dem HiCon Modell

Die Lastbalancierungsumgebung *Loadman* ist eine prototypische Realisierung der in diesem Bericht vorgestellten Ansätze, um ein weites Spektrum datengestützter Anwendungen effizient auf heterogenen parallelen und verteilten Systemen zu bearbeiten. Wir beschreiben kurz das Design, die bisher implementierten Balancierungsverfahren und vorläufige Messergebnisse.

4.2.1 Systemstruktur und Anwendungsschnittstelle

In *Loadman* werden Lastbalancierungsinformationen und -Entscheidungen bislang durch einen zentralen Prozeß verwaltet, der sowohl die Zuweisung - Ort und Zeitpunkt - als auch die Einpla-

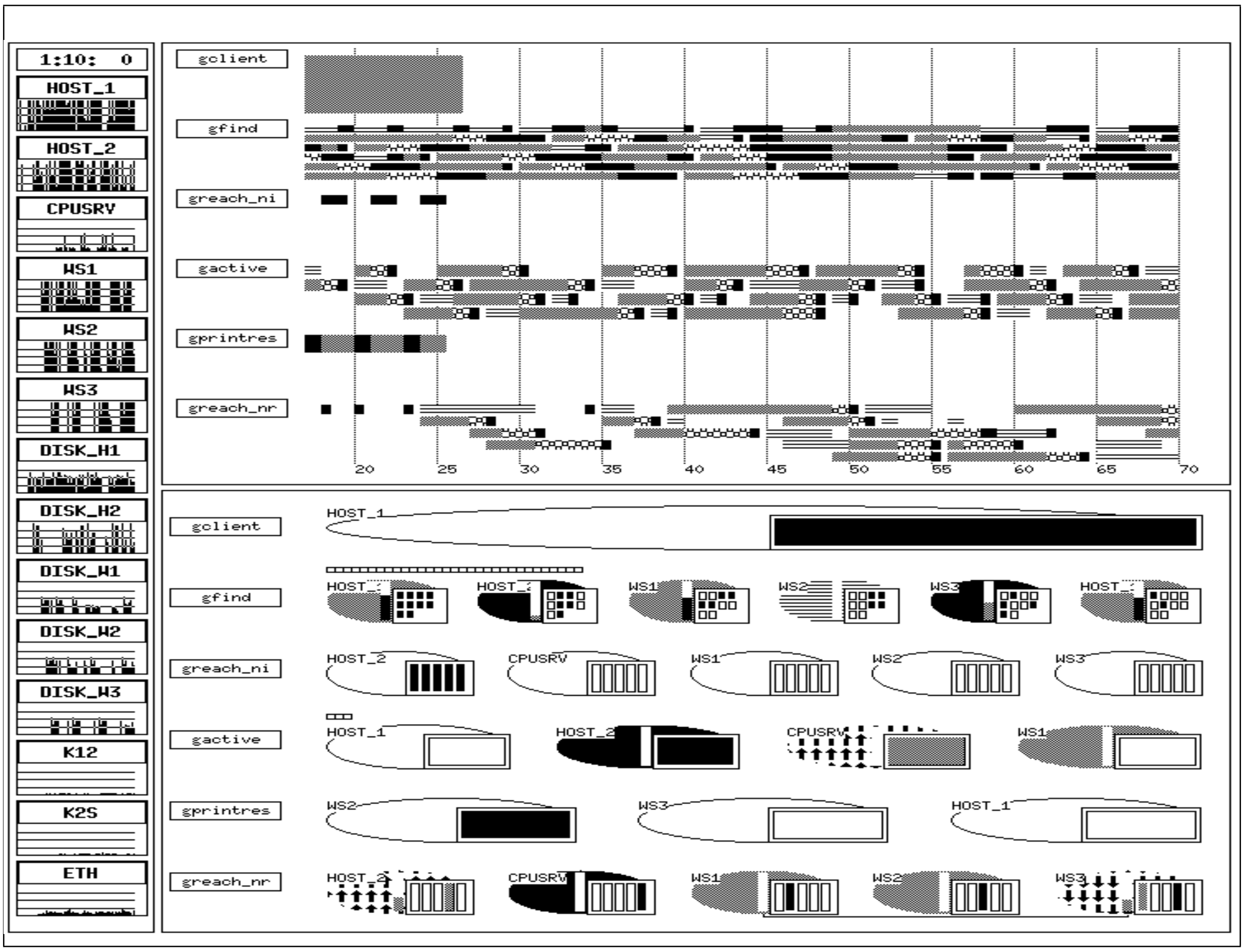

Abbildung 7 : Bildschirmansicht während eines Simulationslaufes.

nung von Auftragsgruppen durchführt. Dieser Prozeß soll später gemäß Kapitel 3.5.4 physisch verteilt werden.

Wie in Abbildung 8 veranschaulicht besteht die Balancierungskomponente derzeit neben dem zentralen Prozeß aus einem Lastmessungsagenten auf jedem teilnehmenden Prozessor und an die Serverinstanzen angebundenen Bibliotheksfunktionen. Die eigentliche Anwendung ist in Form eines Prozesses je Serverinstanz auf das System verteilt und verkehrt über die Bibliotheksfunktionen mit der Balancierungskomponente.

Die eigentliche Interprozeß-Kommunikation wird mittels angebundener Funktionen der *Coin*-Library durchgeführt. Diese Nachrichtenschnittstelle wählt automatisch das schnellste verfügbare Nachrichtenprotokoll. So werden Nachrichten innerhalb eines Hauptspeichers, wie etwa auf einer UNIX-Workstation oder auf einem shared memory Multiprozessor über *pipes*, innerhalb von Parallelrechnern durch proprietäre Mechanismen, wie *Guardian*s Kommunikationssystem auf Tandem-Rechnern und zwischen verschiedenen Rechnern mit Hilfe des *TCP* Protokolls versandt. Die transparente Nutzung dieser verschiedenen Nachrichtensysteme ist für die Lastbalancierung in heterogenen Systemen unerläßlich, da unterschiedlich effiziente Kommunikationswege und damit verschiedene ‘Entfernungen’ zwischen Prozessen einen wesentlichen Faktor für die Lastverteilung darstellen können. Die prinzipielle Beschränkung auf ein allgemeines, langsames Medium wie *TCP* ist nicht für alle Anwendungsgebiete sinnvoll.

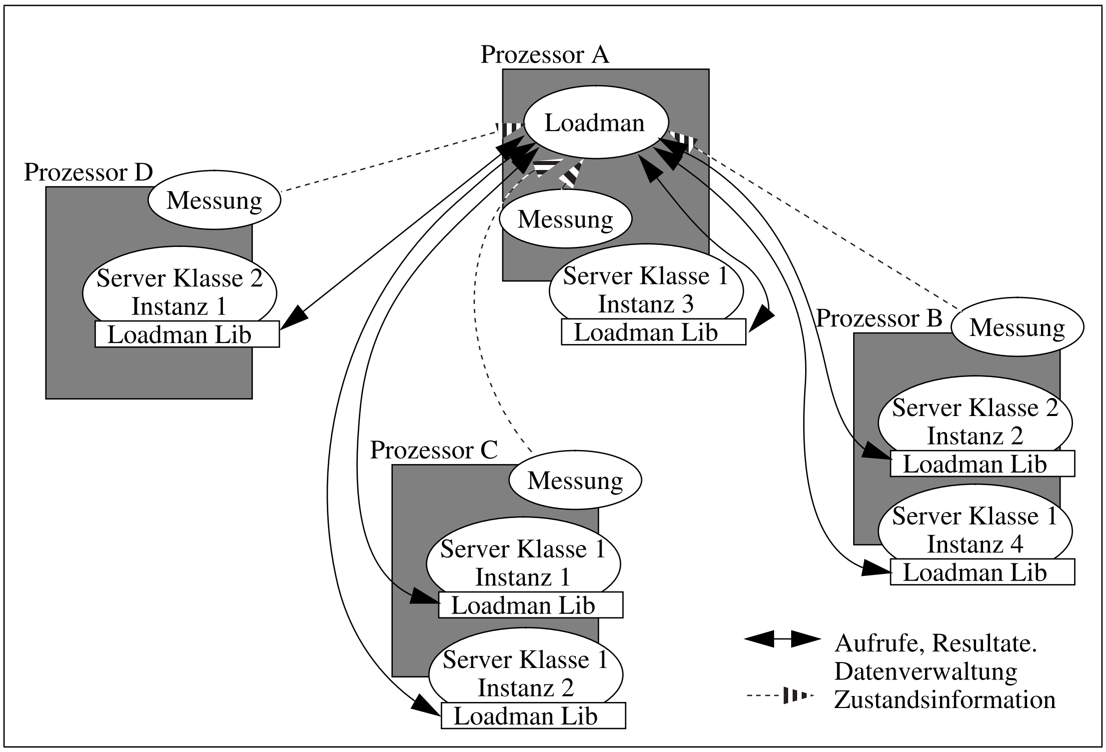

Abbildung 8 : Prozeß-Struktur des Prototypen.

Loadman ermöglicht derzeit verteilte Anwendungen auf einer aus Sun-, DEC-, HP840- UNIX Workstations, Sequent shared-memory-Multiprozessoren sowie Tandem shared-nothing-Parallelrechnern beliebig gemischten Basis. Multiprozessoren mit gemeinsamem Hauptspeicher werden von *Loadman* derzeit als je ein Prozessor betrachtet. Da die Lastbalancierung im *HiCon* Modell relativ grobgranular konzipiert ist und auf verteilten Ressourcen basiert, nutzt man hier besser die vorhandenen Scheduling-Strategien, meist eine gemeinsame Run-Queue.

Dynamische Lastmessung erfolgt auf UNIX-Plattformen durch Ablesen bestimmter Datenstrukturen aus dem Betriebssystemkern, auf *Guardian*-Rechnern durch das *Measure*-Subsystem. Die statische Beschreibung des Systems, mit Prozessorleistungen, Platten, Netzstruktur und -Leistung, wird in einer Konfigurationsdatei spezifiziert.

Loadman kann wahlweise mit oder ohne graphischer Oberfläche ablaufen. Abbildung 9 zeigt die Bildschirmansicht einer laufenden Anwendung. Die graphische Darstellung informiert zur Laufzeit über den Verarbeitungszustand: die Server-Konfiguration wird durch ein Rechteck pro Serverinstanz symbolisiert, wobei die Klassenzuordnung in der Horizontalen, die Ressourcenzuordnung durch die Vertikale erfolgt. Die momentanen Längen der Klassen- und Instanzen-Auftragswarteschlangen werden durch waagrechte Balken über dem Serverklassen-Namen bzw. im oberen Teil der Serverinstanzen angezeigt. Im unteren Teil einer Serverinstanz wird die Verteilung der Datenpartitionen und deren Replikate gezeigt. Die Auslastungsanzeige der Ressourcen beschränkt sich derzeit auf die jeweilige Run Queue Length über dem Rechnernamen.

Diese Darstellung hat sich als geeignet herausgestellt, um Anwendungen bezüglich ihres Kontext- und Auftragsgranulats zu justieren, geeignete Server-Konfigurationen zu ermitteln und die Auswirkungen verschiedener Balancierungsstrategien zu beobachten. Für exakte Zeitmessungen ver-

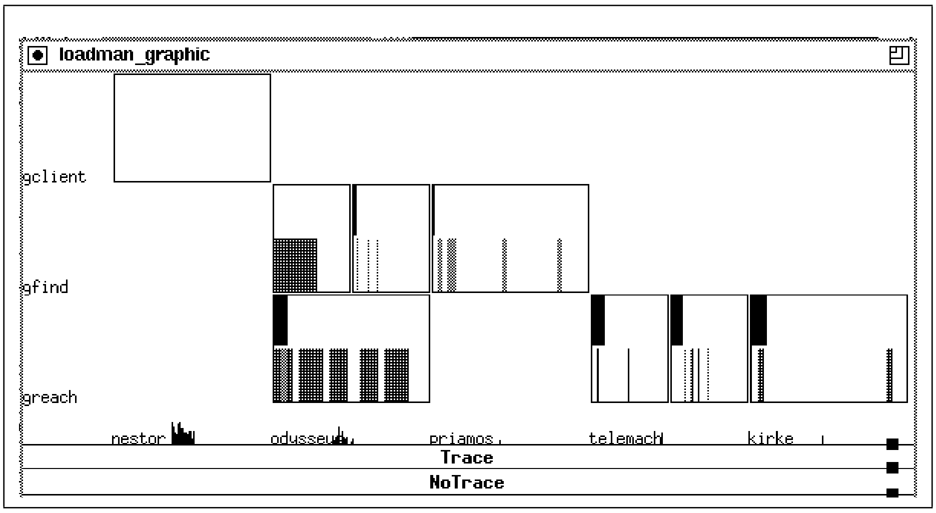

Abbildung 9 : Bildschirmansicht des Prototypen während des Ablaufs einer Anwendung.

zichtet man freilich auf die graphische Oberfläche, weil es gewisse Störungen und Verzerrungen verursacht. Während oder nach Ablauf einer Anwendung kann der Benutzer statistische Informationen über das Verhalten der Serverklassen abrufen: die mittlere Leerlaufzeit der Instanzen einer Klasse zwischen zwei Auftragsbearbeitungen, die mittlere Bearbeitungszeit eines Auftrags ohne Zugriffskosten auf den globalen Datenbestand und der mittlere Zeitbedarf für diese Zugriffe jeweils pro Auftragsbearbeitung.

4.2.2 Fehlerbehandlung

In parallelen und verteilten Systemen besteht verstärkter Bedarf an Fehlerbehandlungsmechanismen und Unterstützung bei der Fehlersuche. Gegenüber sequentiellen Programmen erhält man weitere Fehlerquellen durch Rechnerausfall, Prozeßabsturz, Nachrichtenprobleme und Synchronisations- bzw. Protokollfehler. Da wir im HiCon Modell den Schwerpunkt auf Lastbalancierung setzen, bieten wir lediglich zwei Konzepte zur Fehlererkennung an: *Timeouts* und Protokolldateien.

Jeder Aufruf einer *Loadman*-Bibliotheksfunktion kann mit einem *Timeout* versehen werden. Dies ist die einfachste und häufig eingesetzte Methode, um Protokollfehler und Ausfälle in der Anwendung zu erkennen. Allerdings sind die Funktionsaufrufe nicht atomar; die *Timeout*-Spezifikation wird beim Senden und Empfang von Nachrichten verwandt und die Funktionsausführung wird im Fehlerfall an dieser Stelle abgebrochen.

Alle am Ablauf teilnehmenden Prozesse schreiben in Protokolldateien. Dabei benutzt eine Serverinstanz ihre Protokolldatei, in die also auch Anwendungsinformation geschrieben werden kann, gemeinsam mit der *Loadman*-Bibliothek und der darunterliegenden *Coin*-Nachrichtenbibliothek. Nach Ablauf der Anwendung lassen sich Fehler durch Vergleiche der Protokolldateien feststellen. In Fehlerfällen fügen die Bibliotheksfunktionen grundsätzlich neben Rückgabe des Fehlercodes eine textuelle Fehlerbeschreibung in die Protokolldatei.

4.2.3 Verfügbare Strategien

Wir haben zunächst nur einfache Strategien, die jeweils auf maximal einer dynamischen Meßgröße basieren, untersucht. Derzeit sind folgende Verfahren auf *Loadman* verfügbar:

- **Round Robin**: in jeder Serverklasse werden die Aufträge reihum verteilt. Dabei sollen die Auftragswarteschlangen der Instanzen möglichst bis zu einer festen Länge gefüllt sein. Grundsätzlich wird der älteste Auftrag in der Klassenwarteschlange zuerst zugewiesen. Wenn in einer Klasse viel mehr Aufträge anfallen als die Serverinstanzen abarbeiten können, d.h. die Klassen-Warteschlange lang ist, wird der älteste Auftrag der Instanz zugeordnet, die als erste wieder Platz in ihrer Warteschlange hat. Das entspricht nicht mehr der reinen Reihum-Verteilung sondern eher der dritten Strategie, ist jedoch sehr lukrativ. Unter der Annahme gleicher Auftragsgrößen und Prozessorleistungen und Mißachtung sonstiger Einflußfaktoren (siehe Kapitel 1.1) realisiert dieses Verfahren Lastbalancierung zugunsten minimaler Auftragsausführungszeit.
- **Daten-Lokalität**: Aufträge werden stets an die Instanz abgegeben, welche die 'meisten' der benötigten Daten bereits lokal verfügbar hat. Für Lesezugriffe genügen Kopien, für Änderungsoperationen bedarf es der Originaldaten. Beim Aufruf werden Wahrscheinlichkeiten für Änderungsoperationen angegeben, nach denen das Verfahren die Daten-Beschaffungskosten gewichtet. Man beachte, daß jeder Auftrag auch dann wartet, bis seine bevorzugte Instanz genügend Raum in ihrer lokalen Warteschlange aufweist, wenn andere Instanzen arbeitslos sind (im Gegensatz zu der im Round Robin Verfahren gewählten Lösung). Die Entscheidung für eine bestimmte Instanz erfolgt aber nicht bei der Entstehung des Auftrages; vielmehr wird, sobald eine Instanz verfügbar wird, überprüft, ob sie für Aufträge aus der Klassen-Warteschlange die bestgeeignete ist. Diese späte Zuweisung ist notwendig, da sich die Datenlokationen ständig ändern. Grundsätzlich wird der älteste Auftrag, welcher diese Instanz beansprucht, zuerst abgegeben. Wie alle hier betrachteten Verfahren wird hier die Ausführungszeit der Einzelaufträge unter Mißachtung weiterer Faktoren minimiert.
- **Kürzeste Auftrage-Warteschlange**: der älteste Auftrag wird an die Instanz verwiesen, die gerade die wenigsten Aufträge in ihrer Warteschlange hat. Wie bei allen anderen Strategien werden auch hier die Warteschlangen der Instanzen nur bis zu einem festen Maximalwert gefüllt. Die Motivation hierfür sind nicht begrenzte Pufferfähigkeiten der Instanzen sondern die möglichst späte Bindung eines Auftrages an eine feste Instanz sowie die Kontrolle der entstehenden Parallelität (siehe Kapitel 3.5 und 3.5.1).

  Beim Verfahren 'kürzeste Auftrage-Warteschlange' muß man einen Kompromiß zwischen häufiger Zusendung der aktuellen Schlangenlängen und der Abschätzung durch den Lastbalancierer selbst finden. Wenn Instanzen nach oder mehrmals während der Abarbeitung eines Auftrages Zustandsnachrichten an den Balancierer schicken entsteht großer Zusatzaufwand. Wenn der Balancierer hingegen allein aus den Aufruf-Angaben und der Rechnerbelastung den Abarbeitungszustand vermutet, d.h. Fremdlast und bekannte Last durch weitere arbeitende Instanzen betrachtet, kann es sehr schnell zu Fehleinschätzungen und dadurch zu Fehlentscheidungen kommen. Im derzeitigen Prototyp bleibt die Entscheidung dem Anwender überlassen: Serverinstanzen können zu beliebigen Zeitpunkten Zustandsinformation senden. Bei Rücksendung eines Resultates wird sie stets automatisch beigefügt.

- **Leistungsfähigste Instanz**: die momentane Leistungsfähigkeit eines Prozessors wird abgeschätzt, indem man seine Maximalleistung durch die Zahl der aktiven Prozesse (Run Queue Length) - zuzüglich dem gerade zu vergebenden - teilt. Der älteste Auftrag wird an eine der Instanzen auf diesem Prozessor vergeben, wenn ihre lokale Auftragsschlange nicht zu voll ist.

Diese Sammlung an relativ primitiven Methoden soll zunächst genügen, um die Auswirkungen verschiedener Methoden durch die exekutive Lastbalancierung zu studieren. Der *Loadman* Prototyp ist derzeit noch nicht in der Lage, strategische Lastbalancierungsentscheidungen zu treffen, d.h. selbständig Anwendungscharakteristiken mit bestimmten Strategien zu verbinden. Daher muß bislang der Benutzer diese Aufgabe wahrnehmen, indem er Strategien anwählt. Für die Realisierung der Strategischen Balancierungskomponente sollte zuvor eine Sensitivitätsanalyse durchgeführt werden, um festzustellen, für welche Klassen von Lastprofilen welche Lastfaktoren relevant sind und anhand welcher Größen einzelne Balancierungsverfahren als geeignet identifizierbar sind.

4.2.4 Evaluierung eines Anwendungsbeispiels

Für eine erste Validierung des Lastbalancierungsprinzips durch den Prototyp *Loadman* wurde eine Wegesuche als Anwendung gewählt. Sie findet den kürzesten Weg zwischen zwei vorgegebenen Knoten in einem gerichteten Graphen mit gewichteten Kanten. Leider würden In Client-Server Architekturen durch synchrone rekursive Aufrufe binnen kurzer Zeit alle Serverinstanzen belegt sein, da sie auf die Ergebnisse ihrer rekursiven Aufrufe warten. Im HiCon Modell verzichten wir gewöhnlich auf Multi-Threading, obwohl auch ‘multi-threaded’ Instanzen teilnehmen können. Daher müssen die Aufrufe asynchron und ohne Resultatrückgabe abgesetzt werden. Das hierbei auftretende Terminierungsproblem wird gelöst, indem eine Serverklasse - unter anderem - die Zahl der noch zu untersuchenden Knoten bzw. die Zahl der noch zu bearbeitenden Suchaufträge verwaltet. Der *Supervisor* [Schiele91] stellt dazu im zentralen Fall das Konzept des Klassen-Events zur Verfügung.

Die Wegesuche wurde in Gestalt dreier Serverklassen realisiert:

- *gclient*: führt die Benutzerinteraktion durch, startet die Suche und verfolgt an deren Ende den kürzesten Weg zurück.
- *gfind*: bekommt im Aufruf jeweils eine Menge von Startknoten (aus derselben Partition). Sie sucht im Graph alle zugehörigen Folgeknoten samt Kosten heraus, eliminiert bei Duplikaten den teureren Weg. Sie teilt die Menge der gefundenen Zielknoten nach Partitionen auf und verschickt jeweils die Knoten einer Partition als Auftrag an die Klasse *greach*. Die Graphenbeschreibung ist in Form je einer Datei pro Partition abgelegt.
- *greach*: verwaltet die Liste der bisher erreichten Knoten. Sie bekommt im Aufruf jeweils eine Liste neu erreichter Knoten und trägt die Neuigkeiten bzw. Verbesserungen in ihrer Liste ein. Knoten, die keine Verbesserung darstellen, eliminiert sie aus der Aufrufliste. Dann schickt sie die Restliste als Aufruf an die Klasse *gfind*. Die Klasse verwaltet dabei auch die Zahl der noch zu untersuchenden Knoten. Diese Zählung ist nicht trivial, da sie nicht weiß, wieviel Folgeaufträge eine *gfind*-Instanz aus einem Aufruf von einer *greach*-Instanz generieren wird. Der Datensatz zur Terminierungszählung stellt sich als ‘Hot Spot’ unter den statischen Datensätzen der Klasse *greach* heraus, da er bei jedem Aufruf modifiziert wird.

Wenn man die Partitionierungsfunktion der Knoten des Graphen für die Serverklassen unterschiedlich wählt, also der Knotenbereich einer *gfind*-Datenpartition nicht mit dem einer *greach*-Partition übereinstimmt, so muß *greach* im allgemeinen einige Partitionen pro Aufruf anfassen und oft auch aktualisieren. Man sieht anhand der Wartezeiten zwischen Aufträgen und der Datenverwaltungskosten, daß *greach* deshalb bei feinerem Granulat - sowohl bezüglich der Auftragsgröße als auch der Größe einer Datenpartition - zum Engpaß der Anwendung wird. Bei größeren Graphen werden die *gfind*-Instanzen zum Engpaß, d.h. zum bestimmenden Faktor.

Abbildung 10 veranschaulicht noch einmal die Aufrufstrukturen in der Wegesuche-Anwendung. Die Ellipsen stellen Serverinstanzen dar, die Pfeile sollen Serverklassen-Aufrufe symbolisieren.

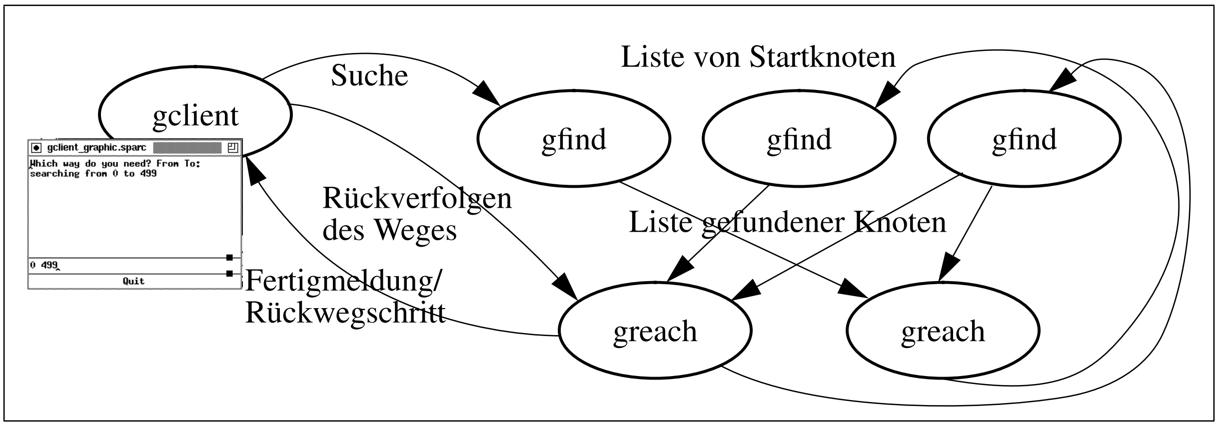

Abbildung 10 : Aufrufstruktur der Beispielanwendung.

Die Ressourcenbedürfnisse - wir beschränken uns hier auf die Rechenzeit - der Serverklassen-Aufrufe wurden statistisch ermittelt; sie hängen in erster Näherung linear von der Anzahl der Eingabeknoten und der Größe des Graphen ab. Die ‘Zahl der Instruktionen’ ist dabei architekturunabhängig und nur auf die in Benchmarks übliche Leistungskenngröße MIPS (Millionen Instruktionen pro Sekunde) bezogen. Die Plattenzugriffe und Kommunikationskosten wurden außer Acht gelassen:

gfind: #Instruktionen = 182755 + #Kanten * 26 + #Startknoten * 30000

greach: #Instruktionen = 6808 + #Kanten * 2 + #Erreichte_Knoten * 16200

Die Aufrufe an die Klassen *greach* und *gfind* beziehen sich stets nur auf eine *gfind*-Partition. Die in einem Aufruf enthaltenen Knoten sind also alle in derselben statischen Datenpartition abgespeichert. Diese Entscheidung wurde getroffen, weil *gfind*’s Partitionen als relativ große Dateien realisiert sind und neben dem Zugriff auch die Replikation und Migration sehr aufwendig machen. Da es sich jedoch ausschließlich um Lesezugriffe handelt, verschwindet der Verwaltungsaufwand im Verlaufe der Anwendung in dem Maße als die Instanzen mit Kopien versorgt sind.

Im folgenden betrachten wir Messergebnisse der Wegesuche-Applikation zuerst in einem kleinen und darauf in einem relativ großen Graphen, wobei sowohl die Anzahl der Instanzen je Serverklasse als auch die Balancierungsstrategien variiert werden. Der kleine Graph besteht aus 500 Knoten und 10000 Kanten. Letztere sind auf 10 Kontext-Partitionen der Klasse *gfind* und damit auf Dateien der mittleren Größe von 6 KB verteilt. Der Graph weist eine Lokalität von 95% auf. Lokalität bezeichnet dabei den Anteil der Kanten, die zwei Knoten derselben Partition verbinden. Abbildung 11 zeigt die maximal verwendete Server-Konfiguration; in die durch graue Rechtecke symbolisierten Rechner sind die Serverinstanzen als Ellipsen eingezeichnet. Die Rechner sind durch ein gemeinsames Netz verbunden.

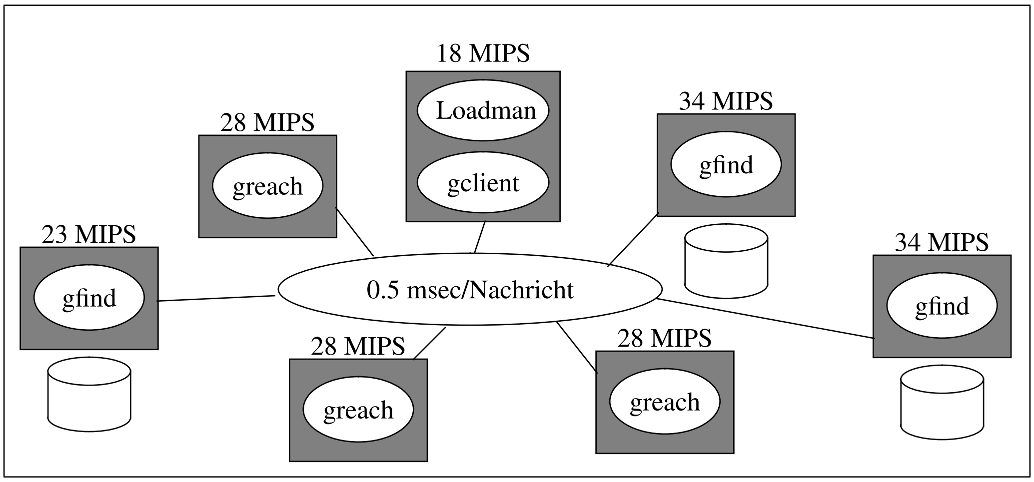

Abbildung 11 : zur Evaluierung der Beispielanwendung verwendete Systemkonfiguration.

Die erreichten Laufzeiten sind in Tabelle 1 aufgeführt. Das Instanzenverhalten schlüsselt die Zeitanteile einer Auftragsbearbeitung einer Serverinstanz auf. Sie sind jeweils über die Instanzen einer Klasse und über deren bearbeitete Aufträge gemittelt. Die Angabe *busy* gibt die mittlere Zeit, die Instanzen der Klasse pro Auftrag gerechnet haben, wobei die Wartezeit auf statische Datensätze nicht enthalten ist. Die *data* Angaben zeigen die mittlere Zeit, die Instanzen pro Auftragsbearbeitung auf Datenpartitionen gewartet haben; *idle* ist schließlich die Zeit, welche sie im Mittel nach Ende einer Auftragsbearbeitung bis zum Erhalt eines nächsten warten mußten. Die obere Zeile bezieht sich jeweils auf die Klasse *gfind*, die untere auf *greach*. Die Zahlenwerte sind jeweils gemittelte Werte aus mehreren gleichartigen Messreihen und weisen eine Streuung von etwa 2% auf.

Tabelle 2 enthält die Ergebnisse der Wegesuche in einem relativ großen Graphen, bestehend aus 100000 Kanten, die auf 30 Partitionen zu je 20 KB verteilt sind. Die übrigen Eigenschaften entsprechen denen des kleinen Graphen.

Je nach Strategie und Granulat wird die Klasse *gfind* bzw. die Klasse *greach* zum kritischen Faktor, d.h. Engpaß. Eine eher ausgewogene Belastung deutet auf relativ starke Auslastung des Lastbalancierers *Loadman* selbst hin. Man erkennt dies leicht an den durchschnittlichen Wartezeiten. Der Balancierer selbst wird - ohne Verwendung der graphische Anzeige - lediglich zu 5..20% beansprucht Für obige Messungen wurde er gemeinsam mit der Serverinstanz der Klasse *gclient* auf einem separaten Prozessor konfiguriert.

Das Abschneiden der unterschiedlichen Strategien in den verschiedenen Situationen wollen wir hier nicht detaillierter betrachten. Die Messungen sollen im wesentlichen die Einsatzfähigkeit des Lastbalancierungsprototyps belegen und ihn damit für zukünftige Realisierungen des HiCon-Konzeptes qualifizieren. Wichtig ist unter anderem die Abschätzung, inwiefern die zentrale Balancierungskomponente in der Lage ist, auch bei relativ feinem Auftragsgranulat den Durchsatz bei Erhöhung der Parallelität zu steigern ohne selbst ein Engpaß zu werden. Ein optimaler *Speedup* ist bei Operationen auf gemeinsamen Daten durch die resultierenden Reihenfolgeabhängigkeiten, Konsistenzbedingungen und Kommunikationskosten - und dies sind die wesentlichen

Grenzen der Parallelisierung - ohnehin nicht erreichbar. Deshalb verzichten wir an dieser Stelle auch auf genauere Auswertungen und auf Optimierung der Beispielanwendung bzw. der einzelnen Lastbalancierungsmethoden.

Instanzen je Klasse	statische Balancierung	dynamische Balancierung	Laufzeit [sec]	Instanzenverhalten [msec]			Engpaß
				busy	*data*	*idle*	
1	gleichgültig	gleichgültig (Round Robin)	15.2	61	0	0	gfind.0
				2	0	16	
2	Data Distribution	Data Locality	9.9	67	0	6	loadman
				* 2	3	6	
		Run Queue Length	13.5	66	8	56	gfind.0 und gfind.1
				* 2	2	23	
		Instance Queue Length	45.9	67	6	368	greach.0-1
				2	95	3	
		Round Robin	48.0	62	7	277	greach.0-1
				2	93	4	
	None	Round Robin	49.9	84	# 12	284	greach.0-1

#: Daten-Wartezeiten entstanden nur bei Instanz 1. *: nur Instanz 0 wurde genutzt

Tabelle 1 : Messergebnisse der Wegesuche bei feinem Granulat.

Instanzen je Klasse	statische Balancierung	dynamische Balancierung	Laufzeit [sec]	Instanzenverhalten [msec]			Engpaß
				busy	*data*	*idle*	
1	gleichgültig	gleichgültig (Round Robin)	127.5	156	0	0	gfind.0
				4	0	40	
2	Data Distribution	Run Queue Length	105.7	213	22	44	gfind.0
				4	61	30	
		Data Locality	109.8	212	0	41	gfind.1
				* 4	2	31	
	None	Round Robin	124.1	209	# 18	92	greach.0-1
				4	86	2	
	Data Distribution	Instance Queue Length	127.3	213	23	107	greach.0-1
				4	86	2	
		Round Robin	131.4	215	21	79	greach.0-1
				4	85	2	
3	Data Distribution	Data Locality	98.7	230	0	+ 81	gfind.1-2
				* 4	2	23	
		Run Queue Length	132.1	227	49	231	greach.0
				5	84	32	
		Instance Queue Length	155.0	266	47	316	greach.0-2
				4	156	2	
	None	Round Robin	157.1	227	38	284	greach.0-2
				4	157	3	
	Data Distribution	Round Robin	164.0	261	41	232	greach.0-2
				4	157	3	
#: Daten-Wartezeiten nur bei Instanz 1. +: Wartezeiten nur bei Instanz 0 *: nur Instanz 0 wurde genutzt							

Tabelle 2 : Messergebnisse der Wegesuche bei grobem Granulat.

Man sieht bereits an dieser prototypischen Realisierung, daß die Berücksichtigung verschiedener Lastfaktoren und Anwendungscharakteristiken und die Anwendung unterschiedlicher Strategien signifikanten Einfluß auf den Nutzen allgemeiner Lastbalancierung hat. Somit unterlegt bereits diese Auswertung das in Kapitel 1.1 theoretisch angedeutete Potential komplexerer Lastbalancierungsansätze wie das HiCon Projekt.

5 Ausblick

Zur Einschätzung des Anwendungsspektrums, für welches das vorgestellte Modell zur dynamischen Lastbalancierung geeignet ist, sind sicherlich noch weitere Applikationen zu untersuchen. Der nächste Schritt zur Realisierung des bisher entwickelten Balancierungsverfahrens besteht in der Charakterisierung der oben betrachteten Einzelmethoden, wobei die Schwierigkeit vor allem darin liegt, die Sensitivität der diversen Kenngrößen für verschiedene Anwendungsklassen zu ermitteln. Als weitere Schritte stehen die genaue Definition und Realisierung der Balancierungshierarchie (siehe Kapitel 3.5.4) sowie die Konkretisierung der Planungskomponente für Auftragsgruppen (siehe Kapitel 3.5.2) an.

Von weitergehender Optimierung unseres Prototyps wie etwaigen Verkürzungen von Nachrichtenwegen oder der Nutzung niedrigerer Software-Ebenen sehen wir derzeit ab. Stattdessen werden wir tiefergehende Untersuchungen prinzipieller Design-Alternativen genauer erwägen: die Einführung einer Balancierungsinstanz je Serverklasse verspricht Vereinfachungen im Modell, bereitet jedoch Schwierigkeiten bei auftragsübergreifender Einplanung. Ebenso ist eine konzeptionelle Dezentralisierung der Komponente, wie etwa die lokale Zuweisung kurzer, unwichtiger oder vom Profil her unbekannter Aufträge zu erwägen. Schließlich ist das derzeit gewählte Programmiermodell in seinen Vorzügen und Nachteilen mit Varianten zu vergleichen. So werden häufig, unter anderem im *Supervisor* [Schiele91], Petrinetze zur Spezifikation und Synchronisation paralleler Abläufe bevorzugt; weiterhin kann man ‘multi-threaded’ Serverinstanzen (siehe Kapitel 1.4 bzw. 3.3) in das Balancierungsmodell aufnehmen oder in Anlehnung an Datenbanksysteme globale Datensätze, die allen Serverklassen gemeinsam zugänglich sind, einführen. Schließlich läßt sich die Palette der bisher beleuchteten Lastbalancierungsfaktoren noch um Berücksichtigung der Verzögerungen durch Datensperren oder um Beachtung des verfügbaren Hauptspeicherplatzes bereichern. Dies wird sich für einige Anwendungstypen als wesentlich herausstellen.

Literaturverzeichnis

[Barak85] A. Barak, A. Shiloh, *A Distributed Load-balancing Policy for a Multicomputer*, Software-Practice and Experience, Vol. 15, No. 9, 1985.

[Becker92] W. Becker, *Lastbalancierung in heterogenen Client-Server Architekturen*, Fakultätsbericht 1992 /1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992.

[Berger87] M. Berger, S. Bokhari, *A Partitioning Strategy for Nonuniform Problems on Multiprocessors*, IEEE Transactions on Computers, Vol. 36, No. 5, 1987.

[Blazewicz86] J. Blazewicz, M. Drabowski, J. Weglarz, *Scheduling Multiprozessor Tasks to Minimize Schedule Length*, IEEE Transactions on Computers, Vol. 35, No. 5, 1986.

[Bokhari81] S. Bokhari, *A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System*, IEEE Transactions on Software Engineering, Vol. 7, No. 6, 1981.

[Bowen88] N. Bowen, C. Nikolaou, A. Ghafoor, *Hierarchical Workload Allocation for Distributed Systems*, Proceedings Parallel Processing, 1988.

[Bruno85] J. Bruno, *On Scheduling Tasks with Exponential Service Times and In-Tree Precedence Constraints*, Acta Inform. 22, 1985.

[Bryant81] R. Bryant, R. Finkel, *A Stable Distributed Scheduling Algorithm*, IEEE Proceedings Distributed Computing Systems, 1981.

[Casavant88] T. Casavant, J. Kuhl, *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Engineering, Vol. 14, No. 2, 1988.

[Chandy75] K. Chandy, P. Reynolds, *Scheduling Partially Ordered Tasks with Probabilistic Execution Times*, Proceedings Operating System Principles, Operating Systems Review, Vol. 9, No. 5, November 1975.

[Ciciani88] B. Ciciani, D. Dias, P. Yu, *Load Sharing in Hybrid Distributed - Centralized Database Systems*, Proceedings Distributed Computing Systems, 1988.

[Copeland88] G. Copeland, W. Alexander, E. Boughter, T. Keller, *Data Placement in Bubba*, Proceedings SIGMOD, 1988.

[Cybenko89] G. Cybenko, *Dynamic Load Balancing for Distributed Memory Multiprocessors*, Journal of Parallel and Distributed Computing, No. 7, 1989.

[Douglis91] F. Douglis, J. Ousterhout, *Transparent Process Migration: Design Alternatives and the Sprite Implementation*, Software-Practice and Experience, Vol. 21, No. 8, 1991.

[Eager85] D. Eager, E. Lazowska, J. Zahorjan, *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Austin, Texas, 1985.

[Eager86] D. Eager, E. Lazowska, J. Zahorjan, *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 12, No. 5, 1986.

[Ezzat86] A. Ezzat, *Load Balancing in NEST: A Network of Workstations*, Proceedings Fall Joint Computer Conference, Dallas, Texas, 1986.

[Ferrari86] D. Ferrari, S. Zhou, *A Load Index for Dynamic Load Balancing*, Proceedings Fall Joint Computer Conference, Dallas, Texas, 1986.

[Gavish90] B. Gavish, O. Sheng, *Dynamic File Migration in Distributed Computer Systems*, Communications of the ACM, Vol. 33, No. 2, 1990.

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

[Hosseini90] S. Hosseini, B. Litow, M. Malkawi, J. Mc Pherson, K. Vairvan, *Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm*, Journal of Parallel and Distributed Computing, No. 6, 1990.

[Hsu86] C. Hsu, J. Liu, *Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems*, Proceedings Distributed Computing Systems, 1986.

[Iqbal86] M. Iqbal, J. Saltz, S. Bokhari, *A Comparative Analysis of Static and Dynamic Load Balancing Strategies*, Proceedings Parallel Processing, 1986.

[Kale88] L. Kale, *Comparing the Performance of two Dynamic Load Distribution Methods*, Proceedings Parallel Processing, 1988.

[Kanet91] J. Kanet, V. Sridharan, *PROGENITOR: A generic algorithm for production scheduling*, Wirtschaftsinformatik, Heft 4, August 1991.

[Kuchen90] H. Kuchen, A. Wagener, *Comparison of Load Balancing Strategies*, Bericht 5/90, Lehrstuhl für Informatik II, RWTH Aachen, 1990.

[Lam75] S. Lam, R. Sethi, *Analysis of a Level Algorithm for Preemptive Scheduling*, Proceedings Operating Systems Principles, Operating Systems Review, Vol. 9, No. 5, November 1975.

[Li90] K. Li, K. Cheng, *Static Job Scheduling in Partitionable Mesh Connected Systems*, Journal of Parallel and Distributed Computing, No. 10, 1990.

[Lin87] F. Lin, R. Keller, *The Gradient Model Load Balancing Method*, IEEE Transactions on Software Engineering, Vol. 13, No. 1, 1987.

[Lin92] H. Lin, C. Raghavendra, *A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC)*, IEEE Transactions on Software Engineering, Vol. 18, No. 2, 1992.

[Lo88] V. Lo, *Algorithms for Static Task Assignment and Symmetric Contraction in Distributed Computing Systems*, Proceedings Parallel Processing, 1988.

[Lo88/2] V. Lo, *Heuristic Algorithms for Task Assignment in Distributed Computing Systems*, Proceedings Parallel Processing, 1988.

[Ma82] P. Ma, E. Lee, M. Tsuchiya, *A Task Allocation Model for Distributed Computing Systems*, IEEE Transactions on Computers, Vol. 31, No. 1, 1982.

[Schiele91] G. Schiele, *Kontrollstrukturen zur Ablaufsteuerung massiv paralleler Datenbankanwendungen in Multiprozessorsystemen*, Dissertation, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1991.

[Smith80] R. Smith, *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, IEEE Transactions on Computers, Vol. 29, No. 12, 1980.

[Staib92] R. Staib, *Simulator zur Lastbalancierung in Client / Server Architekturen*, Diplomarbeit, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992.

[Thomasian86] A. Thomasian, P. Bay, *Analytic Queueing Network Models for Parallel Processing of Task Systems*, IEEE Transactions on Computers, Vol. 35, No. 12, 1986.

[Towsley86] D. Towsley, *Allocating Programs Containing Branches and Loops within a Multiple Processor System*, IEEE Transactions on Software Engineering, Vol. 12, No. 10, 1986.

[Trippner92] R. Trippner, *Lastbalancierung einer exemplarischen Anwendung auf vernetzten Workstations*, Diplomarbeit, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992.

[Varadarajan88] R. Varadarajan, E. Ma, *An Approximate Load Balancing Model with Resource Migration in Distributed Systems*, Proceedings Parallel Processing, 1988.

[Weinmann92] T. Weinmann, *Datenparallelität durch Partitionierung und Replikation*, Studienarbeit, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992.

[Yu86] P. Yu, S. Balsamo, Y. Lee, *Dynamic Load Sharing in Distributed Database Systems*, Proceedings Fall Joint Computer Conference, 1986.

[Yu91] P. Yu, A. Leff, Y. Lee, *On Robust Transaction Routing and Load Sharing*, ACM Transactions on Database Systems, Vol. 16, No. 3, 1991.

[Zhou87] S. Zhou, D. Ferrari, *An Experimental Study of Load Balancing Performance*, Report No. 86.8, Computer Science Division, University of California, Berkeley, 1987.