Lastmessung und -verteilung
in heterogenen, verteilten Datenbanksystemen

Diplomarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von

Linke, Ansgar

geboren am 16.06.1967 in Rostock

Betreuer: Prof. Dr. P. Forbrig

Dr. H. Meyer, Dipl. Inf. U. Langer

Rostock, den 20.05.1994

Danksagung

Herrn Prof. Dr. P. Forbrig, Dr. H. Meyer, Dipl. Inf. U. Langer danke ich für die aktiven Diskussionen und die sehr gute Unterstützung und Förderung bei der Durchführung dieser Diplomarbeit.

Rostock, 20.05.1994 Ansgar Linke

Zusammenfassung

Eine ungleiche Belastung der Rechner des verteilten Systems verursacht Leistungseinbußen und eine uneffektive Ressourcenausnutzung. Durch den Ausgleich dieser Lastunterschiede kann die Leistung des Gesamtsystems verbessert werden. Um eine effiziente Lastbalancierung zu realisieren, müssen Informationen über die bestehende Last im System verfügbar sein. Lastkomponenten auf den einzelnen Rechnern bestimmen daher periodisch die Belastung des Rechnerknotens und des Netzwerks. Die aktuell gemessenen Lastinformationen eines Rechners werden in einem Lastdeskriptor zusammengefaßt. Um die Belastung anderer Rechnerknoten in die Lastbalancierungsentscheidung des lokalen Knotens einzubeziehen, erfolgt ein Lastinformationsaustausch. Jeder Rechner verwaltet die Lastinformationen in einem Lastvektor. Der Zugriff auf die Informationen des Lastvektors wird durch Anfragen realisiert. Damit der erzielbare Nutzen aus der Lastbalancierung gegenüber dem dafür erforderlichen Aufwand dominiert, werden Einschränkungen bzgl. der Häufigkeit von Lastmessung und Lastinformationsaustausch getroffen.

Abstract

In distributed systems performance is often poorer than possible and resources are wasted because of unbalanced load of nodes. Load balancing enables performance improvement by equalizing the load over available hosts. It is nessesary to measure the current load in oder to make efficient load balancing decisions. Therefore load components determine periodic the current load situation on each node and on the network connecting different nodes. The load information is represented as a load descriptor. A frequently information exchange takes place to included load information from other nodes of systems in load balancing decisions. Each node held the load information in a load vector. Access to information of load vector is realized by request. The frequencies of load measurements and load information exchanges are reduced to avoid cost of load balancing outweighting its benefits.

CR-Klassifikation

C.2.4	Distributed systems (H.2.4)	verteilte Systeme
	Distributed databases	verteilte Datenbanken
C.4.	Performance of systems	Leistung von Systemen
	Measurement techniques	Meßtechniken
D.2.8	Performance measures	Leistungsmessungen
H.2.5	Heterogeneous database	Heterogene Datenbanken

Key Words: distributed database system, distributed query processing, load balancing, load sharing, load measurement

Abkürzungen

ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DBS	Datenbanksystem
DBVS	Datenbankverwaltungssystem
GQEP	Global Query Evaluation Plan
GQM	Global Query Manager
GRA	Global Resource Analyzer
HEAD	Heterogeneous Extensible and Distributed Database Management System
IDP	Internet Datagram Protocol (XNS)
IP	Internet Protocol (ARPANET)
IPC, ipc	Interprocess Communication
LAN	Local Area Network
LQM	Local Query Manager
LRA	Local Resource Analyzer
OSI	Open Systems Interconnection
PEX	Packet Exchange Protocol (XNS)
QEP	Query Evaluation Plan
RDP	Reliable Data Protocol (ARPANET)
SPP	Sequenced Packet Protocol (XNS)
SVR4	UNIX System V Release 4
TCP	Transmission Control Protocol (ARPANET)
TP1-TP4	OSI Transportprotokolle
UDP	User Datagram Protocol (ARPANET)
XDR	External Data Representation
XNS	Xerox Network System

1. Einleitung

Um den gestiegenen Anforderungen der Nutzer nach hoher Verfügbarkeit, hohem Durchsatz bzw. hohen Verarbeitungsgeschwindigkeiten und nach modularem Wachstum der Systeme gerecht zu werden, kommen verstärkt verteilte Systeme zum Einsatz. Gründe für diese Entwicklung sind einerseits die Existenz leistungsfähiger und zugleich preisgünstiger Rechnersysteme als Hardwarebasis für verteilte Systeme. Außerdem finden Kommunikationssysteme wie z.B. LAN eine immer größere Verbreitung. Auf der anderen Seite konnte den gestiegenen Leistungsanforderungen nur durch die Ausnutzung von Parallelitäten sowie durch die Verteilung der zu bearbeitenden Aufträge auf durch ein Netzwerk miteinander verbundene Rechner entsprochen werden.

Mit der Aufteilung der zu bearbeitenden Aufträge auf mehrere Rechnerknoten geht häufig eine Verteilung der Daten auf diese Rechner einher, um auch den obigen Anforderungen der Nutzer bei der Verarbeitung großer Datenmengen bzw. Datenbanken gerecht zu werden. Die Dezentralisierung der Datenverarbeitung in einem verteilten System bzw. die Nutzung von verteilten Datenbanken erfordert die Anwendung von verteilten Datenbanksystemen (DBS). Ein verteiltes DBS unterstützt die Heterogenität und die Autonomie der auf den verschiedenen Rechnern lokal gehaltenen Daten. Die Verwaltung, der Zugriff auf die verteilten Daten und deren Integration wird durch ein verteiltes Datenbankverwaltungssystem (DBVS) realisiert.

Ein kostenaufwendiger Weg, um die Leistung des Systems entsprechend den Nutzeranforderungen zu steigern, wäre die Erhöhung des Leistungsvermögens der Rechnerknoten oder das Hinzufügen mehrerer Rechner zum System. Diese Herangehensweise verursacht weiterhin eine Vergeudung der Leistungsfähigkeit vorhandener Ressourcen, denn ursächlich hemmend auf die Leistung wirkt sich die ungleiche Lastverteilung über die Rechner des verteilten Systems und somit die uneffektive Ausnutzung bereits bestehender Ressourcenkapazitäten aus. Erfolgt keine Lastbalancierung in einem verteilten DBS, so ist die größte Rechenleistung bei der Anfragebearbeitung von den Rechnerknoten zu erbringen, auf denen die meisten Daten gespeichert sind. Gleichzeitig sind mit hoher Wahrscheinlichkeit andere Rechner nur gering belastet bzw. untätig [Livn82]. Daher liegt in der Vermeidung dieser Lastunausgeglichenheiten der Ansatzpunkt für eine akzeptable und kosteneffektive Leistungssteigerung.

Indem die Last entsprechend der Belastung der Rechnerknoten auf diese verteilt bzw. balanciert wird, kann die Leistung des Systems verbessert werden. Die Lastbalancierung verhindert, daß ein gering oder unbelasteter Rechner vorhanden ist, während auf anderen Knoten Aufträge noch auf die Ausführung warten. Durch die Lastbalancierung in einem heterogenen, verteilten System werden leistungsfähige Rechnerknoten und deren Ressourcen allen Anwendungen und Rechnern zugänglich [Zhou93]. Es ist daher nicht erforderlich, daß auf jedem Rechner eine ausgewogene Menge der zur Auftragsausführung notwendigen Ressourcen verfügbar ist.

Mit der Verteilung der Last auf die vorhandenen Rechnerknoten wird die Verfügbarkeit, die Zuverlässigkeit und die Leistung des gesamten Systems erhöht [Ni85]. Als Ergebnis der Lastbalancierung wird neben der effektiven Verringerung der Antwortzeit von Nutzeranforderungen zusätzlich ein Glätten der zeitlichen Lastschwankungen auf den Rechnern erreicht. Damit wirkt sich die Lastbalancierung positiv auf die Stabilität des Systems aus. Als stabil wird ein System bezeichnet, wenn trotz einer mittleren Belastung der Systemressourcen die Antwortzeit begrenzt und akzeptabel bleibt.

Die Ermittlung der aktuellen Lastsituation ist eine notwendige Voraussetzung zur Realisierung der Lastbalancierung. Auf der Grundlage der dabei gewonnenen Lastinformationen sowie des vorhandenen Wissens über statische Eigenschaften von Systemressourcen und Aufträgen können dann die Verteilungsentscheidungen getroffen werden.

Die mit der Lastbalancierung verbundenen Aufgaben, wie Lastbestimmung, Lastinformationsmanagement sowie Festlegung der Transfer- und Plazierungsstrategien für die Prozeßverlagerung, werden durch die auf jedem Rechnerknoten vorhandene Lastkomponente realisiert.

Das HEAD - Projekt

Im Rahmen des HEAD - Projektes (Heterogeneous Extensible and Distributed Database Management System) [Wiso90, Flac93a] wird der Prototyp eines heterogenen, erweiterbaren und verteilten Datenbankverwaltungssystems entwickelt.

HEAD wird als *heterogenes* DBVS bezeichnet, da verschiedene Hardware (Sun SPARC 10, Sun SPARC 2, Sun IPX, IBM-kompatible PC), unterschiedliche UNIX-kompatible

Betriebssysteme (Sun-OS, BSD-UNIX, SVR4) und verschiedene lokale relationale DBVS (Ingres, Oracle, Postgres, Sybase) in das System integriert werden können. Die lokalen DBVS bearbeiten nur Anfragen, die die lokal gehaltenen Relationen betreffen. Für die komplexere und rechnerübergreifende Datenverarbeitung wird das verteilte DBVS HEAD genutzt.

Das HEAD - Projekt baut auf einer Shared nothing-Architektur auf [Ston86] (siehe Abb. 1). Die autonom arbeitenden Rechner sind lose miteinander gekoppelt und kommunizieren mittels Message Passing über das Netzwerk (Ethernet-LAN). Ein Vorteil der Shared nothing-Architektur ist die Möglichkeit der *Erweiterbarkeit* des Systems durch Hinzunahme neuer Rechnerknoten. Somit ist eine wachsende Parallelität der Datenbankanwendungen erreichbar.

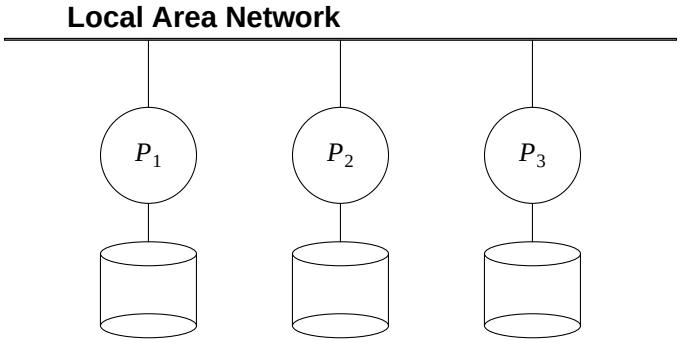

Abb. 1: Die Shared nothing-Architektur

HEAD ist ein *verteiltes* DBVS, das den Zugriff und die Verwaltung der auf mehrere Rechner verteilten Datenbank realisiert [Ceri85, Ozsu91]. Die Rechnerknoten sind miteinander über ein Kommunikationssystem verbunden, wodurch der Zugriff auf Daten, die sich auf anderen Knoten befinden, gewährleistet wird. Für Anwendungen und Benutzer ist die Verteilung der Daten transparent, d.h. dem Nutzer erscheint die verteilte Datenbank als eine einzige logische Datenbank, die nur durch ein einziges konzeptuelles Datenbankschema beschrieben wird. Die Anfragen können unabhängig von der Lokation der Daten gestellt werden. Ein verteiltes DBS besitzt im Vergleich zu zentralisierten DBS mehrere Vorteile:

- hohe Zuverlässigkeit und Verfügbarkeit
- Anwendungsnähe und lokale Autonomie
- gute Anpaßbarkeit der EDV-Struktur an die Organisationsstruktur von Unternehmen
- Flexibilität und Erweiterbarkeit

Ziel der Untersuchungen und Analysen im Rahmen des HEAD - Projektes sind Realisierung und Optimierung der parallelen Anfragebearbeitung in einem heterogen, verteilten DBS. Um eine Leistungssteigerung zu erreichen, die in einer Verkürzung der Antwortzeit von Anfragen resultiert, werden neben der Parallelisierung der Anfragebearbeitung (Inter- und Intra-Operatorparallelität) Techniken wie Pipelining [Mikk88] und Lastbalancierung [Scha92, Gosc91] angewendet. In die Bearbeitung von Anfragen werden die Rechner entsprechend ihrer Leistungsfähigkeit einbezogen. Dadurch wird das unterschiedliche Leistungsvermögen der Rechnerknoten (heterogene Umgebung) berücksichtigt.

Das HEAD - System wird in funktionsorientierte Einheiten unterteilt [Flac93a]. Die Funktionseinheiten, die den einzelnen Rechnerknoten entsprechend ihrer Leistungsfähigkeit zugeordnet werden, können in den folgenden zwei eigenständigen Systemkomponenten zusammengefaßt werden:

- Funktionen zur Anfrageübersetzung und -optimierung (können auf allen Rechnerknoten verwirklicht werden)
- Funktionen zur Abarbeitung der Anfrageausführungspläne

Eine weitere Unterteilung der Funktionen, die die Abarbeitung der Ausführungspläne umfassen, erfolgt in:

- Funktionen zur Abbildung physischer Datenstrukturen auf Mengen von Tupeln (bilden Schnittstelle zur jeweiligen lokalen Datenbank)
- Funktionen zur mengenorientierten, datenflußgesteuerten Verarbeitung von Tupeln (werden auf den Rechnern, die erforderliches Leistungsvermögen bieten, realisiert)

Diese Gliederung in eigenständige Funktionseinheiten ermöglicht es, daß Teile der Anfragebearbeitung auf einzelne Rechnerknoten verteilt werden können. Dabei ist die Ausführung der Anfrage nicht auf die Rechner beschränkt, welche die zur Bearbeitung erforderlichen Daten gespeichert haben. Auf Rechnerknoten, auf denen eine lokale Datenbank gespeichert ist, muß eine lokale Datenbankschnittstelle bereitgestellt werden. Durch die Verteilung der Anfragebearbeitung können stark belastete Rechner entlastet werden. Indem außerdem das Leistungsvermögen der Rechner berücksichtigt wird, kann gleichzeitig die Rechenkapazität leistungsfähiger Workstations ausgenutzt werden.

Die Unterteilung in funktionsorientierte Einheiten bestimmt die funktionelle Architektur von HEAD (siehe Abb. 2). Der SQL-Compiler und der Global Query Manager umfassen die Funktionen im Zusammenhang mit der Anfrageübersetzung und der -optimierung. Die Funktionen zur Abarbeitung der Ausführungspläne sind in den Komponenten Local Query Manager, Execution Monitor und Data driven Relational Algebra Machine (datenflußgesteuerte Relationenalgebra-Maschine) zusammengefaßt. Durch den Local bzw. Global Resource Analyzer werden die Lastmessungen und die Verwaltung der Lastinformationen (Aufgaben der Lastkomponente) realisiert, welche die Grundlage für eine lastabhängige Verteilung der Anfragebearbeitung bilden. Das Global Data Dictionary ist für die Metadatenverwaltung auf globaler und das Local Data Dictionary auf lokaler Ebene verantwortlich.

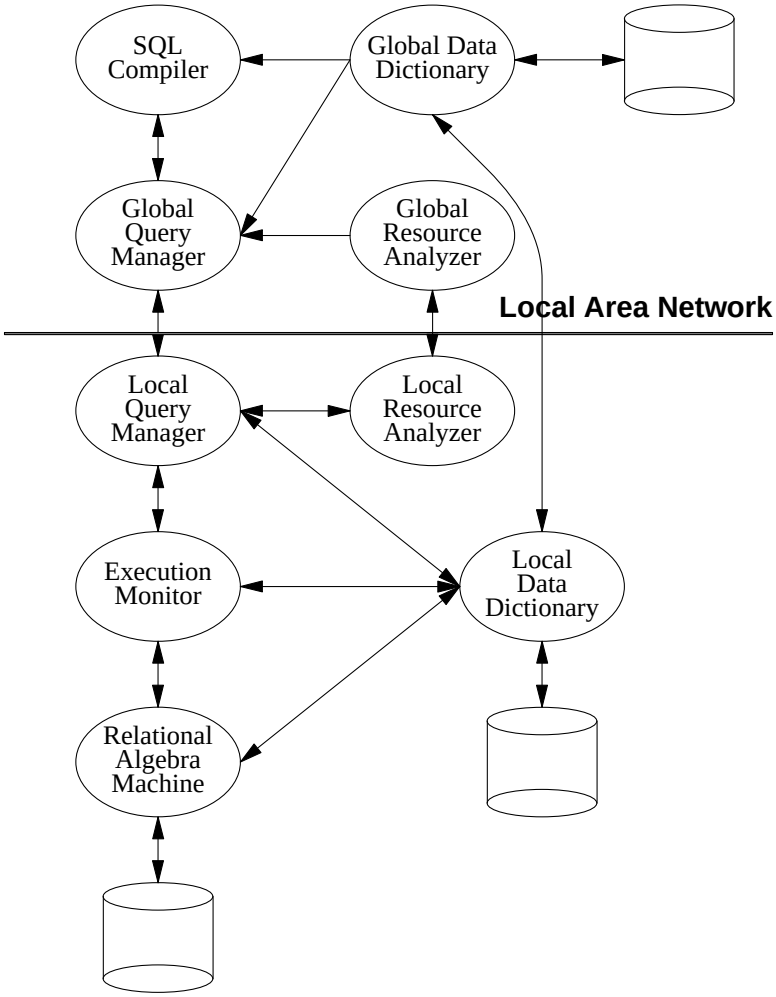

Abb. 2: Funktionelle Architektur von HE$_A$D

2. Lastbalancierung

2.1. Begriffsbestimmung

In verteilten Systemen können gleichzeitig einige Rechnerknoten schwerbelastet sein, während andere Knoten nur gering oder gar nicht belastet sind. Diese Lastunausgeglichenheiten führen dazu, daß Rechnerressourcen nur unzureichend genutzt bzw. ausgelastet sind, was letztendlich in einer geringen Leistung resultiert. Um die auf gering belasteten Rechnerknoten ungenutzten Ressourcenkapazitäten für die Bearbeitung von Nutzeranforderungen auf stark belasteten Rechnern zu nutzen, kann eine Lastverteilung (load sharing) bzw. Lastbalancierung (load balancing) erfolgen. Beide Strategien realisieren die Verteilung der Last auf den verfügbaren Rechnern des verteilten Systems. Durch die Verlagerung eines Prozesses von einem schwer zu einem gering belasteten Rechner und die sich daran anschließende entfernte Ausführung des Prozesses wird auf eine ungleiche Verteilung der Last über das System reagiert. Beide Herangehensweisen streben dabei eine Verbesserung der Leistung des verteilten Systems an. Sowohl bei der Lastverteilung als auch bei der Lastbalancierung werden die Verteilungsentscheidungen auf der Grundlage von Kenntnissen über die Lastzustände der einzelnen Rechnerknoten des Systems getroffen. Beide Strategien unterscheiden sich aber in der Intensität der Forderung, die mit dem Zielkriterium an die Verteilung der Last gestellt wird [Gosc91]:

- Ziel der **Lastverteilung** ist zu gewährleisten, daß kein Rechnerknoten untätig ist, während andere Rechner überlastet sind und Prozesse auf eine Bearbeitung warten.
- Die **Lastbalancierung** strebt nach einem Ausgleich der vorhandenen Last auf allen Rechnern im verteilten System.

Bei der **Lastverteilung** wird ein schwächeres Kriterium im Vergleich zur Lastbalancierung angesetzt. Da unbelastete Rechnerknoten vermieden werden, solange noch Aufträge im System auf ihre Ausführung warten, werden die vorhandenen Ressourcenkapazitäten der unbelasteten Rechner zur Leistungssteigerung genutzt. Die Lastverteilung verhindert extreme Lastunterschiede (unbelastet < – > belastet) zwischen den beteiligten Rechnern [Scha92]. Durch das Zielkriterium der Lastverteilung wird nicht zwingend eine globale Leistungsverbesserung für das Gesamtsystem angestrebt

[Gosc91].

Zur Verdeutlichung soll folgendes Beispiel dienen. Auf drei vorhandenen Rechnern ist eine Anzahl s von Aufgaben zu bearbeiten. Diese Menge von Aufgaben ist auf die drei Rechner entsprechend dem Zielkriterium zu verteilen ((k, l, m) = Aufteilung der Gesamtanzahl von Aufgaben auf die drei Rechner, k = Anzahl von Aufgaben auf einem Rechner, $s = k + l + m$). Eine Lastverteilung wäre bereits durch die Aufteilung ((k, l, m); $0 < k << m$, $0 < l << m$) der Aufgaben auf die verfügbaren Rechner erzielt, wobei kein Rechner untätig ist [Heis92]. Eine derartige Lastverteilung führt zu einer schnellen Bearbeitung von einigen der insgesamt abzuarbeitenden Aufträgen und aber gleichzeitig zu der starken Belastung eines Rechners.

Die **Lastbalancierung** versucht hingegen die Last unter allen Rechnern des verteilten Systems auszugleichen bzw. gleichmäßig zu verteilen, um somit die Leistung global für das Gesamtsystem zu verbessern. Damit wird bei der Lastbalancierung ein stärkeres bzw. weitergehendes Kriterium bei der Verteilung der Last angewendet als im Vergleich zur Lastverteilung. Jede Lastbalancierung erfüllt gleichzeitig das Zielkriterium einer Lastverteilung. Häufig wird die Last der Rechner im System bzgl. eines bestimmten Lastparameters ausgeglichen z.B. CPU-Warteschlangenlängen [Scha92]. Mit der Lastbalancierung wird versucht, die Last der einzelnen Rechner im engen Bereich um einen augenblicklichen Lastdurchschnitt des Systems zu halten [Ferr85]. Lastbalancierung und Lastausgleich werden nachfolgend synonym verwendet. Auf Grund des stärkeren Zielkriteriums bei der Lastbalancierung, ist ein Lastausgleich beim obigen Beispiel durch eine Verteilung der Last auf die Rechner von ((k, l, m); $k = l = m$) erreicht.

Beide Strategien unterscheiden sich in den erforderlichen Operationen (Lastmessung, Lastinformationsmanagement, Prozeßtransfer und Zielrechnersuche), um eine Verteilung der Last durchzuführen, trotz unterschiedlicher Zielkriterien nicht grundlegend.

2.2. Leistungsziele der Lastbalancierung

Prozesse und Aufträge bzw. Datenbank-Anfragen speziell bei Datenbank-Anwendungen, die auf den Rechnern bearbeitet und ausgeführt werden sollen, erzeugen die Last in einem Rechnersystem. Die Last beeinflußt die Leistung des System. Indem die

Last auf die verfügbaren Rechner entsprechend deren Belastungszustände verteilt wird, kann eine Leistungssteigerung des Gesamtsystems erzielt werden. Aus welchen Kriterien können konkrete Aussagen zur Leistung von Systemen abgeleitet werden?

2.2.1. Leistungskriterien

Antwortzeit

Für den Nutzer spiegelt sich die Leistung des Systems deutlich in der Zeit wider, die er warten muß, bis sein Job bearbeitet ist und das Ergebnis vorliegt (= Antwortzeit) [McGr75]. Es gibt verschiedene Möglichkeiten für die Definition der Antwortzeit. Bei [Baum89] wird als Antwortzeit die Zeit bezeichnet, die von der Jobvorlage bis zur Jobbeendigung vergeht. Eine allgemein gefaßte Definition nach [Jain91] lautet:

- Die Antwortzeit ist das Intervall zwischen dem Stellen einer Nutzeranfrage an das System und der zugehörigen Systemantwort (Def. 1).

Das Schreiben der Anfrage durch den Nutzer als auch die Antwortausgabe des Systems erfordert Zeit. Wird die benötigte Zeit bei der Antwortzeit berücksichtigt, lassen sich zwei detailliertere Antwortzeitdefinitionen formulieren [Jain91].

- Die Antwortzeit umfaßt das Intervall vom Ende der Anfrageerstellung bis zum Beginn der entsprechenden Systemantwort (Def. 2).
- Die Antwortzeit ist das Intervall zwischen dem Ende der Anfrageerstellung und dem Ende der dazugehörigen Systemantwort (Def. 3).

Alle drei Definitionen, die entsprechend den Nutzeranforderungen verwendet werden können, sind zusammenfassend in Abbildung 3 dargestellt. In der dritten Definition werden auch längere Zeiträume zwischen Beginn und Ende der Antwortausgabe des Systems berücksichtigt, die ebenfalls für die Leistungsbewertung von Bedeutung sein können. Auf Grund der möglichen entfernten Bearbeitung von Anfragen bzw. Prozessen in einem verteilten System setzt sich die Antwortzeit in diesem Fall aus der erforderlichen Verarbeitungszeit auf dem Verarbeitungsknoten und aus der Kommunikationsverzögerung im Rahmen des Job- bzw. des Ergebnistransfers zusammen [Tant85].

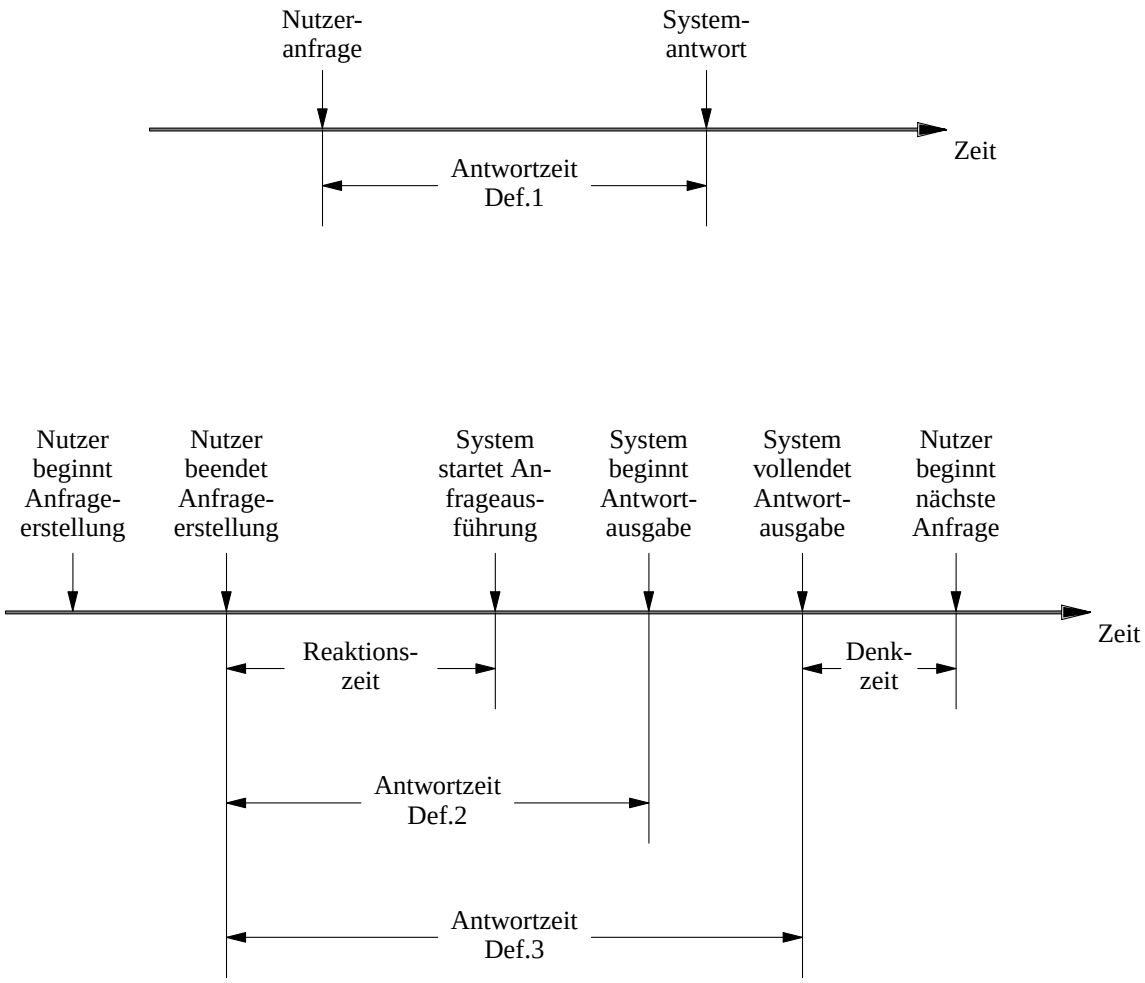

Abb. 3: Antwortzeitdefinitionen

Die Zeit zwischen der vom Nutzer beendeten Anfrageerstellung und dem Beginn der Anfrageausführung durch das System wird als Reaktionszeit des Systems bezeichnet. Ein Beispiel für diesen Zeitraum läßt sich bei der CPU-Zuteilung zu Prozessen im Timesharing-Verfahren angeben. Dabei wird als Reaktionszeit die Zeit bezeichnet, die zwischen dem letzten Tastendruck des Nutzers bei der Anfrageformulierung und der Zuteilung des ersten CPU-Quantums an den Nutzerprozeß vergeht.

Als Denkzeit des Nutzers wird die Zeit zwischen dem Ende der Antwortausgabe des Systems und dem Beginn der nächsten Anfrageerstellung durch den Nutzer bezeichnet.

Die Antwortzeit von Nutzeranforderungen wächst generell bei ansteigender Last im System. Um so mehr Anfragen vom System zu bearbeiten sind, um so länger dauert die Ausführung einer einzelnen Anfrage. Es besteht kein linearer Zusammenhang zwischen Last und Antwortzeit, da bei steigender Last eine überproportionale Vergrößerung der Antwortzeit anzunehmen ist (siehe Abb. 4).

Durchsatz

Ein weiteres wichtiges Kriterium zur Kennzeichnung der Leistung von Systemen ist der Durchsatz.

Definition Durchsatz [Lang92]:
Der Durchsatz X ergibt sich aus dem Verhältnis der Anzahl C fertig bearbeiteter Aufträge bzw. Prozesse und der Länge T des Beobachtungsintervalls.

$$X = \frac{C}{T}$$

Der Durchsatz kann für verschiedene Ressourcen bzw. Systeme wie folgt angegeben werden.

CPU:	Millionen bearbeitete Befehle pro Sekunde (millions of instructions per second = MIPS) oder Millionen bearbeitete Fließkommaoperationen pro Sekunde (millions of floating-point-operations per second = MFLOPS)
Netzwerk:	übertragene Pakete pro Sekunde (packets per second = pps) oder übertragene Bits pro Sekunde (bits per second = bps)
Transaktionssystem:	bearbeitete Transaktionen pro Sekunde (transactions per second = tps)
interaktives System:	bearbeitete Anfragen pro Sekunde

Der Zusammenhang zwischen Durchsatz und Last sowie zwischen Antwortzeit und ansteigender Last ist in Abbildung 4 dargestellt. Sowohl in der Antwortzeit - Last- als auch in der Durchsatz - Last-Kurve gibt es zwei Krümmungspunkte bzw. Kniepunkte. Bis zum ersten Kniepunkt sind die Rechner- und Systemressourcen gering belastet, so

daß anstehende Prozesse schnell bearbeitet werden können. Ausdruck dafür sind das minimale bzw. moderate Anwachsen der Antwortzeit und das fast proportionale Ansteigen des Durchsatzes im Vergleich zur wachsenden Last. Der Durchsatz am ersten Kniepunkt wird als Kniekapazität bezeichnet [Jain91]. Überschreitet die Last den ersten Krümmungspunkt und nimmt weiter zu, führt dies zu einer stärkeren Belastung des Systems. Die Rechner sind durch viele zu bearbeitende Prozesse und Nutzeranforderungen schwer belastet, so daß sich die Zeit zur vollständigen Prozeßausführung verlängert. Die Antwortzeit wächst daher wesentlich schneller mit ansteigender Last an als vor dem Überschreiten des ersten Kniepunktes. Der Durchsatz erhöht sich nur gering. Nimmt die Last weiter zu (Überschreiten des zweiten Krümmungspunktes), erfolgt eine Überlastung des Rechners. Die Antwortzeit steigt rapide an und der Durchsatz sinkt. Der Durchsatz, der bei einer noch akzeptablen Antwortzeit maximal erreicht werden kann, wird als nutzbare Kapazität des Systems bezeichnet.

Der unter idealen Verhältnissen erreichbare Durchsatz wird als nominale Kapazität des Systems bezeichnet. Die nominale Kapazität eines Netzwerks wird Bandbreite genannt.

Definition Effizienz [Jain91]:
Die Effizienz (Nutzeffekt, Wirkungsgrad) ergibt sich aus dem Verhältnis des unter realen Bedingungen erzielbaren Durchsatzes (nutzbare Kapazität) und der nominalen Kapazität.

$$Effizienz = \frac{nutzbare\ Kapazität}{nominale\ Kapazität} \qquad Effizienz = \frac{realer\ Durchsatz}{idealerDurchsatz}$$

Ressourcenauslastung

Die Ressourcenauslastung wird als Prozentsatz der Zeit, den eine Systemkomponente produktiv arbeitet, bezeichnet [Lang92].

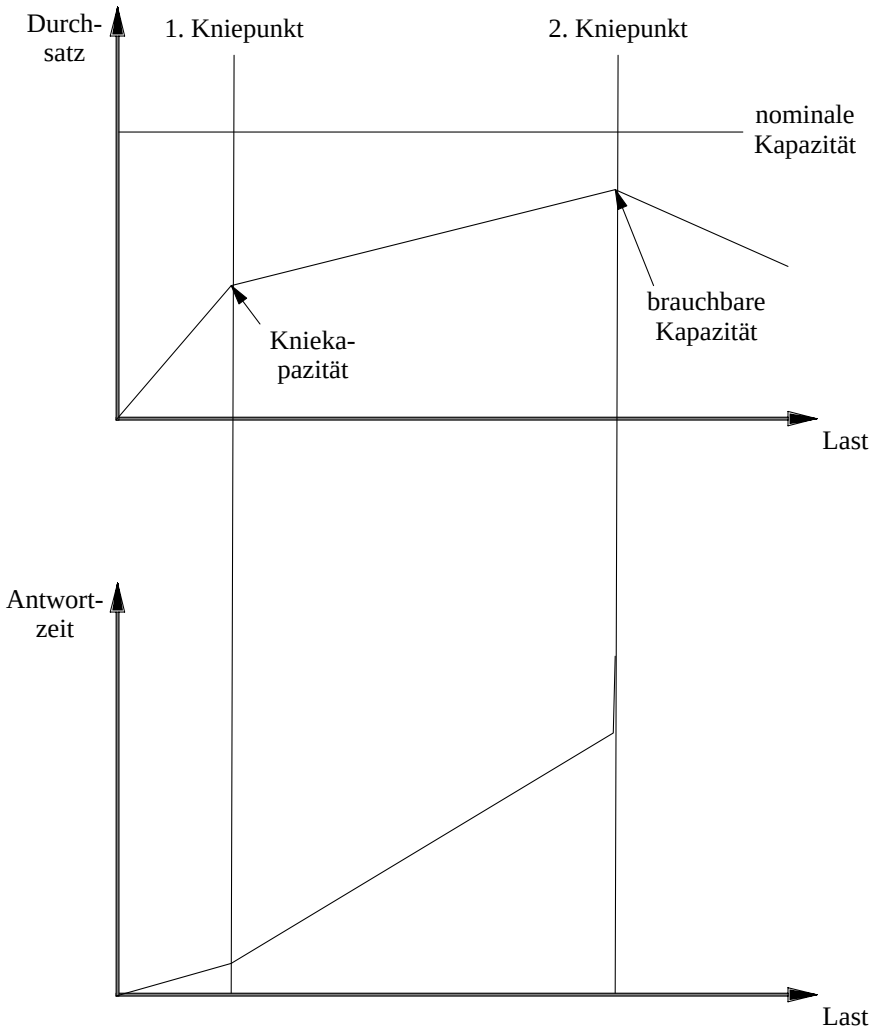

Abb. 4: Durchsatz- und Antwortzeit-Lastkurve

Definition Auslastung [Jain91]:

Die Ressourcenauslastung (utilization) U ergibt aus dem Verhältnis von Beschäftigungszeit (busy time) B der Ressource und der Länge T des Beobachtungsintervalls.

$$U = \frac{B}{T}$$

Das Auslastungsverhältnis aus busy time und Beobachtungszeit ist in abgewandelter Form auf Ressourcen anzuwenden (z.B. Speicher), die ständig genutzt werden, dabei aber unterschiedlich stark ausgelastet sind. Die Auslastung wird angegeben, indem der durchschnittlich genutzte Teil der Ressource (z.B. Speicherseiten) während einer Periode ermittelt wird.

Die Zeit, in der eine Ressource untätig ist, wird als idle time bezeichnet. Der Fall, daß ein Rechner untätig ist, während andere Rechnerknoten stark ausgelastet sind, soll durch die Lastbalancierung bzw. die Lastverteilung im Rechnersystem vermieden werden.

2.2.2. Ziele der Lastbalancierung

Einen Einfluß auf die Leistung eines verteilten Systems haben verschiedene Faktoren. Die Leistung wird sowohl von den auszuführenden Programmen und Anfragen (= Last) als auch von der Leistungsfähigkeit der Rechner, die die anstehende Last zu bearbeiten haben, beeinflußt. Während der Lastverteilung erfolgt die Zuteilung der Last zu den Rechnerknoten, um entsprechend den Zielvorstellungen der Nutzer eine Leistungsverbesserung zu erreichen. Bei den Zuteilungs- bzw. den Verteilungsentscheidungen sind der erforderliche Aufwand für die Ausführung der Anfragen und der Belastungszustand der Rechner im verteilten System zu berücksichtigen [Heis91] (siehe Abb. 5).

Der Vergleich und die Ermittlung der Effektivität von Lastbalancierungsalgorithmen erfolgt auf der Basis von aussagekräftigen Leistungskriterien. Die wichtigsten Leistungskriterien eines Lastausgleichsalgorithmus sind die für den Nutzer bemerkbaren und für dessen Anforderungen entscheidenden Leistungsmaße wie Antwortzeit und Durchsatz [Bono89]. Daraus lassen sich die Ziele eines optimalen Lastbalancierungsalgorithmus ableiten:

- Minimierung der Antwortzeit der Nutzeranforderungen
- Maximierung des Durchsatz des Gesamtsystems
- Minimierung der Lastunausgeglichenheiten
- Stabilität
- effiziente Systemressourcennutzung

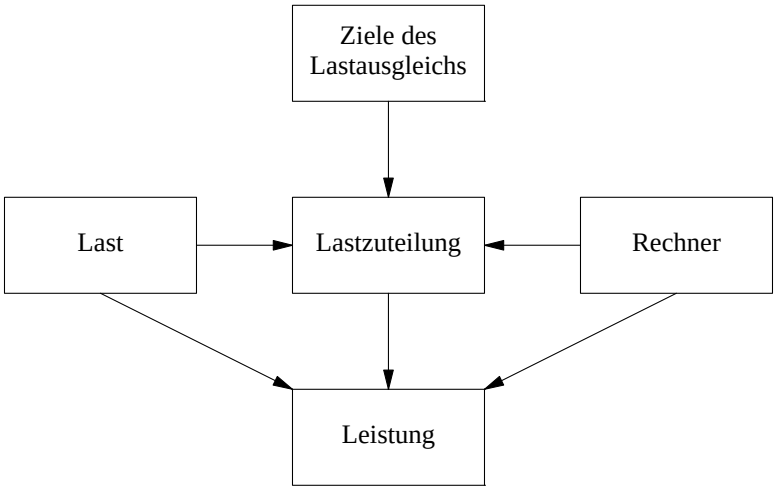

Abb. 5: Systemleistung beeinflussende Faktoren

Diese Zielstellungen sind errreichbar, indem die Last so verteilt wird, daß eine annähernd gleichmäßige Auslastung der verfügbaren Ressourcen im System entsprechend ihrer Leistungsfähigkeit angestrebt wird. Durch die Verteilung der vorhandenen Last soll eine Leistungssteigerung (hoher Durchsatz bzw. minimale Antwortzeit) im Vergleich zu einem unbalancierten System erzielt werden. In [Scha92] wird die Hypothese aufgestellt, daß das größte Potential für eine Leistungssteigerung in einem verteilten System in der Gleichverteilung der Rechenlast der aktiven Prozesse s_i auf alle verfügbaren Rechnerknoten bei Berücksichtigung deren Leistungsfähigkeit liegt (siehe Abb. 6).

In HEAD wird vordergründig eine Minimierung der Antwortzeit der einzelnen Datenbank-Anfragen angestrebt. Dazu werden die Teilanfragen auf die vorhandenen Rechnerknoten derart verteilt, daß die verfügbaren Ressourcen effektiv genutzt werden. Die Maximierung des Systemdurchsatzes ist bei der Lastbalancierung bzw. der Optimierung der Anfragebearbeitung kein unmittelbares Ziel. Diese Einschränkung erwächst daraus, daß für die an der Lastbalancierung beteiligten Lastkomponenten keine globale Sicht auf das gesamte System angestrebt wird. Dadurch erfolgt keine aufeinander abgestimmte Optimierung bei unabhängig voneinander gestarteten Datenbank-Anfragen. Eine faire Aufteilung der Ressourcen bzgl. der konkurrierenden Datenbank-

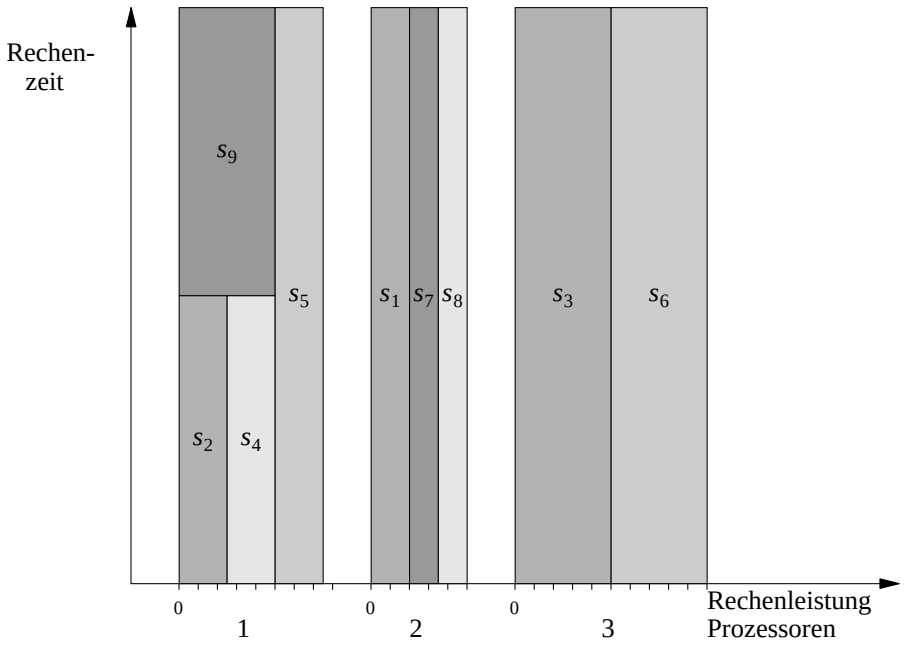

Abb. 6: Ideale Prozeßaufteilung [Flac93a]

Anfragen ist auf Grund der fehlenden Belastungsinformationen über das gesamte System nicht möglich (siehe Abs. 5.4.2.).

2.3. Elemente und Größen in der Lastbalancierung

Im folgenden werden Elemente und deren Eigenschaften, die bei der Lastverteilung und der Lastbalancierung zu berücksichtigen sind [Beck92], dargestellt (siehe Abb. 7). Die Eigenschaften der Elemente können in statische und dynamische unterteilt werden. Die statischen Eigenschaften stellen Parameter dar, die sich während der Laufzeit nicht verändern. Die Kenntnisse über diese statischen Eigenschaften können als bekannt vorausgesetzt werden. Dieses sogenannte a-priori Wissen über die Systemelemente ist als eine Grundlage für die Verteilungsentscheidung zu berücksichtigen. Die dynamischen Eigenschaften lassen sich erst zur Laufzeit wertmäßig mittels Messungen spezifizieren. Aus den dabei gewonnenen Informationen können Aussagen über die aktuelle Belastung der Elemente abgeleitet werden.

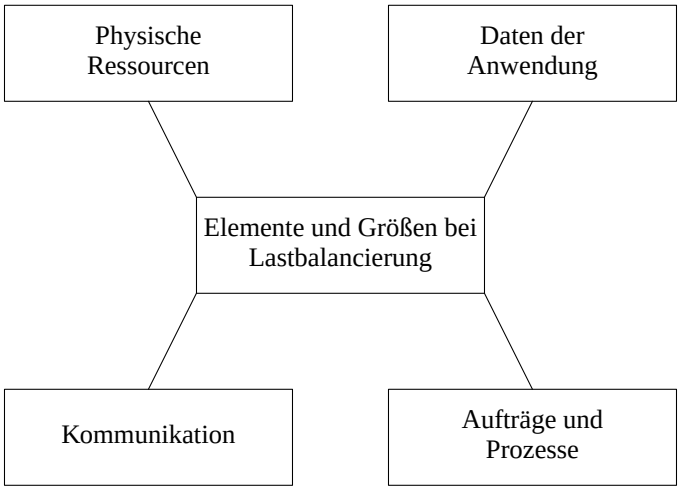

Abb. 7: Elemente und Größen bei der Lastbalancierung

2.3.1. Die physische Ressourcen

Die zu bearbeitenden Prozesse und Aufträge belasten den Rechner und das Netzwerk dahingehend, daß sie Systemressourcen beanspruchen. Damit die im Rahmen der Lastbalancierung getroffenen Verteilungsentscheidungen sich an der Belastung und den Fähigkeiten der Systemressourcen orientieren, sind Kenntnisse über verschiedene Ressourcen notwendig. Folgende Hardwarekomponenten des Systems können im Zusammenhang mit der Lastbalancierung betrachtet werden:

- Prozessor
- Hauptspeicher
- E/A - Geräte
- Netzwerk (Kanäle)

Bei einem heterogenen System, wie dem HEAD-Projekt, ist eine einheitliche Beschreibung der verfügbaren Ressourcen erstrebenswert. Diese Beschreibung bildet eine Grundlage für eine effiziente Lastbalancierungsstrategie. Jede Ressource besitzt statische, durch die Bauart bedingt, und dynamische Eigenschaften (siehe Tab.).

Ressource	statische Eigenschaften	dynamische Eigenschaften
Prozessor	Taktrate Betriebssystem-Typ max. Anzahl an Prozessen Verfügbarkeit	laufende/wartende Prozesse Auslastung
Speicher	Hauptspeichergröße zugehöriger virt. Speicher	Hauptspeicherbelegung Swapping-Rate
E/A-Gerät	Gerätetyp Datenrate Verfügbarkeit	wartende Prozesse Auslastung
Kanal	Datenrate Paketgröße Verfügbarkeit	Auslastung Länge der Nachrichtenwarteschlange

2.3.2. Die Daten der Anwendung

Die Prozesse und Anfragen greifen während ihrer Abarbeitung auf Daten zu. Daher sind bei der Auftragsausführung die unterschiedlichen Spezifika der Daten hinsichtlich der Sichtbarkeits- und der Gültigkeitsbereiche sowie bzgl. der Zugriffs- und der Konsistenzbedingungen zu berücksichtigen.

Im Zusammenhang mit der Lastbalancierung sind der Ort der Datenabspeicherung und das Zugriffsverhalten auf die Daten von besonderer Bedeutung. Für die Zugriffskosten ist der Ort der Datenabspeicherung ausschlaggebend. Um diese Kosten für mehrere Aufträge gering zu halten, werden Kopien (Replikate) der Daten auf mehreren Rechnerknoten abgelegt. Dies ermöglicht eine Leistungssteigerung bei parallelen Lesezugriffen. Bei parallelen Änderungsoperationen wird hingegen durch eine große Anzahl von Replikaten die Leistung gemindert, da die Konsistenzanforderungen eingehalten werden müssen. Einfluß auf die Lastverteilung hat auch die Granularität der Daten, denn eine feinere Aufgliederung der Daten bewirkt ein größeres Potential an möglicher Datenparallelität. Zu beachten ist, daß gleichzeitig ein größerer Overhead verursacht wird.

Nachfolgend wird eine kurze Zusammenfassung der statischen und dynamischen Eigenschaften der Daten dargestellt (siehe Tab.).

statische Eigenschaften	dynamische Eigenschaften
Speicherbedarf	Ort der Datenabspeicherung
Zugriffsbedingungen	Ort der Kopien
Konsistenzbedingungen	Zugriffshäufigkeit und -art (lesen/ändern)
geforderte Verfügbarkeit	

2.3.3. Die Kommunikation

Da der Lastausgleich in einem verteilten System realisiert werden soll, ist die Kommunikation zwischen den Prozessen auf unterschiedlichen Rechnerknoten zu beachten. Bei der Kommunikation kommt es zu einem Austausch von Nachrichten, der wiederum eine zusätzliche Belastung des Systems verursacht. Die Kanäle, über die der Informationsaustausch erfolgt, sind zu betrachten, da sie zu einem Engpaß im System werden können. Grundsätzlich ist zu berücksichtigen, daß in die Prozeßausführungszeit die Kommunikationszeiten mit einfließen. Von Bedeutung kann für die Lastverteilung ebenfalls sein, ob bei der Kommunikation Punkt-zu-Punkt Verbindungen oder Multicast- / Broadcast-Mechanismen genutzt werden und ob der Nachrichtenaustausch verbindungsorientiert oder verbindungslos erfolgt.

Folgende dynamische und statische Eigenschaften der Kommunikation können herausgestellt werden (siehe Tab.).

statische Eigenschaften	dynamische Eigenschaften
geforderte Datenrate	bestehende Verbindungen
verbindungsorientiert/verbindungslos	Nachrichtenaufkommen (Menge, Häufigkeit)
geforderte Verfügbarkeit	Warteschlangenlänge beim Sender/Empfänger

2.3.4. Die Aufträge und Prozesse

Wie bereits mehrfach erwähnt, besteht ein Ziel der Lastbalancierung in der Antwortzeitverkürzung der zu bearbeitenden Aufträge. Da ein Auftrag meistens aus mehreren kooperierenden Teilaufgaben bzw. Prozessen besteht, müssen zur Lastverteilung

Kenntnisse über die Ressourcenanforderungen und die gegenseitigen Abhängigkeiten der Teilaufgaben vorhanden sein. Dabei liefert die Beantwortung folgender Fragen notwendige Informationen zu den Aufträgen:

- Wann soll welcher Teilauftrag ablaufen?
- Welche Reihenfolgebeziehung der Aufträge untereinander sind zu beachten?

Indem der Ressourcenbedarf eines Einzelauftrages abgeschätzt wird, kann ein Lastprofil für den Auftrag erstellt werden. Diese Vorabschätzung der wahrscheinlich entstehenden Lastgrößenordnung ist erforderlich, da die tatsächlich verursachte Last erst zur Laufzeit meßbar ist. Die Kenntnisse über die wahrscheinliche Belastung des Systems durch den Auftrag bzw. Prozeß sind für die zu treffenden Verteilungsentscheidungen während des Lastausgleichs von Bedeutung.

Daher ist es notwendig die Eigenschaften der Aufträge und Prozesse zu berücksichtigen (siehe Tab.).

statische Eigenschaften	**dynamische Eigenschaften**
mittlerer Rechenzeitbedarf	Anzahl und Ort der ausführenden Instanzen
maximal zulässige Laufzeit	bisherige Datenzugriffe
Zugriffe auf feste Datensätze	bisherige Kommunikation mit anderen Prozessen
Startzeitpunkte der Aufträge	mittlere Ausführungszeit
Reihenfolgebeziehungen	bisherige Kooperation
Graph gegenseitiger Aufrufe	

2.4. Lösungsansätze für auftretende Probleme bei der Lastbalancierung

Die Lastkomponente kann bei der Erfüllung der Aufgaben und Dienste im Rahmen des Lastausgleichs mit verschiedenen Problemen und Schwierigkeiten (siehe Abb. 8) konfrontiert werden [Beck92], die es gilt zu lösen. Aus diesem Grund müssen die möglicherweise auftretenden Probleme bereits bei der Realisierung der Lastkomponente Beachtung finden.

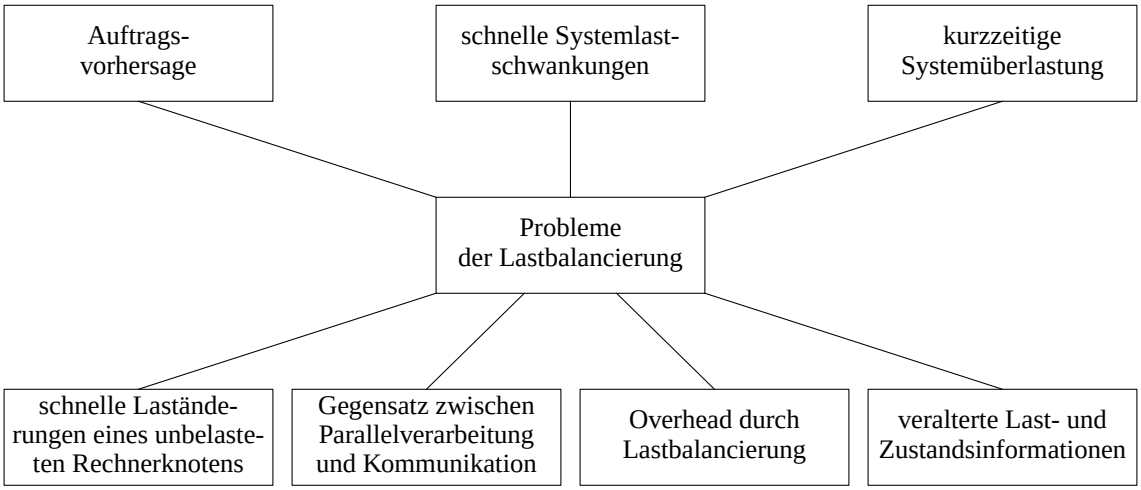

Abb. 8: Probleme bei der Lastbalancierung

2.4.1. Auftragsvorhersage

Durch den Lastausgleich wird eine Balancierung der anstehenden Aufgaben auf die vorhandenen Ressourcen angestrebt wird. Daher sind bei der Verteilung der Prozesse Informationen über die wahrscheinliche Belastung der Knoten, die durch einen Auftrag verursacht wird, zu berücksichtigen. In diesem Zusammenhang werden Vermutungen und Annahmen über das Laufzeitverhalten der Aufträge getroffen. Grundlage dafür bieten Kenntnisse über die statischen Eigenschaften der Aufträge, wie z.B. mittlerer Rechenzeitbedarf und Zugriffe auf feste Datensätze (siehe Abs. 2.3.4.). Auf der Basis des für den Auftrag abgeschätzten Lastprofils und der Informationen über den aktuellen Lastzustand der Rechner wird die Verteilungsentscheidung gefällt. Je mehr Informationen über den zu bearbeitenden Prozeß verfügbar sind, um so genauer kann das Lastprofil erstellt werden, und desto wirkungsvoller ist die Entscheidung betreffs der Zuordnung des Prozesses zu einem Prozessor. Liegt der Lastbalancierung ein dynamisches Verfahren zu Grunde (siehe Abs. 3.1.1.), so kann auch während der Auftragsabarbeitung auf Differenzen zwischen der im voraus erstellten Prognose über die Last (Lastprofil) und dem tatsächlichen Lastverhalten reagiert werden. Dabei wird die Lastbalancierungsentscheidung korrigiert bzw. erfolgt die Zuteilung weiterer Aufträge und Prozesse entsprechend der momentanen realen Lastsituation.

2.4.2. Schnelle Systemlastschwankungen

Schnelle Schwankungen in der Systemlast können durch folgende Ereignisse verursacht werden:

- Beendigung großer Aufträge
- Start vieler neuer Aufträge
- Aufträge ändern ihr Lastverhalten (z.B. Wechsel zwischen E/A-Phase und Rechenphase)

Dies führt dazu, daß Aufträge auf einem bisher gut balancierten System plötzlich ungünstig verteilt sind. Mögliche Reaktionen der Lastbalancierungskomponente sind:

- Ist eine Übertragung von Aufträgen möglich, dessen Abarbeitung bereits begonnen hat, kann eine Lastbalancierung durch den Transfer von großen Aufträgen auf gering- bzw. unbelastete Prozessoren erreicht werden.
- Sind viele kleine parallel ausführbare Prozesse vorhanden, so existieren kaum lange laufende Aufträge. In diesem Fall kann die Lastkomponente weitere Aufträge entsprechend der neuen Situation verteilen und somit auf Lastschwankungen reagieren. Kurze Lastspitzen sind häufig nicht zu vermeiden.
- Besteht die Möglichkeit, die laufende Ausführung von Aufträgen abzubrechen, so können diese auf weniger belasteten Knoten unter der Berücksichtigung vorhandener Konsistenzbedingungen neu gestartet werden.

2.4.3. Kurzzeitige Systemüberlastung

Ursachen für kurzzeitige Überlastungen von Rechnerknoten sind z.B.:

- Ausfall eines Rechnerknotens
- große Menge von Aufträgen, die plötzlich zur Bearbeitung anstehen

Auf eine kurzzeitige Systemüberlastung kann die Lastkomponente reagieren, indem versucht wird, unwichtige Aufträge zu stoppen und neu hinzukommende abzuweisen. Dabei muß aber sichergestellt werden, daß keine wichtigen Aufträge behindert werden. Die Lastkomponente muß sich bei ihren Entscheidungen von der Priorität der Aufträge leiten lassen.

2.4.4. Schnelle Laständerung eines bisher unbelasteten Rechnerknotens

Bei der Lastbalancierung werden Aufträge bzw. Prozesse grundsätzlich von einem schwer belasteten zu einem schwach bzw. unbelasteten Prozessor transferiert. Daher sind die gering belasteten Knoten das Ziel der Prozeßübertragung. Wird ein solcher Knoten von mehreren überlasteten Prozessoren als Zielknoten erkannt, kann dies dazu führen, daß dieser zuvor schwach belasteter Knoten wenig später selbst überlastet ist.

Bei einem zentralen Lastbalancierungsverfahren (siehe Abs. 3.1.3.) kann dieser Erscheinung wie folgt entgegen gewirkt werden. Bei der Zuteilung von Aufträgen zu einem Prozessor wird durch die zentrale Verteilungsinstanz der augenblickliche Belastungszustand des Knotens unter Berücksichtigung der vermuteten Last des neuen Auftrages aktualisiert.

Die Überlastung eines bisher un- bzw. geringbelasteten Rechnerknotens ist auf Grund der nicht vorhandenen zentralen Verteilungsinstanz bei dezentralen Lastausgleichsverfahren (siehe Abs. 3.1.3.) schwieriger zu vermeiden. Eine Möglichkeit besteht darin, daß nicht nur der am geringsten belastete Knoten das Ziel für alle Prozeßübertragungen ist. Die Aufträge werden auf mehrere gering belastete Knoten in Abhängigkeit der Rechnerknotenbelastung aufgeteilt.

2.4.5. Gegensatz Parallelverarbeitung und Kommunikationsbedarf

Einerseits ist durch die Steigerung der parallelen Abarbeitung von Aufträgen eine Verbesserung der Leistung bzw. eine Verkürzung der Bearbeitungszeit von Anwendungen realisierbar. Andererseits steigt mit zunehmender Parallelität bei der Prozeßausführung die erforderliche Nachrichtenmenge, die sowohl zwischen den einzelnen Prozessen auf dem lokalen Rechnerknoten als auch auf verschiedenen Rechnern auszutauschen sind. Jede Kommunikation verursacht zusätzlichen Aufwand und verringert die Systemleistung. Daher ist es notwendig, daß bei der Prozeßverteilungsentscheidung im Zusammenhang mit der Lastbalancierung der wahrscheinliche Kommunikationsaufwand berücksichtigt wird.

2.4.6. Overhead durch Lastbalancierung

Die Lastbalancierungskomponente erzeugt selbst eine Last, die Systemressourcen beansprucht und daher die Leistung beeinflußt. Ursachen dafür sind die Lastmessungen, der Lastinformationsaustausch, die Berechnungen für die Lastverteilungsentscheidungen und das Sammeln von Lastinformationen der Nachbarknoten durch die lokale Lastkomponente. Der zusätzliche Aufwand ist aber erforderlich, um die Last auf die verfügbaren Rechnerknoten derart zu verteilen, daß dadurch eine Verringerung der Gesamtabarbeitungszeit der Aufträge und Prozesse ermöglicht wird. Um zu vermeiden, daß die Ausführungszeiten länger werden als bei einem unbalancierten Vorgehen, sollten bei einem dynamischen Lastbalancierungsverfahren folgende Ansätze berücksichtigt werden:

- Der Aufwand für jede Messung kann minimal gehalten werden, indem eine Beschränkung auf entscheidende und aussagekräftige Systemgrößen erfolgt.
- Da durch jede Messung ein Overhead verursacht wird, ist das Meßintervall an die Lastwechselzyklen des Systems anzupassen. Dadurch werden unnötige Messungen vermieden. Trotzdem kann rechtzeitig auf einen Laständerung reagiert werden.
- Um den Aufwand für die Lastbalancierungsentscheidungen zu verringern, ist der Zeitpunkt für die Entscheidungsfindung ebenfalls den Lastwechselzyklen im System anzupassen.
- Bei der Realisierung der Aufgaben der Lastkomponente ist stets auf das richtige Verhältnis von Aufwand und Nutzen zu achten.
- Mittels einer dezentralen Lastbalancierung, die sich jeweils auf Teilgebiete des Gesamtsystems beschränkt, kann ein einfacher und schnellerer Lastausgleich erreicht werden.
- Zur Verringerung des Kommunikationsoverhead sollten Festlegungen getroffen werden, ab wann eine Benachrichtigung der entfernten Lastkomponenten über einen Lastwechsel auf dem lokalen Rechnerknoten erforderlich ist.

2.4.7. Veralterte Last- und Zustandsinformationen

Ändert sich die Last in einem System häufig, besteht die Gefahr, daß die Lastkomponente über veralterte und falsche Daten bezüglich des Belastungszustand im System verfügt. Werden auf dieser Grundlage Verteilungsentscheidungen getroffen, kann dies zur Folge haben, daß uneffektive Prozeßübertragungen ausgeführt werden. Demgegenüber kann durch einen zu häufigen Informationsaustausch, um alle beteiligten Knoten auf dem aktuellen Belastungszustand zu halten, ein nicht geringer Overhead verursacht werden. Ein großer Kommunikationsaufwand zum Lastinformations-austausch im Rahmen der Lastbalancierung kann das System stark belasten, so daß die Realisierung des Lastausgleichs nicht mehr effektiv ist. Um dies zu vermeiden, ist ein Grenzwert festzulegen, ab wann eine Laständerung als so bedeutend angesehen wird, daß die Nachbarknoten darüber verständigt werden müssen. Es sind Entscheidungen darüber zu treffen, welche Aktualitätsanforderungen an die Lastinformationen gestellt werden und ob alle Rechnerknoten des Systems in den Lastinformationsaustausch des lokalen Rechners einzubeziehen sind.

3. Klassifikation von Lastausgleichsverfahren

In der Literatur werden eine Vielzahl von Lastbalancierungsverfahren vorgeschlagen. Um diese Verfahren miteinander vergleichen zu können, wird nachfolgend die Möglichkeit einer Klassifikation dargestellt. Entsprechend diesem Klassifikationsschema kann eine Einordnung der verschiedenen Ansätze aus der Literatur sowie des Lastausgleichsverfahren des HEAD-Systems vorgenommen werden.

3.1. Schema zur Klassifikation von Lastausgleichsverfahren

In [Scha92] wird ein Klassifikationsgerüst zur Einordnung der Lastausgleichsverfahren beschrieben. Bevor die Kriterien erläutert werden, nach denen die Einteilung bzw. die Unterscheidung der Verfahren erfolgt, soll ein Überblick zur Klassifikation gegeben werden (siehe Tab.).

3.1.1. Statische oder dynamische Lastbalancierungsverfahren

Bei statischen Lastbalancierungsverfahren erfolgt die Zuteilung von Prozessen eines Auftrages zu den Prozessoren einmalig vor Beginn der Auftragsabarbeitung. Neben der Plazierung von Prozessen können die Reihenfolge und der Zeitpunkt einzelner Auftragsbearbeitungen festgelegt werden. An dieser Zuordnung ändert sich während der Ausführung des Auftrages nichts, sie bleibt statisch. Auch trotz starker Änderungen des Systemlastzustandes wird an der zu Beginn getroffenen Entscheidung hinsichtlich der Prozeß-Prozessor-Zuteilung festgehalten. Merkmale dieser Verfahrensweise sind ein geringer Overhead, relative Einfachheit sowie ein mittleres Potential zur Leistungssteigerung. Einen entscheidenden Einfluß auf die Wirksamkeit der im Vorfeld der Prozeßbearbeitung getroffenen Zuteilungsentscheidung hat die vorhandene und berücksichtigte Menge an a-priori Wissen über den auszuführenden Prozeß.

Bei dynamischen Lastausgleichsverfahren können während der Ausführung die zu Beginn der Auftragsabarbeitung getroffenen Entscheidungen über die Zuordnung der Prozesse zu Prozessoren geändert werden. Diese Änderungen bezüglich der Prozeßzuteilung beruhen auf Informationen über die aktuelle Belastung einzelner Rechnerknoten. Die dynamischen Verfahren besitzen einen größeren Overhead als statische Verfahren. Gleichzeitig müssen bei der Verlagerung von Prozessen die Prozeßkontextinformationen

Verfahren	Kriterium	Merkmal
statisch	Zuordnung der Prozesse zu Prozessoren	einmalig, vor Beginn der Auftragsbearbeitung
dynamisch		mehrmalig, auch während der Auftragsbearbeitung
optimal	Optimierung der Kostenfunktion bei der Prozeßzuordnung zu Rechnerknoten	bezieht sich auf gesamten Lösungsraum
suboptimal		bezieht sich auf Teil des Gesamtlösungsraumes mit eingeschränkter Suchtiefe
zentral	Ort der Verteilungsinstanz	ein einziger Rechnerknoten im verteilten System
dezentral		mehrere im Netz verteilte Rechnerknoten
senderinitiiert	Ausgangspunkt der Initiative zum Prozeßtransfer	überlasteter Rechner, der an andere Rechnerknoten Prozesse senden will
empfängerinitiiert		unterbelasteter Rechner, der von anderen Rechnerknoten Prozesse empfangen will
kooperativ	Grad der Einbeziehung von Umgebungsinformationen des Rechnerknoten	groß, Zuhilfenahme von Kenntnissen über Zustand der anderen Rechnerknoten
autonom		klein, Zustand anderer Rechnerknoten bleibt unbeachtet
adaptiv	Anpassung der Verteilung nach Reaktionen auf vorherige Entscheidungen	ja, Nutzung von Erfahrungen vorangegangener Entscheidungen
nichtadaptiv		nein, unabhängig von Systemreaktionen

mit übertragen werden. Die Menge an zu transferierenden Kontextinformationen ist bei einem laufenden Prozeß wesentlich größer als bei einem Prozeß, dessen Abarbeitung noch nicht begonnen hat. Der Hauptvorteil der dynamischen Verfahren besteht darin, daß während der Ausführungszeit Entscheidungen über die Prozeßverteilung auf der Grundlage von Kenntnissen über den aktuellen Systemzustand getroffen werden können. Dadurch wird eine Balancierung der Last ermöglicht.

3.1.2. Optimale oder suboptimale Lastausgleichsverfahren

Bei optimalen Verfahren ist das Ziel der Zuordnung von Prozessen zu Rechnerknoten, eine optimale Zuteilung bezüglich einer gegebenen Kostenfunktion zu finden. Dafür

sind umfangreiche Informationen über den Zustand des Systems und über den Bedarf an Betriebsmitteln der einzelnen Prozesse notwendig. Mögliche Ziele der Kostenfunktion sind:

- minimale Gesamtausführungszeit eines Auftrages
- maximale Betriebsmittelauslastung
- maximaler Systemdurchsatz

Das Erreichen einer optimalen Lösung ist häufig mit einem großen Aufwand verbunden.

Ein Verfahren ist suboptimal bezüglich einer Kostenfunktion, wenn auf Grund unzureichender Informationen bzw. einer zu großen Komplexität statt einer optimalen eine "hinreichend gute" Lösung akzeptiert wird. Mittels approximativer und heuristischer Verfahren ist es möglich, suboptimalen Lösungen zu erhalten.

Approximative Verfahren streben wie optimale Verfahren eine optimale Zuteilung bzgl. einer gegebenen Kostenfunktion an. Um dies zu erreichen, wird aber die Suchtiefe eingeschränkt. Durch diese Maßnahme ist keine Überprüfung des gesamten Lösungsraumes erforderlich. Innerhalb des Teilraumes wird versucht, ein Optimum zu erzielen. Somit ist der erforderliche Aufwand zur Erreichung der Lösung im Vergleich zu optimalen Verfahren geringer. Wie bei optimalen Verfahren sind ausreichende Informationen über das System notwendig.

Bei heuristischen Verfahren sind relativ wenig Vorabkenntnisse über die Systemlast und den Prozeßbetriebsmittelbedarf vorhanden. Heuristiken liegen häufig Vermutungen über Zusammenhänge zwischen der zu optimierenden und anderen Größen zu Grunde. Heuristische Verfahren nutzen das verfügbare Wissen, um eine hinreichend gute Lösung zu finden.

3.1.3. Zentrale oder dezentrale Lastbalancierungsverfahren

Bei einem zentralen Lastausgleichsverfahren trifft ein einziger Rechnerknoten im verteilten System die Verteilungsentscheidungen. Dies birgt die Gefahr einer Engpaßbildung in sich, da das gesamte Systems vom Funktionieren und dem Belastungszustand dieses einen Rechners abhängt. Bei der Sammlung und der Verwaltung der Belastungsinformationen an einem zentralen Ort besteht das Risiko, daß die Entschei-

dungen bzgl. der Verteilung auf Informationen basieren, die auf Grund der möglichen Übertragungsverzögerung bereits unaktuell sind [Gosc91]. Zentrale Verfahren sind einfach zu implementieren und bieten gute Analysemöglichkeiten.

Gibt es mehrere über das Netzwerk verteilte Instanzen, die Lastbalancierungsentscheidungen treffen, liegt ein dezentrales Verfahren vor. Jede Lastkomponente kann in diesem Fall entweder nur für einen Rechnerknoten oder für ein Teilnetz von Knoten verantwortlich sein. Dezentrale Verfahren erzwingen einen großen Kommunikationsaufwand und erzeugen dadurch einen entsprechenden Overhead. Ursache dafür ist der erforderliche Informationsaustausch zwischen den Lastkomponenten, die für den Ausgleich der Last im gesamten System verantwortlich sind.

3.1.4. Sender- oder empfängerinitiierte Lastausgleichsverfahren

Eine Unterscheidung in sender- oder empfängerinitiierte Lastbalancierung setzt ein dezentrales Verfahren voraus. Prozesse werden im Zusammenhang mit der Lastbalancierung von überlasteten zu unterbelasteten Prozessoren transportiert, also vom Sender zum Empfänger. In Abhängigkeit davon, von welchem Rechnerknoten die Initiative zum Prozeßtransfer ausgeht, lassen sich die Lastausgleichsverfahren in sender- oder empfängerinitiierte einteilen.

Beim erstgenannten versucht der überlastete Rechner (der Sender) einen weniger belasteten Rechnerknoten zu finden, auf den er Prozesse übertragen kann. Beim empfängerinitiierten Vorgehen bietet sich ein schwachbelasteter Prozessor (der Empfänger) an, Prozesse zu übernehmen.

Wird die Warteschlangenlänge einer Ressource bzw. die Anzahl von Prozessen in dieser Warteschlange als Lastmaß genutzt, so erfolgt die Entscheidung zur Prozeßverlagerung beim Auftreten folgender Ereignisse:

- Bei einem senderinitiierten Verfahren ist die Ankunft eines Auftrages bzw. Prozesses im System entscheidend. Zu diesem Zeitpunkt erhöht sich die Warteschlangenlänge und der Knoten kann überbelastet sein. Dadurch ist der Anlaß für eine Prozeßverlagerung gegeben.
- Die Entscheidung zum Prozeßtransfer wird bei empfängerinitiierten Verfahren beim Abgang eines Prozesses getroffen. Durch dieses Ereignis verringert sich die

Länge der Warteschlange und der Knoten kann schwach belastet sein [Wang85].

3.1.5. Kooperative oder autonome Lastbalancierungsverfahren

Um eine Einteilung in kooperative oder autonome Verfahren vornehmen zu können, werden dezentrale Verfahren vorausgesetzt. Für die Unterscheidung ist es notwendig, folgende Fragen zu beantworten:

- Welche Informationsmenge über die Umgebung eines Rechnerknoten fließt in die Verteilungsentscheidung mit ein?
- Wie groß ist der Abstimmungaufwand der beteiligten Rechnerknoten?

Bei einem autonomen Verfahren ist die Kommunikationsintensität gering, denn der Zustand anderer Rechnerknoten im Netz wird nicht beachtet. Entscheidungen werden auf der Grundlage der eigenen Belastung und des statischen Wissens über das Rechnersystem (z.B. Topologie und Rechenleistung anderer Rechnerknoten) getroffen. Die autonom erzielten Verteilungsentscheidungen können im Widerspruch zu auf anderen Knoten getroffenen Entscheidungen stehen und somit Ressourcenkonflikte verursachen.

Bei kooperativen Verfahren werden die Entscheidungen mit größerem Kommunikationsaufwand getroffen. Der Grad der Zusammenarbeit kann dabei von der Einbeziehung der Lastinformationen nur eines Rechnerknotens bis zu kompletten Vergleichsverfahren mit allen Rechnerknoten des Netzes reichen. Als schwierig erweist sich die Festlegung, welche Menge an Informationen über die anderen Knoten zu berücksichtigen sind. Dabei ist ein Kompromiß zu finden zwischen der Notwendigkeit von umfassenden sowie aktuellen Informationen, um eine richtige Verteilungsentscheidung zu treffen, und dem Aufwand bzw. dem Overhead, der durch jede Übertragung verursacht wird. Bei kooperativen Verfahren ist der Overhead größer als bei autonomen Verfahren. Andererseits bietet ein autonomes Verfahren kein großes Potential zur Leistungssteigerung. Bei einer starken Belastung der Rechnerknoten im Netz und autonomer Prozeßverlagerung durch die Knoten kommt es zu einer zusätzlichen Belastung des Systems, was zu einer Instabilität führen kann.

3.1.6. Adaptive oder nichtadaptive Lastausgleichsverfahren

Ein adaptives Verfahren nutzt Erfahrungen vorangegangener Verteilungsentscheidungen, indem eine Anpassung des Verteilungsalgorithmus mittels Parameterveränderung erfolgt. Dabei werden Parameterwichtungen in Abhängigkeit von den Reaktionen des Systems auf vorherige Entscheidungen geändert. Adaptive Verfahren werden auch als lernfähig bezeichnet. Bei einem nichtadaptiven Vorgehen werden dagegen Parameter und Balancierungsverfahren nicht angepaßt.

Ein solches Angleichen an sich ändernde Bedingungen bringt den Nachteil der höheren Komplexität für adaptive Verfahren mit sich. Bei adaptiven Verfahren ist der Overhead zu begrenzen, um das Antwortzeitverhalten nicht zu verschlechtern und um eine positive Kosten-Nutzen-Relation beizubehalten.

3.2. Einordnung von HEAD in das Klassifikationsschema

Durch Beobachtung der aktuellen Belastung eines Rechnerknotens kann ein **dynamisches** Verteilungsverfahren realisiert werden. Obwohl ein größerer Aufwand als bei einem statischen Vorgehen notwendig ist, kann dadurch wesentlich besser auf aktuelle Zustandsänderungen des Systems reagiert werden. Damit die einzelnen Werte des Lastindexes stets Auskunft über die aktuelle Beanspruchung des Rechnerknoten geben, wird durch die lokale Lastkomponente in regelmäßigen Zeitintervallen die Belastung ermittelt.

Ziel der Kostenfunktion ist auf Grund der Anwendungscharakteristik des HEAD-Systems (vorwiegend ad hoc - Anfragen) die Minimierung der Antwortzeit von Aufträgen [Flac93b]. Bei der Optimierung werden Einschränkungen betreffs der Suchtiefe des Lösungsraumes gemacht. Eine optimale Realisierung der Prozeß - Prozessor - Zuteilung in Bezug auf die gegebene Kostenfunktion wird für den beschränkten Lösungsraum der Domäne angestrebt (siehe Abs. 5.4.2.). Dadurch wird das angewendete Lastausgleichsverfahren als **suboptimal** klassifizierbar.

Jeder Rechnerknoten verfügt über eine lokale Lastkomponente. Dadurch können sich alle Knoten aktiv als Verteilungsinstanz am Lastausgleich beteiligen. Indem die anfallende Arbeit bzgl. des Lastausgleichs auf mehrere Rechnerknoten verteilt wird, kann verhindert werden, daß eine einzige zentrale Instanz zum Engpaß des Systems wird. Da im

Netz mehrere Knoten als Verteilungsinstanzen wirken, erfolgt die Lastbalancierung **dezentral**.

Zum Anlaß für einen Lastausgleich zwischen mehreren Knoten wird ein überlasteter Rechnerknoten genommen. Dieser schwerbelastete Knoten sucht einen weniger belasteten Knoten, der als Ziel für die Prozeßübertragung und die anschließende Prozeßverarbeitung geeignet ist. Dieses Vorgehen kann entsprechend dem Klassifizierungsschema als **senderinitiiert** eingestuft werden.

Zwischen den Rechnerknoten bzw. deren Lastkomponenten erfolgt eine stetiger Lastinformationsaustausch. Dadurch werden die Voraussetzungen geschaffen, daß globale Lastinformationen jederzeit lokal verfügbar sind und daß der Zustand anderer Knoten berücksichtigt werden kann. Bei diesem **kooperativen** Vorgehen werden Kenntnisse aus der Umgebung eines Rechnerknotens mit in die Entscheidungsfindung zur Verteilung der Belastung einbezogen.

Es wird angestrebt, daß die Lastkomponente aus den Erfahrungen über die Reaktionen des Systems auf vorangegangene Verteilungsentscheidungen lernt. Dadurch kann der Verteilungsalgorithmus unter Berücksichtigung dieser Kenntnisse angepaßt werden. Das Lastausgleichsverfahren ist somit als **adaptiv** einzuordnen.

Abschließend sind in der folgenden Übersicht die Merkmale des HEAD-Lastausgleichsverfahrens zusammenfassend dargestellt (siehe Abb. 9).

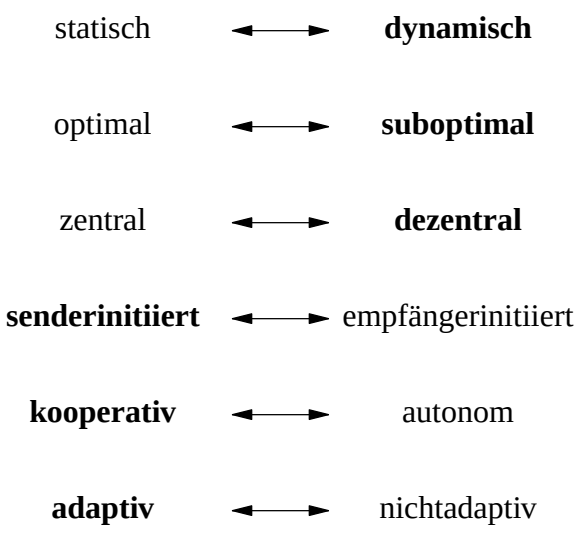

Abb. 9: Merkmale des HEAD-Lastausgleichsverfahrens

3.3. Einordnung weiterer Verfahren ins Klassifikationsschema

In der folgenden Zusammenstellung (siehe Tab.) erfolgt die Einordnung in der Literatur vorgeschlagener Lastausgleichsverfahren in das oben dargestellte Klassifikationsgerüst. Die Zuordnung eines Verfahrens zu einer Ausprägung bzgl. eines Kriteriums wird durch ein "*" verdeutlicht. Wird in der angegebenen Literatur ein Verfahren vorgestellt, das zu einem Kriterium beide Merkmalsausprägungen besitzt, so erfolgt eine Kennzeichnung mittels "*" in beiden Merkmalsspalten. Werden in einem Literaturartikel mehrere Verfahren dargestellt, die jeweils eine andere Merkmalsausprägung zum Kriterium aufweisen, so wird dies durch "+" in beiden Merkmalsspalten ausgedrückt.

Verfahren	**Kriterium**											
	stat.	dyn.	optm.	subo.	zent.	deze.	send.	empf.	koop.	auto.	adap.	n. ad.
HEAD		*		*		*	*		*		*	
[Bara85]		*		*		*	*		*			
[Baum89]	*			*		*	*		*			*
[Brya81]		*		*		*	*	*	*			*
[Casa87]		*		*		*	+	+	*			*
[Chou82]	*			*	*						*	
[Chwd90]	*			*		*	*			*		*
[Chu80]	*			*	*							
[Eage86a]	*			*		*		*		*		*
[Eage86b]	*			*		*	*		+	+		*
[Hac89a]	*			*		*	*		*			*
[Hac89b]		*		*	*							*
[Krue88]		*		*		*	*,+	+	*			*
[Lee86]	*			*		*	*			*	+	+
[Lela86]		*		*		*	*	*	*			*
[Livn82]	*			*		*	+	+	*			*
[Ni85]		*		*		*	*		*			*
[Rahm93]		*		*	*							*
[Shen85]	*		*		*							*
[Stan85a]	*			*		*	*		*		*	
[Stan85b]		*		*		*	*		*		*	
[Ston77]	*		*		*							*
[Tant85]	*		*		*							*
[Wang85]	*			*		*	+	+	+	+		*
[Zhou88]	*			*	+	+	+	+	+	+		*
[Zhou87]	*			*	+	+	*		+	+		*

4. Aufgaben der Lastkomponente

Um eine wirkungsvolle Lastbalancierung zu erzielen, müssen von der Lastkomponente die in den folgenden Unterabschnitten detailliert erläuterten Aufgaben (siehe Abb. 10) realisiert werden.

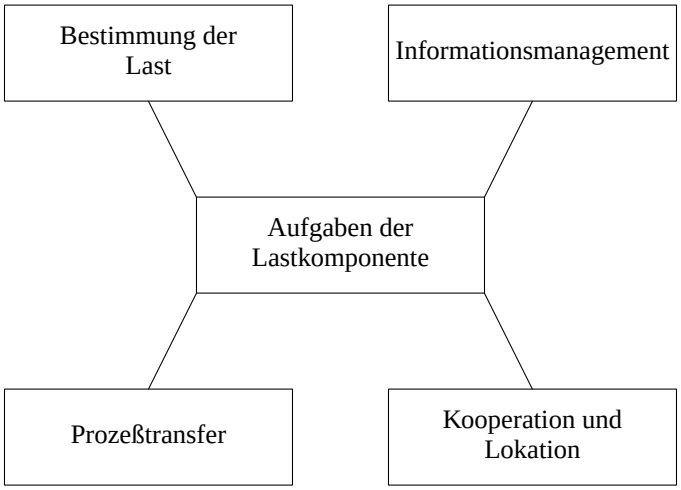

Abb. 10: Aufgaben der Lastkomponente

4.1. Lastbestimmung

4.1.1. Die Last eines Rechners

Um die Entscheidungen über die Prozeßverteilung im Zusammenhang mit der Lastbalancierung auf der Grundlage von Kenntnissen über die aktuellen Belastungen der Rechnerknoten im verteilten System zu treffen, muß eine Lastbestimmung erfolgen. Eine einzelne Größe, die die Last auf einem Rechner repräsentiert, ist nicht vorhanden. Daher ist es notwendig, um Informationen über die Belastung sowohl der Rechnerknoten als auch des Netzwerkes zu erhalten, verschiedene Größen und Parameter zu betrachten. Aus den ermittelten Lastwerten eines Rechners, die zusammen einen Lastindex bzw. Lastdeskriptor bilden, kann abgeleitet werden, unter welchen Verarbeitungsbedingungen die Ausführung eines Prozesses auf diesem Rechner erfolgt. In [Ferr85] wird davon ausgegangen, daß zwischen dem Lastindex *li* und der Antwortzeit *rt* eines Pro-

zesses eine funktionale Beziehung *rt(li)* besteht. Die Last auf einem Rechnerknoten beeinflußt die Antwortzeit eines Prozesses, der auf diesem Rechner abgearbeitet wird. Ist eine Funktion *rt(li)* für einen bestimmten Rechnerknoten vorhanden, so kann unter Nutzung des aktuellen Lastindex *li* die Antwortzeit *rt* für einen Prozeß, der auf diesem Rechner ausgeführt werden soll, abschätzt werden.

Durch Betrachtung der dynamischen Eigenschaften von Elementen und Größen, die für die Lastbalancierung von Bedeutung sind, können Aussagen über deren aktuelle Belastung getroffen werden. Um aber bauartbedingte bzw. konfigurationsabhängige Einflußgrößen der Hardware bei der Lastverteilung zu berücksichtigen, sind auch die Kenntnisse über statische Eigenschaften von Rechnerressourcen mit in die Entscheidungsfindung bzgl. der Lastbalancierung einzubeziehen [Zhou93, Ni85]. Gleichermaßen sind Informationen über die Daten und Aufträge zu berücksichtigen (siehe Abs. 2.3.).

In der Literatur sind verschiedene Ansätze zur Lastbestimmung aufgezeigt. Oft wird die genaue Definition der Last und deren Ermittlung umgangen bzw. ausgespart, indem abstrakt von der Last des Rechnerknotens ohne weitere Erläuterungen gesprochen wird. Die Prozesse, die auf einem Rechnerknoten abgearbeitet werden, belasten einen Rechnerknoten dahingehend, daß sie Rechner- und Systemressourcen, wie die CPU, den Speicher, die E/A-Geräte und das Kommunikationsmedium, beanspruchen. Da die CPU die wichtigste Ressource eines Rechners ist, die für die Ausführung der zu bearbeitenden Prozesse am entscheidendsten ist, rückt die CPU bei vielen Lastbestimmungsansätzen in den Mittelpunkt [Zhou86]. Wie und woraus können Kenntnisse über die Belastung von Systemressourcen gewonnen werden?

4.1.2. Die CPU-Belastung

In [Ni85] wird ein Lastabschätzungsprogramm vorgestellt, das Aussagen über die CPU-Belastung liefert. Dieses Programm läuft auf dem Rechnerknoten und ermittelt die Zeitintervalle zwischen den Perioden, in denen das Programm die CPU selbst beansprucht. Da die CPU im Timesharing-Verfahren allen Prozessen zugeteilt wird, kann von der Länge der Zeitintervalle zwischen der CPU-Zuteilung zum Programm die Last des Prozessors abgeleitet werden. Je größer das Zeitintervall, um so stärker ist die CPU belastet.

Aus der Länge der Warteschlangen von Rechnerressourcen kann ebenfalls auf die Belastung des Rechnerknotens geschlossen werden [Zhou86]. Die Länge der Warteschlange wird durch die Anzahl der Prozesse in der Warteschlange bzgl. einer Ressource definiert. Die CPU-Run queue (queue = Warteschlange) enthält alle Prozesse im lauffähigen Zustand. Die Länge der CPU-Run queue ist ein guter Indikator für die aktuelle Last der CPU und spiegelt somit den gegenwärtigen Lastzustand des Rechnerknotens wider [Zhou88, Baum89]. Erfolgen die Messungen der CPU-Run queue-Länge in kurzen Zeitabständen, werden die häufigen Schwankungen dieser Belastungsgröße sichtbar. Auf Grund der relativ schnell wechselnden Warteschlangenlänge ist deren Nutzung zur Vorhersage der zukünftigen Last bei der Lastbalancierung nicht geeignet.

Ob die absolute oder die geglättete bzw. die durchschnittliche Warteschlangenlänge zur Lastbestimmung genutzt werden sollte, konnte bisher nicht konkret ermittelt werden [Ferr85]. Grundsätzlich ist zu beachten, daß durch jede Messung der Warteschlangenlänge ein Overhead verursacht wird. Das UNIX-System führt selbständig eine exponentielle Glättung der CPU-Run queue-Länge über drei Zeitintervalle (1min, 5min, 15min) durch, die für die Lastbestimmung genutzt werden können. Durch das Glätten können plötzliche Lastschwankungen ausgeglichen werden, indem ein Durchschnitt über mehrere Lastwerte einer bestimmten Zeitperiode gebildet wird. In [Bono89] wird ein Algorithmus zur Ermittlung der durchschnittlichen CPU-Run queue-Länge dargestellt. In diesem Algorithmus wird der Durchschnitt aus mehreren während eines Meßintervalls ermittelten Proben der CPU-Run queue-Länge bestimmt. Die durchschnittliche CPU-Run queue-Länge n_k wird nach folgender Formel abgeschätzt:

Formel zur Berechnung der durchschnittlichen CPU-Run queue-Länge:

$$n_k = \sum_{j=1}^{v_t} \frac{q_k^{(j)}}{v_t}$$

verwendete Parameter:

n_k : asymptotische, durchschnittliche CPU-Run queue-Länge des k-ten Meßintervalls

t : Länge des Meßintervalls k

v_t : Gesamtanzahl von Proben (= Meßhäufigkeit) im Meßintervall k der Länge t; v = 50 - 100 pro Sekunde

$q_k^{(j)}$: Anzahl von Prozessen in der CPU-Run queue bei der j-ten Probe des k-ten Meßintervalls

Je häufiger Messungen in einem Meßintervall erfolgen, um so genauer werden die Abschätzungen.

Um die gesamte Rechner- bzw. Systemlast widerzuspiegeln, ist es nicht ausreichend, wenn nur die CPU-Run queue-Länge betrachtet wird [Cici92]. In dieser Warteschlangenlänge ist der Einfluß von Prozessen, die in der Ein- bzw. Ausgabe oder gesperrt sind, nicht enthalten. Die Anzahl der Prozesse in der CPU-Run queue kann sich schnell ändern, da viele Prozesse nur auf die Vollendung von E/A-Operationen warten. Für einen Augenblick wird aber mit der aktuellen Warteschlangenlänge der falsche Eindruck vermittelt, daß der Rechner leicht belastet sei. Eine einmalige Messung zur Ermittlung des Belastungswertes ist nicht ausreichend, um daraus konkrete Aussagen über den Lastzustand ableiten zu können [Bara93]. Das Problem der Verfälschung kann sich durch stattfindende Prozeßverlagerungen verschärfen. Die Prozesse, die zum Rechner übertragen werden, sind nicht in der Warteschlangenlänge enthalten. In einem virtuellen Lastwert können die Prozeßverlagerungen berücksichtigt werden. Der Betrag der aktuellen Warteschlangenlänge eines Rechners wird dazu um die Anzahl von Prozessen erhöht, die zu diesem Rechner momentan transferiert werden. Um ein Lastmaß mit geringeren Schwankungen nutzen zu können, kann der aktuelle Lastwert über eine Periode beibehalten werden, die mindestens die Länge einer Prozeßübertragung hat [Krue84].

4.1.3. Weitere Parameter der Lastmessung

Neben der CPU-Run queue-Länge kann auch die Gesamtanzahl an Prozessen auf einem Rechnerknoten als meßbarer Parameter genutzt werden, um die Rechnerbelastung abzu-

schätzen [Ni85]. Als alleiniges Kennzeichen für die Rechnerbelastung ist dieser Parameter jedoch nicht zu nutzen [Baum89].

Da sowohl die CPU-Run queue-Länge als auch die Gesamtanzahl an Prozessen auf einem Rechnerknoten als einzelne Maße zur umfassenden Widerspiegelung der Rechnerbelastung nicht ausreichend sind, sollten weitere Parameter und Größen bzw. Kombinationen mehrerer Maße in die Lastbestimmung einbezogen werden. In diesem Zusammenhang lassen sich die folgenden zwei Fragen ableiten:

- Wieviel Informationen über den Systemzustand sollte jede Lastverteilungsinstanz zur Verfügung haben, um wirkungsvolle Lastbalancierungsentscheidungen treffen zu können?
- Welche Lastparameter sollten auf jedem Rechnerknoten zur Lastbestimmung gemessen werden?

Einfache Algorithmen beschränken sich auf einen einzigen Lastparameter z.B. die Gesamtanzahl an Prozessen, auch wenn dadurch keine umfassenden Kenntnisse über die Belastung des Rechners vorhanden sind. Komplexere Methoden berücksichtigen Informationen sowohl über die Kommunikationskanalbelastung als auch über die Ressourcen, die neben der CPU bei der Prozeßausführung in Anspruch genommen werden. Als relevante Parameter für die Lastmessung sind Aussagen über die Nutzung von Ressourcen eines Rechnerknotens, wie CPU und Speicher, sowie des Netzwerkes, für die Lastbalancierung heranzuziehen [Ludw93] (siehe Tab.).

Diese Lastparameter bilden eine fundierte Grundlage, um aussagekräftige Informationen über die Belastung des einzelnen Rechnerknotens bzw. des Gesamtsystems zu erhalten. Bei der Wahl der zu messenden Lastparameter, aus denen Aussagen über die Belastung abgeleitet werden können, gibt es in der Literatur keine einheitliche Herangehensweise (siehe Tab.).

In [Zhou87] wird als Lastindex eine Kombination der Prozeßwarteschlangen von der CPU, dem Paging- und I/O-System verwendet. In Tabelle wird deutlich, daß trotz unterschiedlicher Ansätze grundsätzlich Informationen über die Nutzung der CPU, des Speichers und auch des Kommunikationskanals berücksichtigt werden.

Lastparameter des	
Rechnerknotens	**Prozesses**
Prozessor idle-time	Nutzungszeit der CPU
CPU-Run queue-Länge	Zeit in der CPU-Run queue
Sende-/Empfangs queue-Länge	Zeit in der Sende-/Empfangs queue
freier Speicher	genutzter Speicher
Menge von gesend./empfang. Daten	Menge von gesend./empfang. Daten
Anzahl von gesend./empfang. Nachrichten	Anzahl von gesend./empfang. Nachrichten
	Anzahl von Seitenfehlern (page faults)

4.1.4. Begrenzung des Lastmessungsoverhead

Bereits mit der Wahl der Meßgrößen bei der Lastbestimmung wird beeinflußt, wie effizient der gesamte Lastbalancierungsalgorithmus arbeitet. Ursache dafür ist die Beziehung zwischen der Wirksamkeit des Lastverteilungsalgorithmus und den im Zusammenhang mit der Lastbalancierung genutzten Systemparametern [Livn82].

Um eine wirksame Lastbalancierung zu erzielen, müssen einerseits umfassende und aussagekräftige Zustandsinformationen gesammelt sowie für die Entscheidungsfindung bereitgestellt werden. Auf der anderen Seite wird mit zunehmender Komplexität der Daten und Operationen bei der Lastbestimmung auch der verursachte Overhead vergrößert [Eage86b]. Dieser Mehraufwand kann sich nachteilig auf die Systemleistung auswirken. Daher ist ein Kompromiß zwischen Häufigkeit und Genauigkeit der Lastmessungen sowie dem dadurch hervorgerufenen Overhead zu finden [Hac86]. Dies erfordert, die Messungen und die sich daran anschließende Verarbeitung der Meßergebnisse effizient zu gestalten. Durch die Nutzung einer minimalen aber aussagekräftigen Lastinformationsmenge für die Lastbalancierung kann der zusätzliche Overhead durch das Messen und das Verwalten der Lastinformationen gering gehalten werden [Cici92]. Die Länge der Lastmeßperioden muß so gewählt sein, daß effektive Messungen möglich sind, die eine den Aktualitätsanforderungen angemessene aktuelle Belastung widerspiegelt. Dabei muß beachtet werden, daß Entscheidungen bzgl. der Lastbalancierung, die auf unaktuellen Zustandsinformationen aufbauen, sich negativ auf die Systemleistung auswirken können. Um den durch die Messungen verursachten

Last-parameter	**Verfahren**						
	[Zhou93]	[Zhou86]	[Hac86]	[Cici92]	[Bono89]	[Baum89]	[Osse92]
CPU-Run queue-Länge	*	*	*	*	*	*	*
freie Speicherseiten	*	*	-	-	-	-	-
CPU-Auslastung	-	-	*	-	-	-	*
Anzahl von Datensperren zur Vermeidung v. Zugriffskonflikten	-	-	-	*	-	-	-
Anzahl der aktiven Prozesse	-	-	*	-	*	*	*
Platten E/A	*	-	-	-	-	-	-
Paging Rate	-	-	-	-	-	*	-
Paging u. swapping Rate	-	*	-	-	-	-	-
Verfügbarer swap-Bereich	*	-	-	-	-	-	-
Anzahl der laufenden Transaktionen	-	-	-	*	-	-	-
Anzahl konkurrenter Nutzer	*	-	-	-	-	-	-

Overhead zu minimieren, können im Rahmen der Lastbalancierung die im UNIX-Kern gesammelten und gepufferten CPU-Warteschlangenlängen genutzt werden, die dann in längeren Zeitintervallen (z.B. 1min) aus dem Kern ausgelesen werden [Zhou86].

Im Zusammenhang mit der weiteren Nutzung der während der Lastmessung gewonnenen Informationen kann eine Reduzierung bzw. Verdichtung der Datenmenge vorgenommen werden, um einfach zu interpretierende Aussagen über die Belastung einzelner Ressourcen zu erhalten. Außerdem kann eine Vereinigung einzelner Größen zu einem einzigen aussagekräftigen Parameter angestrebt werden. Die Aufbereitung der Meßdaten erfordert einerseits CPU-Zeit und stellt einen zusätzlichen Aufwand dar. Andererseits werden aber aussagekräftige Informationen zum Systemverhalten bzw. zu

Systemressourcen für die weiteren Entscheidungen bzgl. der Lastbalancierung bereitgestellt [Ferr83]. Einzelne Parameter können bei der Zusammenfassung entsprechend ihrer Bedeutung gewichtet werden. Dies bedeutet, daß der Einfluß bestimmter Parameter entweder verstärkt oder abgeschwächt wird. Der CPU-Belastungswert kann in Abhängigkeit von der Prozessortaktrate normalisiert werden [Zhou93, Bono89].

mögliche Normalisierungsformel:

$$normalisierte\ CPU-Belastung = \frac{CPU-Run\ queue-Länge}{Prozessortaktrate}$$

Der normalisierte Lastwert liefert eine Basis, um die Belastungswerte unterschiedlich schneller Prozessoren vergleichen zu können.

In [Bara93] erfolgt eine Normalisierung des lokalen Lastwertes, indem dieser Wert durch die relative Taktrate des Prozessors dividiert wird. Die relative Taktrate ist ein Verhältnis zwischen der Taktrate des schnellsten Prozessors im System und der Taktrate des speziell betrachteten Prozessors.

Normalisierungsformel nach [Bara93]:

$$normalisierter\ Lastwert = \frac{Load_T * CPU_{\max}}{CPU_{SP}}$$

verwendete Parameter:

$Load_T$: lokaler Belastungswert

$CPU_{\max}$: Taktrate des schnellsten Prozessors im System

CPU_{SP} : Taktrate des lokal betrachteten Prozessors

Eine einheitliche Betrachtung der Parameter und Maße bei unterschiedlichen Konfigurationen in einem heterogenen System ermöglicht einen schnellen Vergleich sowie eine effektive Nutzung der Zustandsgrößen sowohl durch den Anwender als auch durch die Systemroutinen bzw. den Lastbalancierungsalgorithmus [Ludw93].

Um die Informationen über die Belastung des Rechners von den numerischen Werten zu abstrahieren, kann die Last in drei Zustände (leicht, normal, schwer) unterteilt werden [Ni85]. Der Untergliederung liegt folgende Lastcharakterisierung zu Grunde:

leicht belastet: Rechner kann einige übertragene Prozesse akzeptieren und bearbeiten

normal belastet: keine Prozeßtransferbemühungen des Rechners erforderlich

schwer belastet: Rechnerknoten kann Prozesse zu anderen Rechner übertragen, um die lokale Belastungssituation zu entspannen

Eine Zuordnung der Rechnerlast in diese Zustandsgruppen erleichtert die Entscheidungen bei Laständerungen. Erst wenn die Belastung des Rechners sich so stark ändert, daß ein Zustandswechsel (z.B. leicht $\rightarrow$ normal) erfolgt, können Reaktionen notwendig sein.

4.2. Informationsmanagement

4.2.1. Austausch von Lastinformation

In Abhängigkeit vom Lastbalancierungsverfahren variiert der Komplexitätsgrad und die Anzahl von Belastungsinformationen, die jeder lokale Rechnerknoten in die zu treffenden Lastbalancierungsentscheidungen einbezieht. Der Komplexitätsgrad an Informationen, die bei diesen Entscheidungen genutzt werden, reicht von der Verwendung nur lokaler Belastungsinformationen bis hin zur Berücksichtigung der Kenntnisse bzgl. der

Last von allen Rechnerknoten des Systems (siehe Abs. 3.1.5.). Um die Informationsgrundlage für die Lastverteilungsentscheidungen des lokalen Rechners zu erweitern, erfolgt ein Austausch von Informationen, die während der Lastmessungen gewonnen wurden, zwischen den Rechnerknoten. Durch die Informationsübertragung wird die Möglichkeit und Basis geschaffen, daß der Lastbalancierungsalgorithmus über eine globale und aktuelle Beschreibung der Belastungsverhältnisse im System verfügt. Die Nachrichtenversendung zum Zweck der Lastinformationsübertragung erfolgt häufig mittels Broadcast (siehe Abs. 5.4.1.).

Im Zusammenhang mit dem Informationsaustausch ist ein Kompromiß zwischen den umfassenden Kenntnissen über die Belastung der Rechner im verteilten System und dem durch die Übertragung verursachten Overhead zu finden [Livn82]. Daher gilt es zu klären, welchen Systemüberblick die Lastkomponente des lokalen Rechners besitzen sollte, bzw. von wieviel anderen Rechnerknoten die Lastinformationen berücksichtigt werden sollten [Ludw93]. Je größer die vorhandene Menge an detaillierten Belastungsinformationen von Nachbarknoten ist, um so fundierter ist die Informationsgrundlage bei der Entscheidung bzgl. der Lastverteilung. Dies kann letztendlich in einer verbesserten Leistung des Gesamtsystems resultieren. Auf der anderen Seite ist für eine umfangreiche Informationssammlung und -verwaltung ein größerer Aufwand erforderlich. Je mehr Informationen ausgetauscht werden, um so größer sind die erforderlichen Kommunikationskosten [Wang85]. Da jede Informationsübertragung einen zusätzlichen Aufwand und eine Belastung darstellt, gilt es abzuwägen, wann ein Lastinformationsaustausch eines lokalen Rechners notwendig ist, um die anderen Rechner auf einem angemessenen aktuellen Informationsstand bzgl. der Belastung des lokalen Rechners zu halten.

4.2.2. Verringerung des Overhead beim Lastinformationsaustausch

Der durch den Lastinformationsaustausch verursachte Overhead kann verringert werden bzw. auf einem niedrigen Niveau gehalten werden, wenn Einschränkungen hinsichtlich des Umfangs der am Lastinformationsaustausch beteiligten Rechner getroffen werden. Am Informationsaustausch eines lokalen Rechners haben dann nicht alle Rechnerknoten des gesamten Systems Anteil. In [Zhou93] werden die Rechnerknoten des Gesamtsystems in Gruppen unterteilt. Bei dieser Untergliederung wird sich an der Struktur

des verteilten Systems orientiert. Der Umfang des Lastinformationsaustausches eines Rechnerknotens beschränkt sich dann auf den Rahmen der jeweiligen Gruppe, zu der der Rechner gehört. Der lokale Rechner braucht somit nicht die Zustandsinformationen des Gesamtsystems zu berücksichtigen und zu verwalten. Gleichzeitig wird dadurch die Menge an Belastungsinformationen verringert, die in die Lastbalancierungsentscheidungen des lokalen Rechners einfließen [Lin92].

Eine anderer Ansatzpunkt, um den durch den Informationsaustausch verursachten Aufwand zu verringern, besteht in der Reduzierung der Häufigkeit von Lastinformationsversendungen. Dazu können Grenzen festgelegt werden, die bestimmen, ab wann der Lastaustausch zwischen den Rechnerknoten erforderlich ist. Die Auswirkungen geringer Lastschwankungen eines Rechners auf die Leistung des verteilten Systems werden in [Ludw93] als nicht so bedeutend angesehen, daß darauf im Rahmen der Lastbalancierung effizient reagiert werden kann. Daher ist es nicht erforderlich, über jede Belastungsänderung eine Nachricht an die anderen Rechnerknoten zu versenden [Lin92]. Häufig werden nur signifikante bzw. markante Änderungen in der Last des lokalen Rechners zum Anlaß für einen Lastinformationsaustausch genommen [Zhou93, Zhou87, Shoj91]. Als markant werden solche Belastungswechsel bezeichnet, die eine Reaktion des Lastbalancierungsalgorithmus bzw. einen Prozeßtransfer erfordern, um der veränderten Lastsituation Rechnung zu tragen [Ni85].

In [Shoj91] wird ein Lastwechsel als signifikant angesehen, wenn sich die aktuelle Last des Rechners so stark geändert hat, daß die Differenz zwischen dem vorletzten und dem zuletzt ermittelten Lastwert einen festgelegten Schwellwert überschreitet. Bei der Fixierung des Schwellwertes sind die Kosten und der Nutzen jeder zusätzlichen Informationsübertragung gegeneinander abzuwägen. Als Anlaß für eine Belastungsinformationsversendung kann auch das Eintreten des lokalen Rechnerknotens in den idle (= untätig) Zustand genutzt werden [Lin92].

Bei der Anwendung von drei Zuständen (leicht, normal, schwer) zur Charakterisierung der Rechnerlast kann ebenfalls mittels Einschränkungen eine Minimierung der Anzahl von Informationsaustauschen über das Netzwerk erreicht werden [Ni85]. Wechselt die Last eines Rechners häufig zwischen den Zuständen "normal" und "leicht" (normal $\rightarrow$ leicht, leicht $\rightarrow$ normal), erfolgt eine Reaktion der Lastkomponente und somit eine Versendung von Informationen über einen Lastwechsel vom Zustand "normal" zum Zu-

stand "leicht" nur dann, wenn der Rechnerknoten vor dem Lastzustand "normal" schwerbelastet war (Zustand "schwer").

In [Eage86b] wird die Anzahl von Lastinformationsversendungen durch folgende Herangehensweise gering gehalten. Erst wenn ein lokaler Rechner auf Grund der lokalen Lastinformationen seinen Überlastungszustand erkennt, erbittet er von den anderen Rechnerknoten des Systems deren Lastinformationen. Aus den übersendeten Antworten wählt der überlastete Rechner dann einen Zielknoten für den Prozeßtransfer aus.

In [Zhou93] und [Zhou87] werden Algorithmen aufgezeigt, bei denen die Versendung von Lastinformationen in periodischen Zeitabständen erfolgt. Die Größe der Lastaustauschperiode hat einen Einfluß auf die Effizienz des Lastbalancierungsalgorithmus. Eine kurze Periode verursacht einen häufigen Informationsaustausch und somit einen nicht geringen Overhead. Hingegen sind die Lastinformationen sehr aktuell, auf deren Basis dann wirksame Lastverteilungsentscheidungen getroffen werden können. Wird die Lastaustauschperiode zu groß gewählt, können unaktuelle Informationen in die Entscheidungsfindung einfließen. Mögliche Folgen sind, daß zu viele Prozesse während einer langen Periode zu einem leichtbelasteten Rechner transferiert werden, so daß dieser Zielrechner daraufhin selbst überlastet wird. Der geänderte Belastungszustand des Zielrechners wird zu spät erkannt und an die anderen Rechner verbreitet. Auf Grund der Kommunikationsverzögerung bei der Übertragung der Lastinformationen zwischen den Rechnern können Einbußen hinsichtlich der Informationsaktualität eintreten [Ni85].

4.2.3. Verwaltung der Lastinformationen

Neben dem Austausch von Lastinformationen ist auch die Sammlung und Verwaltung der Lastinformationen notwendig. Die bei den Messungen gewonnenen Informationen sind so zu halten, daß sie ohne größeren Aufwand bei den weiteren Operationen des Lastbalancierungsalgorithmus verfügbar sind.

Abhängig von dem angewendeten Lastausgleichsverfahren (zentral oder dezentral - siehe Abs. 3.1.3.) werden die Belastungsinformationen der Rechner entweder an einem Ort (zentral) oder von jedem Rechnerknoten selbst (dezentral) verwaltet und gesammelt. Ein zentraler Rechner kann auf der Basis der ihm verfügbaren globalen Belastungsin-

formationen alle im System anfallenden Lastverteilungsentscheidungen treffen oder nur die Verteilung der bei ihm zusammenlaufenden Lastinformationen des gesamten Systems vornehmen. Bei einer zentralen Informationsverwaltung sendet die Lastkomponente des lokalen Rechners die Informationen über die lokale Belastung an den zentralen Rechner. Im Fall der dezentralen Informationssammlung verschickt die lokale Lastkomponente die Belastungsinformationen an alle Rechnerknoten, die am Lastinformationsaustausch des lokalen Rechners beteiligt sind.

Häufig werden die Lastinformationen der Rechnerknoten in Form eines Lastvektors zentral oder dezentral gesammelt [Zhou88, Zhou87, Ni85]. Die im Lastvektor enthaltenen Informationen bilden die Basis für die zu treffenden Lastbalancierungsentscheidungen.

In [Shoj91] wird ein Lastvektor, der als Liste von Lastwerten organisiert ist, in folgender Art genutzt: Ist der Anlaß für einen erforderlichen Lastinformationsaustausch auf einem lokalen Rechnerknoten gegeben, so erfolgt die Versendung der Belastungsinformationen des lokalen Rechners an die anderen am Informationsaustausch beteiligten Rechner des Systems. Empfängt ein Rechner die Lastinformationen eines Nachbarknotens, so erfolgt entweder eine Einordnung der Informationen in die Liste oder eine Aktualisierung des Rechnerknoteneintrages in der Liste. Indem zusammen mit der Belastungsinformation eine Rechnerknotenidentifizierung abgelegt wird, ist eine eindeutige Zuordnung der Lastinformationen zu einem Rechner erzielt. Aus dem Lastvektoreintrag kann entnommen werden, wie stark der entsprechende Rechnerknoten belastet ist. Die Liste ist sortiert nach aufsteigenden Lastwerten. Der erste Listeneintrag repräsentiert den am geringsten belasteten Rechner. Wird ein gering belasteter Rechnerknoten als Ziel für einen Prozeßtransfer gesucht, so wird der erste Eintrag in der Liste dafür genutzt. Nachdem auf diese Weise der Zielrechner ausgewählt wurde, wird der erste Eintrag aus der Liste entfernt. Eine derartige Verwendung einer Liste im Rahmen der Lastbalancierung bietet folgende Vorteile:

- Der am geringsten belastete Rechnerknoten als Ziel eines Prozeßtransfers kann schnell und mit geringem Aufwand ausgewählt werden.
- Die Auswahl des Zielrechners für die Verlagerung eines Prozesses von einem überlasteten zu einem gering belasteten Rechnerknoten erfolgt in einer

gleichmäßigen Vorgehensweise.

- Da nach der Prozeßübertragung zu dem am geringsten belasteten Rechner der Eintrag des Zielrechners aus der Liste entfernt wird, kann eine Überlastung dieses Rechners durch mehrere Prozeßverlagerungen verhindert werden.

Eine andere Herangehensweise bei der Nutzung eines Lastvektors zur Verwaltung der Lastinformationen wird in [Bara85] dargestellt. Jeder Rechnerknoten verwaltet einen Lastvektor L der Größe v (= Anzahl der Rechnerknoteneinträge). Der erste Eintrag des Lastvektors enthält den Lastwert des lokalen Rechners. Die restlichen (v - 1) Einträge beinhalten die Lastinformationen einer Untermenge von Rechnerknoten des Systems. Bereits die Festlegung des Wertes v hat einen Einfluß auf die Leistung des Lastbalancierungsalgorithmus. Der Lastvektor muß auf der einen Seite groß genug sein, um ausreichend viele aktuelle Belastungsinformationen von anderen Rechnern dem lokalen Rechnerknoten für die Prozeßzuteilungsentscheidung zur Verfügung zu stellen. Andererseits muß der Lastvektor klein genug sein, um einen unnötig großen Overhead, der durch die Sammlung und Verwaltung von Lastinformationen verursacht wird, zu verhindern. Der Lastvektor wird periodisch nach einem Zeitintervall t aktualisiert. Ebenfalls von Bedeutung für die Leistung des Verteilungsalgorithmus ist die Wahl dieser Zeitperiode t. Im Rahmen der Lastvektoraktualisierung werden vom lokalen Rechner folgende Operationen realisiert:

- Aktualisierung des eigenen Lastwertes im Lastvektor
- zufällige Auswahl eines Rechnerknotens k
- Versenden der ersten Hälfte des lokalen Lastvektors an den Rechner k

Empfängt ein Rechnerknoten die Hälfte des Lastvektors (= L_R) von einem anderen Rechner, so werden diese Informationen mit dem Inhalt des lokalen Lastvektors unter Anwendung folgender Regeln vermischt:

- $L[i] \rightarrow L[2i]$; $1 < i < v/2-1$
- $L_R[i] \rightarrow L[2i+1]$; $0 < i < v/2-1$

Durch diese Art von Lastinformationsaustausch ändert sich nach jeder Vermengung des lokalen Lastvektors mit dem empfangenen Lastvektorteil (L_R) die Zusammensetzung der Menge von Rechnerknoten, deren Belastung dem lokalen Rechner bekannt sind.

Außerdem ist nicht ausgeschlossen, daß im Lastvektor mehrere Einträge über die Belastung eines einzelnen Rechners enthalten sind.

Einen anderen Ansatzpunkt zur Verwaltung der Belastungsinformationen der Rechner ist eine gemeinsam genutzte Datei (shared File), auf die alle Rechnerknoten Zugriff haben [Osse92]. In dieser Datei sind die Belastungszustände aller Rechnerknoten abgelegt. Wechselt der Lastzustand eines Rechnerknotens signifikant, so wird auf die gemeinsam genutzte Datei vom lokalen Rechner zugegriffen und der entsprechende Rechnerknoteneintrag aktualisiert. Durch den eintragenden Rechner wird die Datei solange blockiert, bis der Änderungseintrag abgeschlossen ist. Auf diese Weise werden konkurrente Zugriffsprobleme verhindert. Wird ein Zielrechner für einen Prozeßtransfer von einem lokalen Rechner gesucht, so erfolgt zur Auswahl des Zielknotens ein Zugriff auf die von allen Rechnern gemeinsam genutzte Datei.

4.3. Prozeßtransfer

Nach der Lastbestimmung und dem Austausch von Lastinformationen zwischen den Rechnerknoten verfügt ein lokaler Rechner über Belastungsinformationen, die die Basis für die weiteren Operationen im Rahmen der Lastbalancierung bilden. Im Zusammenhang mit den Transferstrategien zur Übertragung eines Prozesses sind folgende Fragen zu klären:

- Unter welchen Bedingungen ist eine Prozeßverlagerung von einem Rechner zu einem anderen erforderlich?
- Welcher Prozeß ist für eine Übertragung geeignet?

Der eigentliche Prozeßtransfer zwischen unterschiedlich stark belasteten Rechnerknoten bleibt für den Nutzer im verteilten System transparent.

4.3.1. Anlaß für eine Prozeßübertragung

Mit dem Prozeßtransfer zwischen unterschiedlich stark belasteten Rechnerknoten wird die Balancierung der Last in einem verteilten System verwirklicht. Daher werden die Entscheidungen einer lokalen Lastkomponente, ob ein Prozeß lokal oder entfernt ausgeführt werden soll (= Prozeßverlagerungsentscheidung), in Abhängigkeit vom Belastungszustand des lokalen Rechners getroffen. Grundlage für diese Entscheidung sind

die Kenntnisse über die Belastung des lokalen Rechnerknotens, die während der Lastmessung gewonnen wurden [Livn82].

Eine einfache und effektive Vorgehensweise um festzustellen, wann eine Prozeßübertragung notwendig ist, basiert auf einem statischen Schwellwert. Dieser Wert bezieht sich auf die Belastung des Rechnerknotens. Ist der Anlaß für einen Prozeßtransfer ein schwer belasteter Rechner, wird durch den Schwellwert festgelegt, wann die Last des Rechners als schwer eingestuft wird. Solange die Last des lokalen Rechnerknotens unterhalb des Schwellwertes liegt, ist der Rechner noch nicht schwer belastet und kann Prozesse lokal ausführen. Überschreitet die Last eines Rechnerknotens den Schwellwert, d.h. der Rechner ist schwer belastet, sollten Prozesse von dem lokalen überlasteten Rechner (Quellrechner) auf einen gering belasteten Rechner (Zielrechner) zur Ausführung verlagert werden bzw. ein Prozeßtransfer angestrebt werden [Eage86b].

Der Festlegung des Schwellwertes kommt große Bedeutung zu. Der Schwellwert ist auf einem solchen Niveau zu fixieren, daß die Leistung des Rechners nach dem Überschreiten des Lastschwellwertes schnell ohne eine Balancierung der Last sinken würde [Harg90]. Ist der Schwellwert zu hoch angesetzt, verbleiben die Rechner zu lange im schwer belasteten Zustand bevor die Last den Schwellwert überschreitet und Prozeßverlagerungen realisiert werden können. Das Potential der Lastbalancierung zur Leistungsverbesserung wird nicht ausreichend genutzt [Zhou88]. Ein zu niedriger Schwellwert hat zur Folge, daß zu viele Rechner schnell als schwer belastet gelten. Resultat dessen ist, daß viele unwirtschaftliche Prozeßübertragungen unternommen werden, obwohl eine lokale Verarbeitung des Prozesses möglich und effizienter gewesen wäre [Hac86]. Außerdem ist zu berücksichtigen, daß durch jeden Prozeßtransfer ein zusätzlicher Overhead verursacht wird.

Die Rechner des Systems werden bei [Mirc89] in mehrere Rechnerklassen in Abhängigkeit von der Verarbeitungstaktrate und der externen Jobankunftsrate der einzelnen Rechner unterteilt. Den Klassen von Rechnerknoten wird jeweils ein fixierter Lastschwellwert zugeordnet. Überschreitet die Belastung des Rechners den Schwellwert der zugehörigen Rechnerklasse, so wird ein Prozeßtransfer angestrebt. Der Schwellwert sollte in Abhängigkeit von der Systemlast festgelegt werden. Ein dynamisch fixierter Schwellwert wird entsprechend an das im System bestehende Lastniveau angepaßt. Bei einer geringen Last im gesamten System wird ein niedriger Schwellwert verwendet

[Eage86b]. Dadurch können trotz eines geringen Lastniveaus im System Prozeßverlagerungen zum Ausgleich bestehender Lastdifferenzen zwischen den Rechnern realisiert werden.

In [Lin92] werden empfängerinitiierte Transferstrategien (siehe Abs. 3.1.4.) genutzt. Dabei wird nicht der schwer belastete sondern der unterbelastete Rechner zum Anlaß für eine Prozeßübertragung genommen. Ein schwach belasteter Rechnerknoten strebt danach, Prozesse von anderen stark belasteten Rechnern zu übernehmen, um dadurch die Last zwischen den Rechnern des Systems auszugleichen. Der Schwellwert legt fest, ab wann ein Rechner schwach belastet ist und somit für die Übernahme von Prozessen bereit ist. Sinkt die Last des lokalen Rechnerknotens unterhalb des Schwellwerts, kann von diesem Rechner die Initiative zum Prozeßtransfer ausgehen. Im extremsten Fall wird der Schwellwert auf dem Niveau festgelegt, daß der Rechner untätig sein muß, bevor eine Prozeßübernahme angestrebt werden kann.

Die Entscheidung, ob eine Prozeßübertragung notwendig ist, basierte in den bisher dargestellten Transferstrategien nur in Abhängigkeit von der Belastung des lokalen Rechnerknotens. [Krue84] und [Stan84] berücksichtigen zusätzlich die Last anderer Rechnerknoten. Anlaß für einen Prozeßtransfer sind Lastunausgeglichenheiten bzw. Lastdifferenzen im System, d.h. die Belastungen einzelner Rechner variieren voneinander. Um Lastunterschiede festzustellen, werden Differenzen zwischen der Last des lokalen Rechnerknotens und der Last einzelner anderer Rechner bzw. dem Durchschnittslastwert des Gesamtsystems (Durchschnitt der Lastwerte aller Rechner im System) gebildet. Überschreitet die Differenz eine festgelegte Akzeptanzgrenze (z.B. Schwellwert), so wird eine Prozeßübertragung angestrebt, um die Belastungsunterschiede auszugleichen. Der Betrag, um den sich die Rechnerbelastungen untereinander bzw. vom Durchschnittslastwert unterscheiden dürfen, bevor eine Lastverteilung realisiert wird, muß sorgfältig festgelegt werden. Die Differenz darf auf der einen Seite nicht so klein sein, daß die Rechner die meiste Zeit mit dem Prozeßtransfer beschäftigt sind, um einen geringfügigen Lastunterschied auszugleichen bzw. um die eigene Last dicht am Durchschnittslastwert beizubehalten. Andererseits darf der Betrag nicht so groß sein, daß viele mögliche Prozeßübertragungen nicht genutzt werden [Krue84].

Indem bereits in die Prozeßverlagerungsentscheidung des schwer (oder gering) belasteten Rechnerknotens neben dem lokalen Lastwert auch die Lastzustände anderer

Rechner einbezogen werden, erhöht sich die Wahrscheinlichkeit, einen akzeptablen Zielknoten für den zu transferierenden Prozeß zu finden.

4.3.2. Auswahl des zu übertragenden Prozesses

Soll eine Lastverteilung realisiert werden, muß entschieden werden, welcher Prozeß bzw. welche Prozesse verlagert werden können. Häufig wird für die Übertragung der zuletzt eingetretene bzw. der neu angekommende Prozeß ausgewählt [Eage86b, Mirc89]. Dieses Prozeßauswahlkriterium wird dort angewendet, wo ein Schwellwert zur Ermittlung der Notwendigkeit eines Prozeßtransfers bei der Transferstrategie genutzt wird. Der Prozeß, der das Überschreiten des Schwellwertes verursacht, wird zur Übertragung ausgewählt. Bei dieser Vorgehensweise können in einem System mit einem hohen Durchschnittslastwert Instabilitäten auftreten. Der zur Übertragung ausgewählte Prozeß wird fortwährend zwischen den insgesamt schwer belasteten Rechnern transferiert, da der Prozeß bei den Zielrechnern immer den Lastschwellwert überschreitet. Um ein solches Prozeß-Trashing zu vermeiden, kann der Zeitraum begrenzt werden, in dem ein Prozeß übertragbar ist [Ni85].

Mit der Verlagerung des Prozesses auf einen entfernten Rechner soll einerseits der lokale Rechner entlastet werden und andererseits die Antwortzeit des Prozesses verringert werden. Um eine Transferentscheidung unter dem Gesichtspunkt der Prozeßantwortzeitverkürzung zu treffen, ist sowohl der erforderliche Aufwand für die Prozeßausführung abzuschätzen als auch das Leistungsvermögen des Zielknotens zu berücksichtigen. Bei der Auswahl eines für die Verlagerung geeigneten Prozesses sind Kenntnisse über die Prozesse einzubeziehen. Prozeßbezogene Daten umfassen z.B. Anforderungen des Prozesses bzgl. Ressourcen (wie CPU- und Speicherbedarf) und bzgl. der erforderlichen Kommunikation mit anderen Prozessen im System [Ludw93]. Grundsätzlich sind bei der Übertragung eines Prozesses die Prozeßkontextinformationen mit zu transferieren. Eine Prozeßverlagerung ist nur dann auszuführen, wenn der Prozeß bei der entfernten Ausführung keine Leistungseinbußen erfährt. Gleichzeitig muß neben der wahrscheinlichen Verarbeitungszeit des Prozesses auf dem entfernten Rechnerknoten auch die erforderliche Zeit für den Prozeßtransfer und den Empfang der Verarbeitungsergebnisse berücksichtigt werden.

Bezüglich der Wahl eines für die Übertragung geeigneten Prozesses ist zu beachten, daß Prozesse in mobile und unmobile [Zhou88] unterteilt werden können:

- *Mobile Prozesse* können auf einem beliebigen Rechner des verteilten Systems verarbeitet werden. Der Ausführungsort (lokal oder entfernt) hat keinen Einfluß auf die Ergebnisse.
- *Unmobile Prozesse* sind für eine entfernte Bearbeitung hingegen nicht geeignet, da solche Prozesse z.B. lokale Dienste realisieren, lokale Ressourcen zur Bearbeitung benötigen bzw. auf physische Geräte des lokalen Rechners zugreifen oder interaktive Operationen beinhalten.

In [Zhou87] wird festgestellt, daß weniger als die Hälfte aller Prozesse für eine entfernte Ausführung geeignet ist. Eine Leistungsverbesserung mittels Lastbalancierung ist auch noch dann erzielbar, wenn ein bedeutender Teil der Prozesse (bis zu 70% aller Prozesse) unmobil ist. Ein Lastbalancierungsalgorithmus arbeitet noch effizient, wenn nur ein Teil der für den Transfer geeigneten Prozesse übertragen wird [Zhou88].

Hinsichtlich des Zeitpunktes, wann ein Prozeß in Bezug auf seine Verarbeitung übertragen werden kann, gibt es zwei Möglichkeiten [Osse92].

- Der Prozeßtransfer erfolgt vor Beginn der Prozeßbearbeitung.
- Die Übertragung eines Prozesses findet nach begonnener Prozeßbearbeitung statt.

Ist der Prozeßtransfer nur vor Bearbeitungsbeginn möglich, muß ein Prozeß bis zum Verarbeitungsende auf dem Rechnerknoten verbleiben, auf dem die Ausführung begonnen wurde. Bei großen Prozessen (lange Bearbeitungszeit) verursacht dies Einbußen hinsichtlich der Lastverteilung. Der Prozeß, dessen Ausführung begonnen hat, ist trotz Veränderung des Belastungszustands des bearbeitenden Rechners an diesen Bearbeitungsplatz gebunden. Eine Prozeßübertragung, die auch nach dem Verarbeitungsbeginn erfolgen kann, ermöglicht hingegen Reaktionen auf den aktuellen Belastungszustand des ausführenden Rechnerknotens. Bei der Übertragung eines Prozesses, dessen Bearbeitung begonnen hat, sind folgende Operationen zu realisieren:

- Stoppen der Prozeßausführung
- Fixieren des aktuellen Bearbeitungszustands

- Übertragung des Prozesses sowie der erforderlichen Kontextinformationen zum Zielrechner
- Fortsetzung der Ausführung auf dem Zielrechner an der Stelle, wo die Bearbeitung auf dem Quellknoten gestoppt wurde

Zu beachten ist, daß neben dem Prozeß auch die Informationen über die bereits erfolgte Bearbeitung des Prozesses zu übertragen sind.

Die Auswahl der für die entfernte Ausführung geeigneten Prozesse kann explizit durch den Nutzer vorgenommen werden. Der Nutzer spezifiziert selbst die übertragbaren und die nicht übertragbaren Prozesse (unmobile Prozesse) in entsprechenden Listen [Osse92, Zhou87]. Ob ein Prozeß entfernt bearbeitet werden kann, wird überprüft, indem die vom Nutzer generierte Liste der übertragbaren Prozesse nach einem speziellen Eintrag zum Prozeß durchsucht wird. In dieser Liste kann gleichzeitig zu jedem Prozeß eine Aufstellung der für die Ausführung des Prozesses erforderlichen Ressourcen abgelegt sein. Die Liste ist durch den Nutzer regelmäßig zu aktualisieren.

4.4. Kooperation und Lokation

Nachdem eine Entscheidung bzgl. einer Prozeßübertragung und damit für die entfernte Ausführung eines Prozesses getroffen wurde, muß ein Zielrechner für den Transfer und die anschließende Prozeßverarbeitung lokalisiert werden. Zur Ermittlung eines geeigneten Zielknotens für die Prozeßausführung gibt es verschiedene Kooperationsstrategien der Rechner. Die Unterschiede zwischen den Vorgehensweisen ergeben sich aus den unterschiedlichen Informationsmengen, die bei der Wahl des Zielknotens berücksichtigt werden und aus verschiedenen Kooperationsintensitäten zwischen den Rechnern. [Harg90] trifft eine Hauptunterteilung der Strategien in sender- und empfängerinitiierte, wobei entscheidend ist, ob von einem überlasteten oder von einem unterbelasteten Rechner die Initiative zur Balancierung der Last und somit zur Suche nach einem Zielknoten für den Prozeßtransfer ausgeht (siehe Abs. 3.1.4.).

Im weiteren sollen einige Beispiele von Kooperations- und Lokationsstrategien dargestellt werden.

4.4.1. Grundlegende Herangehensweisen bei der Zielrechnersuche

Viele Algorithmen in der Literatur [Eage86b, Wang85] treffen die Transferentscheidungen nur auf der Grundlage von lokalen Lastinformationen. Der Anlaß für die Übertragung eines Prozesses ist das Überschreiten eines statischen Schwellwertes (senderinitiiertes Vorgehen).

Der Zielknoten für einen Prozeßtransfer von einem schwer belasteten Rechner wird bei den einfachsten Strategien per Zufall ausgewählt. Nach erfolgter Wahl wird der Prozeß dorthin übertragen. Der Zielrechner behandelt den ankommenden Prozeß wie einen lokalen. Da das Ziel für die Übertragungsentscheidung zufällig ausgewählt wird, ist der Fall nicht ausgeschlossen, daß der Zielrechner bereits stark belastet ist. Wird der Schwellwert bzgl. der Belastung des Zielrechners durch den ankommenden Prozeß nicht überschritten, so wird der Prozeß zur Bearbeitung auf dem Zielknoten akzeptiert. Verursachte hingegen der übertragene Prozeß das Überschreiten des Lastschwellwertes, wird vom Zielknoten wiederum ein Zielrechner für eine erneute Prozeßübertragung per Zufall ausgewählt. Der Vorgang der Zielrechnersuche wird solange wiederholt, bis entweder ein Rechner gefunden wurde, der den Prozeß zur Bearbeitung akzeptiert oder bis ein Probelimit erreicht ist. Dieses Limit legt fest, wie oft ein Prozeß maximal bei der Suche nach einem Zielrechner übertragen werden kann. Der Rechner, der das Überschreiten des Probelimits verursacht, muß den Prozeß unabhängig von seinem aktuellen Belastungszustand lokal verarbeiten [Shen88].

Ein anderer Algorithmus [Eage86a] berücksichtigt vor dem eigentlichen Prozeßtransfer den Belastungszustand des potentiellen Zielrechners. Die Auswahl des Zielknotens erfolgt wieder per Zufall. Bevor der Prozeß übertragen wird, findet eine Überprüfung statt, ob der beabsichtigte Prozeßtransfer auf dem gewählten Rechner ein Überschreiten des Lastschwellwertes verursachen würde. Die Kontrolle erfolgt, indem entweder die aktuelle Last beim potentiellen Zielrechner mittels einer Probenachricht abgefragt wird oder indem die Informationen über den Lastzustand des möglichen Zielknotens aus dem Lastvektor des lokalen Rechners berücksichtigt wird. Wird festgestellt, daß durch die Prozeßübertragung zum potentiellen Zielrechner kein Überschreiten dessen Lastschwellwertes verursacht werden würde, erfolgt der Prozeßtransfer. Andernfalls (Schwellwertüberschreitung) muß ein neuer Zielknoten ausgewählt werden. Dieser Auswahlvorgang wird wiederholt, bis ein akzeptabler Zielrechner gefunden wurde oder bis

die Anzahl der Versuche das Probelimit überschreitet. Im letzteren Fall muß der Prozeß lokal bearbeitet werden. Dadurch, daß die Belastung des potentiellen Zielrechners bei der Prozeßzuteilungsentscheidung berücksichtigt wird, können uneffiziente Prozeßtransfere vermieden werden [Eage86a].

Eine dritte Strategie trifft die Auswahl eines Zielrechners für die Prozeßübetragung in einer komplexeren Art und Weise. Neben der Überprüfung, ob der übertragene Prozeß auf dem möglichen Zielrechner die Belastung über die Grenze eines Schwellwertes erhöht, wird außerdem das beste Ziel für die Prozeßverlagerung gesucht. Dazu wird aus der Menge der möglichen Zielrechner der Rechner ausgewählt, der die kürzeste Warteschlangenlänge an aktiven Prozessen bzw. die geringste Belastung aufweist [Eage86b]. Die für die Auswahl notwendigen Informationen bzgl. der Rechnerbelastung können entweder aus dem Lastvektor des lokalen Rechners entnommen werden [Zhou88] oder müssen mittels Abfragen der potentiellen Zielrechner ermittelt werden [Osse92]. Die Größe des Suchraumes, aus dem der Zielrechner (Rechner mit der geringsten Last) ausgewählt wird, kann entweder die Gesamtmenge oder eine zufällig aussondierte Untermenge aller möglichen Rechner umfassen [Ferr85]. Zu dem so ermittelten Zielrechner erfolgt anschließend die Prozeßverlagerung.

Mit der Wahl des Probelimits wird ein Parameter festgelegt, der bestimmt, wie oft ein lokaler Rechner versuchen kann, einen Zielrechner für die Übertragung eines Prozesses zu finden bzw. wie oft ein Prozeß verlagert werden kann, ohne daß die Prozeßbearbeitung erfolgt. Erst wenn das Limit an Proben überschritten ist, muß eine lokale Ausführung des Prozesses erfolgen. Ohne eine solche Begrenzung könnte der Fall eintreten, daß alle Rechner Prozeßverlagerungen ausführen, um einen geeigneten Zielrechner zu ermitteln. Mit jeder zusätzlichen Probe erhöht sich zwar einerseits die Chance, einen geeigneten Zielrechner für die Bearbeitung eines zu übertragenden Prozesses zu finden. Auf der anderen Seite wird jedoch durch jeden Versuch ein zusätzlicher Aufwand verursacht, der sich negativ auf die Systemleistung auswirkt [Ferr85]. Die Größe des Probelimits hat somit Einfluß auf die Effizienz des Lastbalancierungsalgorithmus.

4.4.2. Weitere Kooperations- und Lokationsstrategien

Die beschriebenen Algorithmen verdeutlichen Möglichkeiten des grundlegenden Vorgehens bei der Zielrechnersuche für den im Zusammenhang mit der Lastbalancierung zu

realisierenden Prozeßtransfer. Auf diese Strategien bauen weitere Herangehensweisen auf, die in ihrer Komplexität variieren.

In [Osse92] werden bei der Wahl des Ausführungsortes für einen zu übertragenden Prozeß Informationen über die Leistungsfähigkeit des potentiellen Zielrechners wie z.B. Verarbeitungstaktrate des Prozessors (siehe Abs. 2.3.1.) einbezogen. Der Zielrechner wird unter Berücksichtigung folgender Kenntnisse ausgewählt:

- Welche Ressourcen sind für die Ausführung des zu verlagernden Prozeß notwendig?
- Welche Ressourcenanforderungen kann der potentielle Zielrechner befriedigen?

Durch Abwägung und Einbeziehung dieser Informationen kann der beste Zielrechner bei der Prozeßzuteilungsentscheidung gefunden werden [Zhou93].

Bei [Chow79] wird der Prozeß zu jenem Zielrechner verlagert, der die am geringsten vermutete Antwortzeit für den zu bearbeitenden Prozeß anbietet. Dazu wird der Rechnerknoten von allen *n* möglichen Rechnern gesucht, dessen Verhältnis zwischen Belastung und Leistungsvermögen minimal ist.

Auswahlkriterium für die Zielrechnersuche:

$$Zielrechner = \frac{Last_i}{Leistungsverm\ddot{o}gen_i} \rightarrow \min \qquad (i = 1, \ldots, n)$$

Aus diesem Verhältnis können die Verarbeitungsbedingungen, die ein Prozeß auf diesem Rechner erfährt, abgeschätzt werden.

Um die Wahl eines gering belasteten Rechnerknoten als möglichen Zielrechner für den Prozeßtransfer mehrerer Quellknoten und dadurch eine schnelle Überlastung dieses Zielrechners zu vermeiden, kann folgende Einschränkung getroffen werden: Der schwach belastete potentielle Zielrechner antwortet auf eine Probenachricht zwecks einer beabsichtigten Prozeßverlagerung eines überlasteten Rechners nur dann positiv, wenn der Zielknoten nicht bereits einem Prozeßtransfer eines anderen Quellknoten zugestimmt hat [Mirc89].

Zum Anlaß für einen notwendigen Prozeßtransfer werden neben der Über- bzw. Unterschreitung eines Lastschwellwertes auch Differenzen zwischen den Lasten der Rechnerknoten des Systems genommen (siehe Abs. 4.3.1.). [Stan84] entwickelte aufbauend auf den Unterschieden zwischen den Rechnerbelastungen drei Algorithmen. Der Lastinformationsaustausch erfolgt periodisch mittels Broadcast. Dadurch wird erreicht, daß jeder lokale Rechner über den Lastzustand der anderen Rechner des Systems Kenntnis besitzt. Die Belastungsinformationen werden in einem lokalen Lastvektor auf jedem Rechnerknoten verwaltet.

Als Zielknoten für den Prozeßtransfer wird beim 1. Algorithmus der am geringsten belastete Rechner aus dem Lastvektor ausgewählt. Bedingung für die Realisierung der Prozeßübertragung zu diesem Zielknoten ist, daß die Differenz zwischen der Last dieses Zielrechners und der Last des lokalen Rechners einen festgelegten Grenzwert überschreitet.

Beim 2. Algorithmus werden die Lastunterschiede mittels Differenzenbildung (wie beim 1. Algorithmus) zwischen dem lokalen Rechner und jedem anderen Rechnerknoten ermittelt, dessen Belastungsinformationen im lokalen Lastvektor verfügbar sind. Um die Differenzen zwischen der Last der möglichen Zielknoten und der Last des lokalen Rechners für alle im Lastvektor enthaltenden Rechnerbelastungswerte zu berechnen, wird der Lastvektor einmal durchlaufen. Die Lastunterschiede werden wie folgt berechnet (n = Anzahl der Rechnerknoten, deren Lastinformationen im Lastvektor vorhanden sind):

Formel zur Lastdifferenzenberechnung:

$$Lastdifferenz = Last\ des\ lokalen\ Rechners - Last\ des\ Rechners_i \qquad (i = 1, \ldots, n)$$

Überschreitet die Differenz einen festgelegten Wert $b1$, so wird zu diesem Zielrechner ein Prozeß übertragen. Ist die Differenz sogar größer als ein fixierter Wert $b2$, erfolgt der Transfer von zwei Prozessen. Ein statisches Limit begrenzt die Gesamtanzahl von Prozessen, die maximal beim einmaligen Durchlaufen des Lastvektors zur Differenzenbildung übertragen werden können. Dadurch werden Instabilitäten in Folge der Überlastung von Zielrechnern verhindert.

Der 3. Algorithmus arbeitet ähnlich wie der 1. Algorithmus. Zusätzlich wird die Information abgespeichert, zu welchem Zielrechner ein Prozeß vom lokalen Rechner aus hin übertragen wurde. Diese Kenntnisse werden genutzt, indem für die Dauer eines festgelegten Zeitintervalls jede weitere Prozeßverlagerung auf diesen Zielrechner verhindert wird, selbst wenn der Rechner bereits wieder gering belastet ist.

In [Krue84] bildet der globale Durchschnittslastwert über die Belastung aller Rechner des Systems eine Grundlage für die Zielrechnersuche. Wird ein Rechner überlastet (lokale Belastung überschreitet globalen Durchschnittslastwert), erfolgt eine Benachrichtigung der anderen Rechnerknoten über diesen Sachverhalt mittels Broadcast. Der Quellrechner sucht dann einen unterbelasteten Rechner als Ziel für den Prozeßtransfer. Dazu wartet der überlastete Rechner auf eine Antwortnachricht eines unterbelasteten Rechners, womit die Akzeptanz für einen Prozeßtransfer auf diesen Zielrechner signalisiert wird. Aus der Menge der antwortenden unterbelasteten Rechner wählt der übelastete Rechner einen Zielknoten aus. Ein unterbelasteter Rechner erhöht nach dem Versenden der Akzeptanznachricht seinen lokalen Lastwert in der Annahme, daß der Prozeß zu ihm übertragen wird. Dadurch wird erreicht, daß eine spätere Überlastung auf Grund weiterer Prozeßübertragungen zu diesem unterbelasteten Rechner verhindert werden kann. Nach Ablauf einer begrenzten Zeitperiode reduziert der gering belastete Rechner seinen Lastwert wieder, wenn während des Zeitintervalls keine Prozeßverlagerung auf den Rechner erfolgte. Diese Reaktion erfolgt in der Annahme, daß ein anderer Rechner als Ziel für den Prozeßtransfer vom Quellrechner ausgewählt wurde. Kann der überlastete Rechner keinen passenden Zielrechner für den zu übertragenden Prozeß finden (kein unterbelasteter Rechner antwortete), wird der globale Durchschnittslastwert erhöht. Die anderen Rechner werden anschließend über den neuen Durchschnittslastwert informiert.

In [Brya81] werden Paare von Rechnern gebildet, zwischen denen vor dem Prozeßtransfer eine Kommunikation zur Bestimmung des geringer belasteten Rechners und des zu übertragenden Prozesses erfolgt. Der Rechner $R1$ fordert einen Nachbarrechner $R2$ mittels Nachricht zur Paarbildung auf. Die an den Rechner $R2$ gesendete Nachricht enthält eine Zusammenstellung aller Prozesse von $R1$ mit deren wahrscheinlichen Rechenzeiten. Welche Möglichkeiten hat der Rechner $R2$, auf die Paarbildungsanfoderung von $R1$ zu reagieren?

- $R2$ reagiert mit einer Ablehnung der Anforderung falls der Rechner bereits mit einem anderen Rechnerknoten $R3$ ein Paar gebildet hat.
- Ist $R2$ bereits in einem Paar an einen anderen Rechner $R3$ gebunden, aber $R1$ geringer als $R2$ belastet, erfolgt nicht zwingend eine Ablehnung der Paarbildungsanforderung von $R1$. Die Anforderung wird von $R2$ zurückgestellt (nur eine Anforderung kann zurück gestellt werden). Nachdem das Paar zwischen $R2$ und $R3$ aufgelöst ist, wird erneut über die Anforderung von $R1$ entschieden.
- $R2$ akzeptiert die Paarbildung mit $R1$. Damit beide Rechner feststellen können, wer der stärker belastete (bzw. geringer belastete) von beiden ist, sendet $R2$ ebenfalls eine Liste aller Prozesse an $R1$. Der stärker belastete Rechner ermittelt die Prozesse, die auf den geringer belasteten Rechner verlagert werden könnten. Dazu wird eine Liste von übertragbaren Prozessen generiert. In der Liste sind alle die Prozesse des schwerbelasteten Rechners enthalten, die aus der entfernten Ausführung auf dem geringer belasteten Rechnerknoten einen Vorteil (z.B. Antwortzeitverkürzung) erzielen können. Diese Liste wird an den geringer belasteten Rechner gesendet. Danach erfolgt die Auflösung des Paares sowie die Realisierung des Prozeßtransfers zwischen dem schwer und dem gering belasteten Rechner. Zum Zeitpunkt der Prozeßübertragung können bereits neue Paare gebildet bzw. kann über eine zurückgestellte Paarbildungsanforderung entschieden werden.

5. Die Lastkomponente in HEAD

5.1. Einordnung der Aufgaben der Lastkomponente in das HEAD-Projekt

Die im Abschnitt 4 beschriebenen Aufgaben, die im Zusammenhang mit der Lastverteilung bzw. der Lastbalancierung auszuführen sind, werden von der Lastkomponente als ein logisches Objekt realisiert. In dieser logischen Komponente erfolgt eine Zusammenfassung aller zum Lastausgleich notwendigen Operationen. Die Aufgaben der Lastkomponente werden bezogen auf die funktionelle Architektur von HEAD (siehe Abb. 2) nicht von einer einzigen Komponente verwirklicht. Es erfolgt vielmehr eine Zuordnung der einzelnen Aufgaben zu speziellen Komponenten entsprechend deren Funktionsumfang.

Der Local Resource Analyzer (LRA) führt die Lastbestimmung auf jedem Rechnerknoten des Systems durch. Anschließend wird die dabei gewonnene Menge an Informationen verdichtet bzw. zu aussagekräftigen Größen aufbereitet. Diese Informationen bilden die Grundlage für die Realisierung der Lastbalancierung.

Das Informationsmanagement (Informationsaustausch und Informationsverwaltung) wird vom Global Resource Analyzer (GRA) verwirklicht. Die im Zusammenhang mit dem Lastinformationsaustausch empfangenen Lastinformationen der einzelnen Rechnerknoten, die am Lastausgleich beteiligt sind, werden durch den GRA in einem Lastvektor verwaltet. Dadurch ist eine globale Sicht auf die aktuelle Lastsituation der Rechnerknoten der Domäne verfügbar. Gleichzeitig werden die Lastinformationen schnell und kostengünstig durch den GRA für die Lastbalancierung bereitgestellt bzw. wird der Zugriff auf diese Informationen ermöglicht. Der GRA bildet somit eine Schnittstelle zwischen dem LRA und dem Global Query Manager (GQM).

Ein Basisdienst des GQM besteht in der Transformation des globalen Anfragebaumes in Teilanfragebäume, die parallel auf mehreren Rechnern ausgeführt werden können. In diesem Zusammenhang wird durch den GQM der vom SQL-Compiler gelieferte globale Verarbeitungsplan (Global Query Evaluation Plan - *GQEP*) in parallel abarbeitbare Verarbeitungspläne (QEP_i) umgewandelt. Die Verteilung der Teilanfragebäume auf für deren Verarbeitung geeignete Rechnerknoten erfolgt unter Berücksichtigung der Lastsituation im System. Um die Informationen bzgl. der Last einbeziehen zu können,

werden die Dienste des GRA beansprucht. Der GQM nutzt neben den Kenntnissen über die aktuelle Lastsituation auch Informationen aus dem Allokationsschema und dem Fragmentierungsschema, sowie berücksichtigt Angaben der Kostenabschätzung für die Lastbalancierungsentscheidung (siehe Abb. 11).

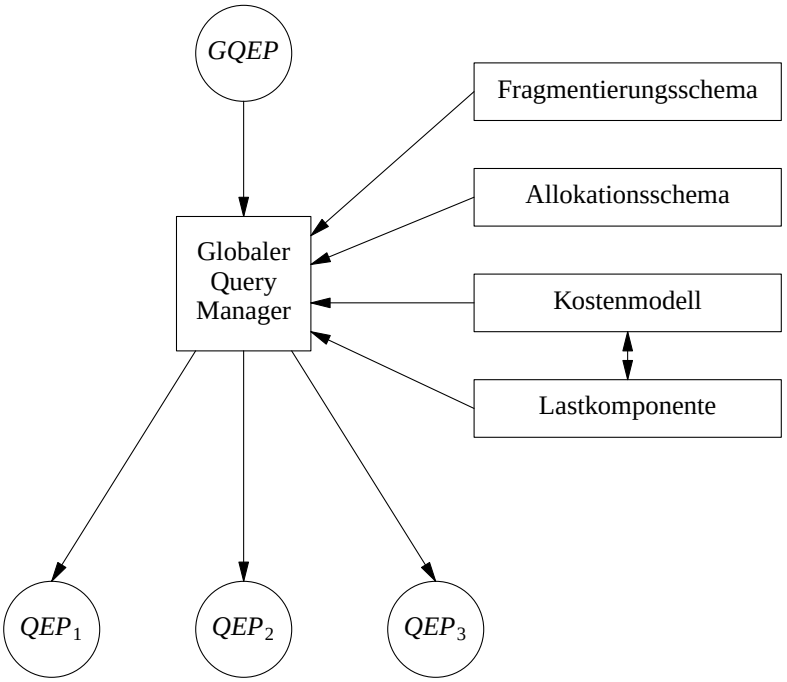

Abb. 11: Informationsquellen des GQM für die Entscheidungsfindung

Beim GQM erfolgt eine statische Lastbalancierung, da der Lastausgleich vor Beginn der Anfrageabarbeitung erfolgt (siehe Abs. 3.1.1.).

Die eigentliche Organisation der Ausführung eines Teilanfragebaumes wird vom Local Query Manager (LQM) realisiert. Dazu erhält der LQM eines Rechnerknotens entsprechend der Lastbalancierungsentscheidung des GQM eine Teilanfrage zur Bearbeitung. Ändert sich die Last des Bearbeitungsknotens während der Ausführung so stark, daß eine Überlastung des Rechners vorliegt, reagiert der LQM darauf, um negative Auswirkungen auf die Anfragebearbeitung zu verhindern. Durch eine Umverlagerung von Algebraprozessen auf andere, geringer belastete Rechner wird eine Balancierung der Last entsprechend der aktuellen Lastsituation auch während der Bearbeitung von Anfragen durchgeführt. Eine Grundlage für die zu treffenden Balancierungsentscheidungen

sind wiederum Kenntnisse sowohl über die Lastsituation des lokalen Rechnerknotens als auch über die Belastungszustände der anderen an der Lastbalancierung beteiligten Rechner. Daher werden auch durch den LQM die Dienste des LRA bzw. des GRA genutzt.

Durch den LQM und den GQM werden konkret die Operationen bzgl. des Prozeßtransfers und der Auswahl und Kooperation bei der Zielrechnersuche als Aufgaben der Lastkomponente zur Verwirklichung der Lastbalancierung bzw. der Lastverteilung realisiert.

In der vorliegenden Arbeit wird sich vorwiegend mit den Aufgaben des LRA und GRA, also mit der Lastbestimmung und dem Lastinformationsmanagement, beschäftigt. In den folgenden Abschnitten wird deren Realisierung ausführlich im Bezug zum HEAD - Projekt beschrieben.

5.2. Lastbestimmung

5.2.1. Die Lastparameter

Durch die Bestimmung der Last werden Informationen über die Belastung von Systemressourcen und über Systemaktivitäten für die weiteren Operationen der Lastbalancierung bereitgestellt. Die Last muß von den Ressourcen ermittelt werden, die sowohl für die Bearbeitung von Prozessen auf einem Rechnerknoten als auch bei der Verteilung der Last im System notwendig sind. Bei der Diskussion verschiedener Ansätze zur Lastmessung im Abschnitt 4.1.3 wurde deutlich, daß grundsätzlich Informationen über die Nutzung der CPU, des Speichers und des Kommunikationsmediums einzubeziehen sind. Darauf aufbauend werden folgende Größen betrachtet:

- CPU Run queue-Länge
- freier Speicherplatz
- Kollisionsrate
- ein- und ausgehende Pakete

Die gemessenen Parameter bilden zusammen einen Lastdeskriptor, aus dem Aussagen über die Rechnerknoten- und Netzwerkbelastung abgeleitet werden können.

CPU Run queue-Länge

In der bzw. den Run queues befinden sich alle Prozesse, die im lauffähigen Zustand sind. Nicht darin enthalten ist der aktuell laufende Prozeß. Beim Prozeß-Scheduling werden die CPU-Zyklen auf die lauffähigen Prozesse verteilt. Im UNIX-System gibt es 128 unterschiedliche Schedulingprioritäten [Leff90], nach denen die Zuteilung der CPU-Zyklen zu Prozessen abgestuft wird. Jedem lauffähigen Prozeß wird eine Schedulingpriorität zugeordnet, die periodisch vom System korrigiert wird. Entsprechend dieser Priorität werden die Prozesse in die Run queue so eingeordnet, daß an erster Stelle der Warteschlange der Prozeß mit der höchsten Priorität steht. Soll der nächste lauffähige Prozeß gestartet werden, so ist bei einer einzigen vorhandenen Run queue der erforderliche Aufwand für die Suche des Prozesses mit der höchsten Priorität relativ gering. Andere Operationen, z.B. das Verschieben der Prozesse innerhalb einer Warteschlange entsprechend der vom System korrigierten Priorität, sind bei nur einer Warteschlange aufwendiger [Leff90]. Ist nach Ablauf des Zeitquantums, das der Prozeß die CPU zugewiesen bekommt, ein Einordnen des Prozesses in die Run queue entsprechend der neuen Priorität erforderlich, ist dies mit größeren Anstrengungen verbunden. Wird für jede Schedulingpriorität eine Run queue angelegt, so würde dies einen erheblichen Aufwand für die Queue-Verwaltung bedeuten. Daher nutzt das UNIX-System 32 Run queues [Leff90]. Die Plazierung der Prozesse in die entsprechende Warteschlange wird dadurch erreicht, daß die Prozeßschedulingpriorität durch vier geteilt wird. Die Warteschlangen sind als ein Feld doppelt verketteter Listen mit entsprechenden Verweisen (*p_link*, *p_rlink*) organisiert (siehe Abb. 12).

Soll ein neuer lauffähiger Prozeß für die Zuweisung von CPU-Zyklen ausgewählt werden, so wird die Suche bei der Warteschlange mit der höchsten Priorität begonnen. Ist diese leer, wird danach die Warteschlange mit der nächst geringeren Priorität durchsucht. Dies wird solange fortgeführt, bis die erste besetzte Queue gefunden wurde. Deren erster Prozeß wird als nächster zu startender Prozeß ausgewählt. Nach Ablauf des dem Prozeß zugewiesenen CPU-Zeitquantums wird der Prozeß am Ende der Run queue wieder eingeordnet, von der er ausgewählt wurde. Sind in der Queue mehrere Prozesse vorhanden, wird anschließend der nun am Anfang der Queue stehende Prozeß mit dem gleichen Zeitquantum gestartet. Die Längen aller 32 Run queues zusammen liefern ein Anzeichen für die Last auf dem Rechnerknoten. Je mehr lauffähige Prozesse

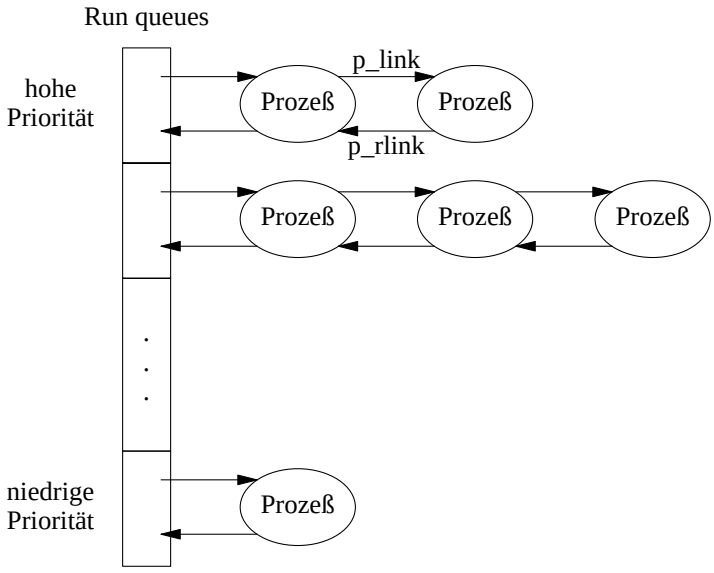

Abb. 12: CPU-Run queues [Leff90]

in den Run queues auf die CPU-Zuteilung warten, um so stärker ist die CPU belastet. Das UNIX-System stellt einen über eine Minute exponentiell geglätteten Durchschnitt der Run queue-Längen als 1min-Lastdurchschnitt (*1min loadaverage*) in der Variablen *loadave[0]* zur Verfügung. Diese Variable wird im weiteren durch die Variable *load_min1* repräsentiert.

Freier Speicherplatz

Neben der Hauptressource eines Rechnerknotens, der CPU, ist auch der Speicher eine entscheidende Systemressource. Jeder Prozeß benötigt für seine Ausführung Speicherraum. Ist nur wenig Speicherplatz verfügbar, so kann auf eine entsprechende Belastung des Rechners durch viele oder einige ressourcenintensive Prozesse geschlossen werden. Außerdem ist mit einem geringen freien Speicherplatz häufig ein entsprechender Anstieg an Swapping- bzw. Pagingaktivitäten verbunden. Daher dient auch die Angabe über den freien Speicherplatz als Indiz für die Belastung auf einem Rechnerknoten. Durch das UNIX-System wird eine aktuelle Größe über den freien verfügbaren Speicherplatz in der globalen Variablen *freemem* (Anzahl freier Speicherblöcke) zur Verfügung gestellt.

Kollisionsrate

Bei der Lastbalancierung in einem verteilten System ist neben den Belastungszuständen auf den einzelnen Rechnerknoten auch die Auslastung des Kommunikationsmediums zu berücksichtigen. Auf den gemeinsamen Kommunikationskanal greifen die Rechnerknoten zu, um Nachrichten auszutauschen. Bei dieser Nutzung des Netzwerkes muß ein wechselseitiger Ausschluß realisiert werden, da Nachrichten, die einander parallel oder überlappt gesendet werden, unbrauchbar sind. Eine solche Überlappung von Nachrichten bzw. das gleichzeitige Senden von Nachrichten auf einem Kommunikationskanal wird als Kollision bezeichnet [Kauf91]. Durch das Auftreten von Kollisionen wird die Kanalkapazität auf den tatsächlichen Durchsatz gemindert. Mit Anstieg der Anzahl an Übertragungen auf dem Netzwerk ist eine Erhöhung der Wahrscheinlichkeit verbunden, daß Kollisionen auftreten. Damit ist die Anzahl an Kollisionen ein Indiz für die Kanalbelastung. In der UNIX-Variablen *if_collisions* der Struktur *ifnet* wird die Anzahl der Kollisionen, die sich seit dem letzten Systemstart ereigneten, kumulativ gesammelt.

Ein- und ausgehende Pakete

Daten werden in Paketen (kleinste Transporteinheit) über das Netzwerk zwischen den Rechnerknoten bzw. Prozessen übertragen [Kauf91]. Die Anzahl der ein- und ausgehenden Pakete eines Rechnerknotens gibt somit sowohl Auskunft über die I/O-Tätigkeit des Rechners, als auch indirekt über die Belastung des Übertragungskanals. In den UNIX-Variablen *if_opackets* und *if_ipackets* der Struktur *ifnet* werden die Anzahl der ein- und ausgehenden Pakete kumulativ zusammengefaßt [Leff90].

5.2.2. Maßnahmen zur Verringerung des Lastbestimmungsoverhead

Die Leistung eines Lastbalancierungsalgorithmus hängt von der Wahl der Lastparameter, die während der Lastmessung betrachtet werden, sowie von der zugrundeliegenden Meßtechnik ab [Ni85]. Dabei ist zu berücksichtigen, daß durch die Operationen im Zusammenhang mit der Lastmessung ein zusätzlicher Overhead verursacht wird [Zhou88]. Um diesen zur Lastermittlung erforderlichen Aufwand gering zu halten, sind Einschränkungen bei der Wahl der Meßparameter und der Meßhäufigkeit notwendig. Grundsätzlich sind die Kosten und der Nutzen der Maßnahmen zur Lastbestimmung gegeneinander abzuwägen, so daß eine effiziente Ermittlung der Be-

lastung von Rechnerknoten und Kommunikationsmedium erfolgt.

Bei der Wahl der zu ermittelnden Parameter bzgl. der Last von Rechnern und Netzwerk erfolgt eine Beschränkung auf eine geringe Anzahl von aussagekräftigen Größen. Alle Größen werden vom UNIX-System selbständig gemessen und in internen Strukturen abgelegt bzw. gepuffert. Dadurch, daß nur vom UNIX-System bereitgestellte Parameter bei der Lastbestimmung genutzt werden, sind keine zusätzlichen systeminternen Routinen erforderlich, um die Größen der Belastung, wie z.B. die CPU-Run queue-Länge, selbst zu messen. Die Lastmeßmethode der Lastkomponente nutzt die Bibliotheksfunktionen *nlist* und *read* (siehe UNIX-User Manual 3). Durch den einmaligen Aufruf der Funktion *nlist* werden zu den in einer Liste zusammengefaßten, ausgewählten Belastungsparametern deren Abspeicherungsadressen im UNIX-Kern ermittelt. Die Funktion *read* nutzt diese Adressen, um auf die im UNIX-Kern abgelegten Belastungsinformationen zuzugreifen. Durch Einfügen zusätzlicher Systemvariablen in die Liste der Belastungsparameter können weitere Lastinformationen ausgelesen werden. Die Belastungsparameter bzw. Systemvariablen, aus denen Aussagen über die Last abgeleitet werden können, sind neben Sun-OS auch in anderen UNIX-kompatiblen Betriebssystemen verfügbar. Mit einem geringen Aufwand ist daher eine Anwendung der Lastmeßmethode auf Rechnerknoten, die unter anderen UNIX-Derivaten arbeiten, möglich.

Das Auslesen der Größen betreffs der Rechner- und der Netzwerkbelastung erfolgt in Intervallen. Mit der Wahl der Intervallänge wird die Häufigkeit des Auslesens festgelegt. Die Häufigkeit der Lastermittlung hat Einfluß sowohl auf die Aktualität und die Genauigkeit der gemessenen Lastparameter als auch auf den verursachten Overhead. Durch zu häufige Lastbestimmungen werden unnötige Aktionen veranlaßt, die keine neuen Erkenntnisse zu den betrachteten Größen liefern. Andererseits kann ein zu großes Intervall bewirken, daß ein signifikanter Belastungswechsel auf dem Rechnerknoten bzw. im Netzwerk nicht sofort erkannt wird und dadurch notwendige Reaktionen zu spät erfolgen. Findet das Auslesen der Lastgrößen in Meßintervallen statt, die an die Lastwechselzyklen des Systems angepaßt sind, können ausreichend aktuelle und aussagekräftige Informationen ermittelt werden [Ferr83]. Gleichzeitig werden einerseits uneffektive Messungen vermieden und andererseits aber trotzdem markante Laständerungen wahrgenommen.

5.2.3. Die Länge des Meßintervalls

Die Messung der CPU-Run queue-Länge bzw. des Belastungsparameters der wichtigsten Ressource des Rechnerknotens wurde als Untersuchungsbasis zur Bestimmung einer geeigneten Meßintervallänge herangezogen. Die Nutzung des über den Zeitraum von einer Minute geglätteten CPU-Run queue-Längenwertes ermöglicht Einschränkungen hinsichtlich der Auslesehäufigkeit, bei diesem relativ schnell wechselnden bzw. stark schwankenden Lastparameter. In welchen Zeitabständen sollte das Auslesen der CPU-Run queue-Länge erfolgen, ohne daß ein Informationsverlust bzgl. der Belastungsänderungen auftritt? Um diese Frage zu beantworten, wurde die CPU-Run queue-Länge auf einem belasteten Rechnerknoten gleichzeitig mit verschiedenen Meßintervallen ermittelt (siehe Tab.) (Angaben in Klammern: Differenz zwischen aktueller und zuvor gemessener Last).

Dabei erwies sich ein Intervall der Länge von fünf Sekunden als ausreichend, um einerseits unnötige Ausleseaktionen ohne neue Erkenntnisse zu verhindern und um auf der anderen Seite aktuelle Informationen über die Belastung und deren Änderungen zur Verfügung zu stellen. Bei den Intervallen der Länge 1 - 4 Sekunden fanden Messungen statt, die trotz einer kontinuierlich steigenden Last keine neuen Informationen lieferten (Differenz zwischen aktueller und zuvor bestimmter Last = Null).

Alle in Abschnitt 5.2.1. angegebenen Parameter bzgl. der Rechner- und der Netzwerkbelastung werden in einem Zeitintervall von fünf Sekunden aus dem UNIX-Kern ausgelesen. Damit steht nach jeweils fünf Sekunden folgender Lastdeskriptor bzw. Lastindex mit den aktuellen Belastungsinformationen für die weiteren Operationen im Zusammenhang mit der Lastbalancierung zur Verfügung.

loadindex: *time*
host
load_min1
freemem
if_collisions
if_ipackets
if_opackets

Zeit in sec	**ermittelte Last mit Meßintervall der Länge von**						
	1 sec	2 sec	3 sec	4 sec	**5 sec**	6 sec	7 sec
1	140	140	140	140	140	140	140
2	140 (0)						
3	140 (0)	140 (0)					
4	140 (0)		140 (0)				
5	140 (0)	140 (0)		140 (0)			
6	**212 (72)**				**212 (72)**		
7	212 (0)	212 (72)	212 (72)			212 (72)	
8	212 (0)						212 (72)
9	212 (0)	212 (0)		212 (72)			
10	212 (0)		212 (0)				
11	**257 (45)**	257 (45)			**257 (45)**		
12	257 (0)						
13	257 (0)	257 (0)	257 (45)	257 (45)		257 (45)	
14	257 (0)						
15	257 (0)	257 (0)					257 (45)
16	**298 (41)**		298 (41)		**298 (41)**		
17	298 (0)	298 (41)		298 (41)			
18	298 (0)						
19	298 (0)	298 (0)	298 (0)			298 (41)	
20	298 (0)						
21	**336 (38)**	336 (38)		336 (38)	**336 (38)**		
22	336 (0)		336 (38)				336 (79)
23	336 (0)	336 (0)					
24	336 (0)						
25	336 (0)	336 (0)	336 (0)	336 (0)		336 (38)	
26	**371 (35)**				**371 (35)**		
27	371 (0)	371 (35)					
28	371 (0)		371 (35)				
29	371 (0)	371 (0)		371 (35)			371 (35)
30	371 (0)						
31	**403 (32)**	403 (32)	403 (32)		**403 (32)**	403 (67)	
32	403 (0)						
33	403 (0)	403 (0)		403 (32)			
34	403 (0)		403 (0)				
35	403 (0)	403 (0)					
36	**432 (29)**				**432 (29)**		432 (61)
37	432 (0)	432 (29)	432 (29)	432 (29)		432 (29)	
38	432 (0)						

Diese Daten bilden eine Grundlage für die Lastbalancierungsentscheidungen. Eine eindeutige Zuordnung der ermittelten Lastinformationen zu einem Rechnerknoten erfolgt, indem der Lastindex zusätzlich den jeweiligen Hostnamen enthält. Außerdem wird der Zeitpunkt, wann die Belastungsinformationen ermittelt wurden, im Lastdeskriptor festgehalten. Als Zeitstempel wird die vom UNIX-Systemaufruf *time()* bereitgestellte Zeit genutzt (vergangene Zeit in Sekunden seit dem 01.01.1970 00.00 Uhr). Aus dem

Zeitstempel im Lastindex kann abgeleitet werden, wie aktuell die enthaltenen Belastungsinformationen sind bzw. welches Gewicht diesen Kenntnissen auf Grund ihres Alters bei weiteren Entscheidungen beizumessen ist.

Die Länge des Lastmeßintervalls ist ein Parameter der gesamten Lastmeßmethode, der an neue Erkenntnisse und an geänderte Anforderungen angepaßt werden kann.

In den Abbildungen 13 - 16 ist der Meßwerteverlauf der Lastparameter des Lastdeskriptors bei Untersuchungen mittels der Wisconsin-Benchmark dargestellt. Bei dieser Studie wurde eine sehr kleine Anzahl von Kollisionen ermittelt. Daher erfolgt hier keine Darstellung der Kollisionsrate. Bei den weiteren Untersuchungen wird die Kollisionsrate aber trotzdem berücksichtigt.

Um aus den während der Lastmessung ermittelten Daten konkrete Aussagen über die Belastung und deren Änderungen abzuleiten zu können, die bei den weiteren Lastausgleichsentscheidungen genutzt werden, erfolgt eine Verdichtung der Meßinformationen. Die ausführliche Darstellung der Aufbereitung der Meßdaten und der damit verbundenen Operationen erfolgt im folgenden Abschnitt.

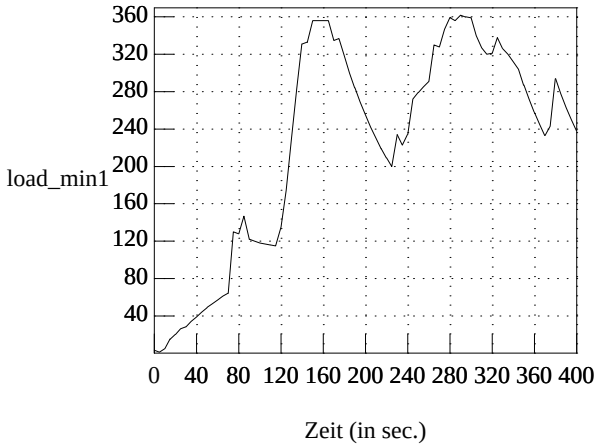

Abb. 13: CPU-Run queue-Länge

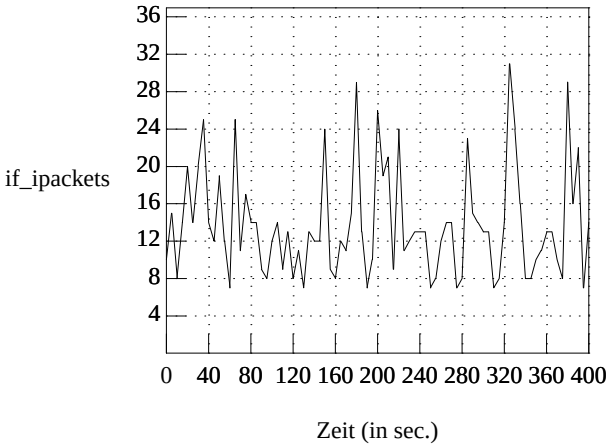

Abb. 15: Anzahl der eingehenden Pakete

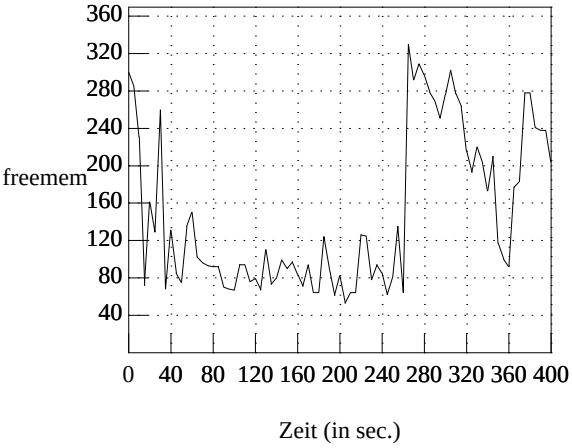

Abb. 14: Anzahl freier Speicherblöcke

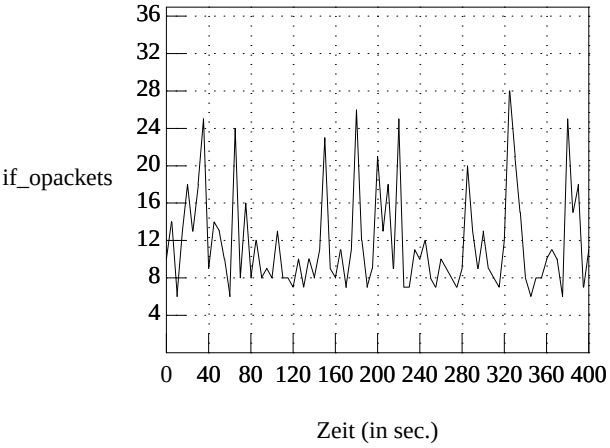

Abb. 16: Anzahl der ausgehenden Pakete

5.3. Verdichtung der Meßwerte

Nachdem die einzelnen Größen des Lastindex in einem festgelegten Meßintervall gemessen bzw. aus den internen Systemstrukturen ausgelesen wurden, sind Datenpaare verfügbar. Ein Datenpaar wird aus dem absoluten Meßwert bzgl. der Belastung einer Ressource und dem Zeitpunkt der Messung gebildet. Es gibt verschiedene Möglichkeiten diese Informationen für die weiteren Aufgaben im Rahmen der Lastbalancierung zu nutzen. Einerseits können die Strategien mit einzelnen Absolutwerten arbeiten oder andererseits kann eine Verdichtung der Daten angestrebt werden. Durch diese Aufbereitung der Daten wird erreicht, daß die zu treffenden Entscheidungen (z.B. über die Notwendigkeit eines Lastinformationsaustausches) auf der Grundlage von mehreren Meßwerten eines zurückliegenden Zeitraumes basieren. Der Zeitraum kann die Meßwerte entweder seit dem Beginn der Messungen oder von einer begrenzten Zeitperiode in der Vergangenheit umfassen [Heis88]. Da die erste Variante einen großen Aufwand zur Verwaltung der gesamten Vorgeschichte verursacht und Informationen dieses Umfangs für die Lastbalancierung nicht erforderlich sind, werden die Lastwerte eines zurückliegenden Zeitintervalls betrachtet. Die Größe des zu berücksichtigenden Zeitintervalls ist den Zielanforderungen der Lastbalancierung anzupassen. Eine Möglichkeit der Komprimierung von Meßwerten ist die lineare Regressionsanalyse, deren theoretische Grundlagen und Anwendung bei der Lastkomponente im folgenden dargestellt werden.

5.3.1. Theoretische Grundlagen der linearen Regressionsanalyse

Durch die Regressionsanalyse wird die Art des Zusammenhangs zwischen zwei Merkmalen X und Y untersucht. X ist eine unabhängige Größe, die als Einflußgröße bezeichnet wird. Y stellt eine abhängige Zielgröße dar. Zwischen X und Y besteht eine stochastische Abhängigkeit. Mittels der Regressionsanalyse soll diese stochastische Abhängigkeit auf eine funktionale Beziehung abstrahiert werden [Sach92]. Durch einen funktionalen Zusammenhang zwischen X und Y wird es möglich, aus vorgegebenen bzw. zu beliebigen Werten der unabhängigen Einflußgröße X die jeweils abhängige Zielgröße Y abzuschätzen. X und Y bilden eine Grundgesamtheit (X, Y). Aus dieser Grundgesamtheit liegen Stichproben in Form von Datenpaaren (x_i, y_i) $x_i \in X, y_i \in Y$ $(i = 1, 2, .., n)$ vor. Die Stichproben werden genutzt, um aus den darin enthaltenden

Informationen Aussagen über die Grundgesamtheit und deren Größen abzuleiten [Beye88]. Daher werden zur Erreichung einer funktionalen Abhängigkeit die beobachteten Werte von Y (y_i , $i = 1, 2, . . , n$) herangezogen. Um die zu untersuchende Beziehung $y_i(x_i)$ ($i = 1, 2, . . , n$) als Funktion darzustellen, werden die y_i-Werte an eine Funktion (Regressionsgleichung) $\hat{y} = f(x)$ angepaßt. Die Funktionswerte der Regressionsgleichung ($\hat{y}$) weichen von den gegebenen y_i-Werten ($i = 1, 2, . . , n$) ab. Bei der linearen Regression wird der funktionale Zusammenhang zwischen Y und X mittels einer linearen Funktion zwischen der Einflußgröße und der Zielgröße dargestellt. Die Funktionsgleichung hat die Form einer Geradengleichung. Diese Gerade wird als Regressionsgerade bezeichnet (siehe Abb. 17):

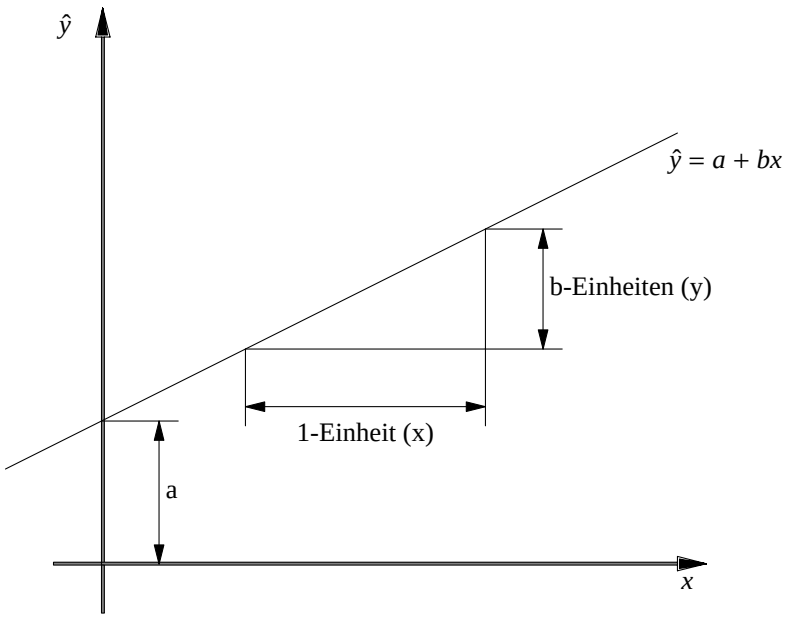

Abb. 17: Regressionsgerade und deren Parameter

Funktionsgleichung der Regressionsgeraden:

$$\hat{y} = a + bx \tag{1}$$

Die Gerade verfügt über die Parameter a und b:

a = Abstand des Schnittpunktes der Regressionsgeraden mit der Ordinate (y-Achse) von der Abszisse (x-Achse), wird als Achsenabschnitt bezeichnet

b = Anstieg der Regressionsgeraden, wird als Regressionskoeffizient bezeichnet

Durch den Anstieg wird verdeutlicht, um welchen Wert sich die $\hat{y}$-Größe ändert, wenn sich die x-Größe um eine Einheit verändert. Durch den Regressionskoeffizienten wird also festgelegt, ob bei zunehmenden x-Werten die zugehörigen $\hat{y}$-Werte und somit auch die Regressionsgerade ansteigt (b = positiv) oder ob die $\hat{y}$-Werte und die Regressionsgerade abfallen (b = negativ).

Der durch die Regressionsfunktion berechnete mittlere theoretische Wert $\hat{y}_i$ weicht von der eigentlichen Zielgröße y_i derart ab, daß die Differenz $d = y_i - \hat{y}_i$ ($i = 1, 2, \ldots, n$) die vertikale Abweichung verdeutlicht (siehe Abb. 18).

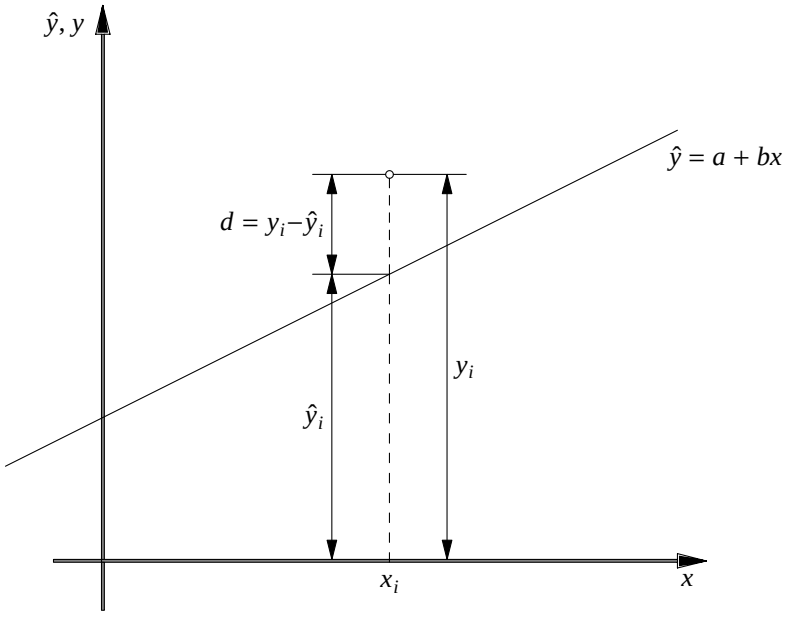

Abb. 18: Vertikale Abweichung

5.3.2. Bestimmung der Parameter der Regressionsgeraden

Um die Regressionsgerade bestimmen zu können, ist die Abschätzung derer beiden Parameter a und b erforderlich. Es erfolgt eine Abschätzung von a und b gegenüber dem tatsächlichen Regressionskoeffizienten β und dem Achsenabschnitt α der Grundgesamtheit (X, Y), da nur Stichproben aus (X, Y) in Form der Datenpaare (x_i, y_i) $(i = 1, 2, .., n)$ genutzt werden können. Die Regressionsgerade soll sich in ihrem Verlauf möglichst gut an alle ermittelten y_i-Werte anpassen. Diese Anforderung kann bei der Anwendung des Verfahrens der kleinsten quadratischen Abweichung in folgendem Zielkriterium definiert werden [Förs79]:

Zielkriterium für die Regressionsgerade(a):

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \rightarrow \min \quad bzw. \sum_{i=1}^{n} d^2 \rightarrow \min \tag{2}$$

Die Summe der quadratischen Abweichungen der ermittelten Werte y_i von den theoretischen Werten $\hat{y}_i$ der Regressionsgeraden soll ein Minimum werden (siehe Abb. 19). Das Minimierungsproblem läßt sich durch Einsetzen der Regressionsgleichung (1) in (2) wie folgt darstellen:

Zielkriterium für die Regressionsgerade(b):

$$\sum_{i=1}^{n}(y_i - a - bx_i)^2 \rightarrow \min \tag{3}$$

Nach dieser Methode, die als Gaußsche Methode der kleinsten Quadratsummen bzw. Methode der kleinsten Quadrate bezeichnet wird, können die beiden Parameter der Regressionsgeraden ermittelt werden [Sach92]. Dabei werden alle gegebenen Datenpaare (x_i, y_i) $(i = 1, 2, .., n)$ mit in die Berechnung einbezogen. Alle Werte y_i $(i = 1, 2, .., n)$ werden derart beachtet, daß die ermittelte Regressionsgerade alle Abweichungen $(y_i - \hat{y}_i)$ $(i = 1, 2, .., n)$ unter dem gegebenen Zielkriterium ausgleicht.

Zur Vereinfachung bei der Darstellung der Berechnungsformel wird folgende Vereinbarung getroffen.

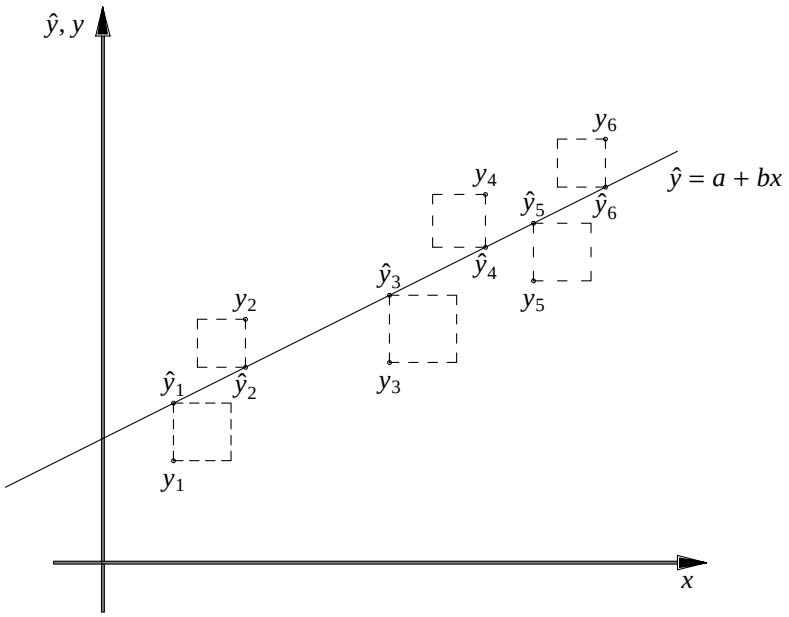

Abb. 19: Die Summe der quadratischen Abweichungen

Vereinfachungsvereinbarung:

$$\sum x = \sum_{i=1}^{n} x_i \quad bzw. \quad \sum y = \sum_{i=1}^{n} y_i \tag{4}$$

Die beiden Parameter der Geraden werden nach folgenden Formeln bestimmt.

Funktionen zur Abschätzung der Regressionsgeradenparameter (a):

$$b = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \tag{5}$$

$$a = \frac{\sum y \sum x^2 - \sum x \sum xy}{n\sum x^2 - (\sum x)^2} \tag{6}$$

Im weiteren Verlauf der Regressionsanalyse werden Teile von der Berechnung der Geradenparameter nochmals genutzt. Daher werden folgende Zusammenfassungen vereinbart.:

Vereinbarung von Zusammenfassungen:

$$Q_X = n\sum x^2 - (\sum x)^2 \tag{7}$$

$$Q_Y = n\sum y^2 - (\sum y)^2 \tag{8}$$

$$Q_{XY} = n\sum xy - \sum x \sum y \tag{9}$$

Unter Anwendung dieser Zusammenfassungen lassen sich die Formeln zur Berechnung der Regressionsgeradenparameter wie folgt angeben:

Funktionen zur Abschätzung der Regressionsgeradenparameter (b):

$$b = \frac{Q_{XY}}{Q_X} \tag{10}$$

$$a = \frac{\sum y \sum x^2 - \sum x \sum xy}{Q_X} \tag{11}$$

Nach der Abschätzung der Parameter a und b der Regressionsgeraden können unter Anwendung der Geradenformel $\hat{y} = a + bx$ die Funktionswerte $\hat{y}_i$ für gegebene Werte der Einflußgröße x_i ermittelt werden. Die Funktionswerte $\hat{y}_i$ bilden nach dem Kriterium der kleinsten Quadrate die beste lineare Annäherung an den Werteverlauf von y_i, da die Streuung der Abweichungen d den kleinstmöglichen Wert hat.

5.3.3. Statistische Überprüfung der Regressionsgeradenparameter

Die berechneten Regressionsgeradenparameter sind Abschätzungen des tatsächlichen Regressionskoeffizienten β und des Achsenabschnittes α der Grundgesamtheit (X, Y). Um zu gewährleisten, daß die abgeschätzten Parameter statistisch gesichert bzw. signifikant sind, erfolgt deren statistische Überprüfung. Im weiteren wird sich auf die Kontrolle der statistischen Absicherung des Regressionskoeffizienten beschränkt, da nur dieser Parameter im weiteren Verlauf der Lastbalancierung genutzt wird. Es wird angenommen, daß die Regressionsgeradenparameter einer t-Verteilung genügen, wodurch

der t-Test zur Signifikanzkontrolle angewendet werden kann [Förs79]. In diesem Zusammenhang lassen sich die folgende Hypothese H_0 und die Alternativhypothese H_A aufstellen.

H_0: $\beta = 0$: es besteht kein signifikanter Einfluß der Größe x_i auf die Zielgröße y_i $(i = 1, 2, .., n)$

H_A: $\beta \neq 0$: es existiert ein wesentlicher Einfluß von x_i auf y_i $(i = 1, 2, .., n)$

Da keine Angaben über den realen Regressionskoeffizienten β vorhanden sind, wird mit der Annahme $\beta = 0$ ($=H_0$) geprüft. Die Hypothese H_0 wird gegen die Alternativhypothese H_A geprüft, wobei H_0 abgelehnt werden kann, wenn folgende Bedingung gilt:

Ablehnungsbedingung für H_0:

$$\hat{t} = \frac{|b|}{s_b} \geq t_{n-2;\alpha;zweiseitig} \tag{12}$$

Bedeutung der einzelnen Größen:

$\hat{t}$: berechneter t-Wert des Regressionskoeffizienten b

$$\hat{t} = \frac{|b|}{s_b} \tag{13}$$

b : Regressionskoeffizienten, Berechnung siehe Formel (10)

s_b : Standardabweichung des Regressionskoeffizienten b

$$s_b = \sqrt{\frac{Q_Y - (Q_{XY})^2 \,/\, Q_X}{n-2 \,/\, Q_X}} \tag{14}$$

$t_{n-2;\alpha;zweiseitig}$: Vergleichswert für den berechneten t-Wert (=$\hat{t}$), der sich für eine gegebene Irrtumswahrscheinlichkeit, gegebene Freiheitsgrade und unter Beachtung, ob eine einseitige oder zweiseitige Fragestellung

vorliegt, aus der Tafel der t-Verteilung ablesen läßt.

n : Anzahl der zugrundeliegenden Datenpaare (x_i, y_i) $(i = 1, 2, .., n)$ für die Berechnung bzw. Abschätzung des Regressionskoeffizienten

$n-2$: Anzahl der Freiheitgrade (Anzahl der Werte, die von n vorhandenen Einzelwerten noch frei wählbar sind)

α : angenommene Irrtumswahrscheinlichkeit

zweiseitig : zweiseitige Fragestellung, negative und positive Regressionskoeffizienten werden betrachtet

Erfüllt der berechnete $\hat{t}$-Wert die Ablehnungsbedingung für H_0, wird damit die Alternativhypothese H_A bestätigt. Dadurch wird nachgewiesen, daß der ermittelte Regressionskoeffizient b statistisch mit einer Irrtumswahrscheinlichkeit von α abgesichert ist.

Ein Vertrauensbereich (= Konfidenzbereich) ist ein Intervall um den geschätzten Wert mit der Annahme, daß darin der reale Regressionskoeffizient der Grundgesamtheit liegt. Mit einer Vertrauenswahrscheinlichkeit von $P = 1-\alpha$ enthält der Vertrauensbereich den realen Parameter β der Grundgesamtheit bzw. mit der Irrtumswahrscheinlichkeit von α liegt der Parameter nicht im Konfidenzbereich. Die Grenzen des Vertrauensbereiches werden angegeben durch:

Grenzen des Vertrauensbereich um den geschätzten Regressionskoeffizient

$$b \pm t_{n-2;\alpha;zweiseitig} * s_b \tag{15}$$

Der Vertrauensbereich für β um den geschätzten Parameter b bei einer Irrtumswahrscheinlichkeit α umfaßt folgendes Intervall:

Vertrauensbereich um den geschätzten Regressionskoeffizient

$$[b - t_{n-2;\alpha;zweiseitig} * s_b \; , \; b + t_{n-2;\alpha;zweiseitig} * s_b] \tag{16}$$

5.3.4. Nutzung der Regressionsanalyse bei der HEAD-Lastkomponente

Nach einmaligem Messen der Größen des Lastdeskriptors liegt zu jeder Belastungsgröße ein Absolutwert vor. Ein einzelner Wert zu jeder Belastungsgröße liefert keine aussagekräftige Information über den Zustand des Knotens, um darauf basierend Entscheidungen für Prozeßübertragungen zu treffen. Daher werden Meßinformationen über ein längeres Intervall gesammelt. Diese Daten werden durch die oben beschriebene Regressionsanalyse aufbereitet bzw. verdichtet. Eine einzelne Belastungsgröße, z.B. der 1min-Lastdurchschnitt (siehe Abs. 5.2.1.) über ein Zeitintervall betrachtet, zeigt häufig einen schwankenden Kurvenverlauf (siehe Abb. 20).

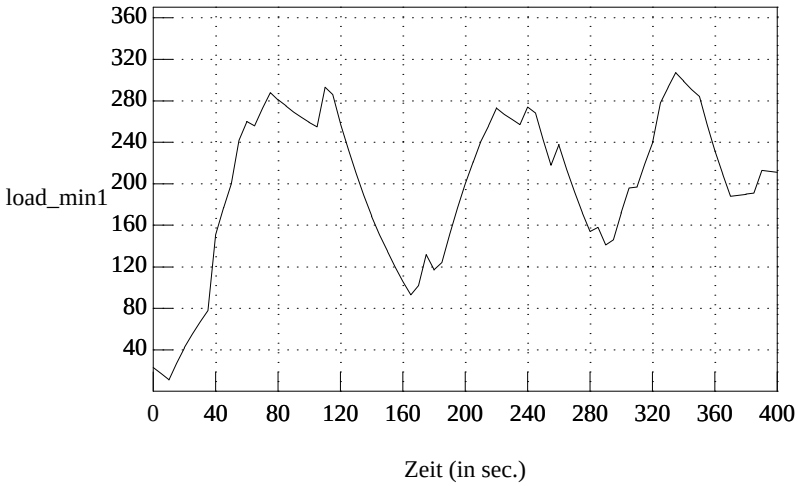

Abb. 20: Kurvenverlauf des 1min-Lastdurchschnitts

Aus einem solchen Werteverlauf lassen sich kaum exakte Aussagen darüber ableiten, wie die CPU belastet ist bzw. ob ein Prozeßtransfer notwendig ist. Durch die Anwendung der linearen Regressionsanalyse wird versucht, die stochastische Abhängigkeit von Meßgröße und Zeit durch eine lineare funktionale Abhängigkeit von beiden Größen vereinfacht darzustellen. Dabei ist die Meßgröße die Zielgröße, auf welche die Zeit Einfluß nimmt.

Mittels der Regressionsgeraden, deren Parameterberechnung während der Analyse erfolgt, werden positive und negative Schwankungen im Meßgrößenverlauf in einem Zeitintervall ausgeglichen. Der Verlauf der Geraden zeigt in welcher Tendenz die schwankende Kurve der Meßwerte während des zurückliegenden Zeitintervalls

resultiert. Aussagen über die Tendenz werden aus dem Anstieg der Geraden (= Regressionskoeffizient *b*) abgeleitet. Der Anstieg der Regressionsgeraden gibt Auskunft darüber, wie stark die Meßwerte im vergangenen Zeitintervall tendenziell ansteigen oder abfallen (siehe Abb. 21).

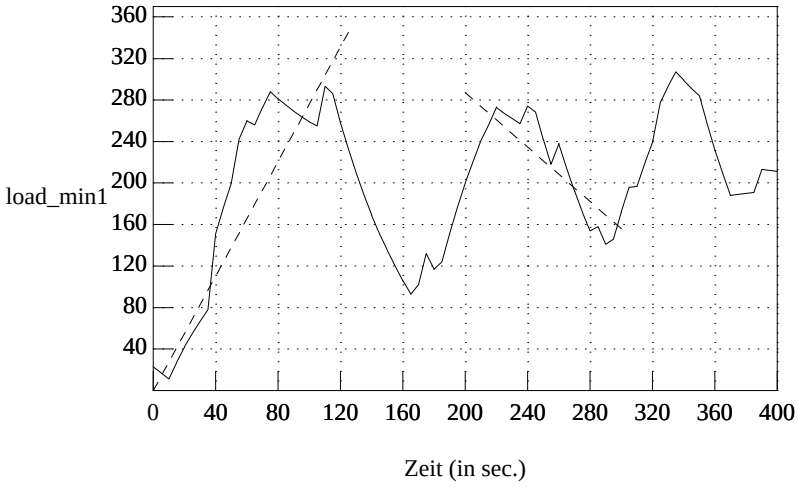

Abb. 21: Regressionsgeraden im Meßgrößenverlauf

Die Kenntnisse über den Anstieg und somit über die Tendenz, die die Meßwerte im vergangenen Intervall zeigten, können für die weiteren Aktionen im Rahmen der Lastbalancierung genutzt werden. Ob der Anstieg der Regressionsgeraden statistisch abgesichert, also aussagekräftig ist, wird durch den oben dargestellten *t*-Test bei einer angenommenen Irrtumswahrscheinlichkeit von fünf Prozent überprüft. Die Regressionsgerade kann in ihrem Verlauf ansteigen (positiver Anstieg) oder abfallen (negativer Anstieg). Daher liegt bei der Überprüfung der statistischen Absicherung eine zweiseitige Fragestellung vor.

Die Regressionsanalyse bietet die Möglichkeit einer Tendenzberechnung aus den Meßwerten des zurückliegenden Zeitintervalls. Daraus können konkretere Aussagen über den Meßgrößenverlauf im Vergleich zu anderen Methoden der Meßwerteverdichtung, z.B. Berechnung des Mittelwertes oder der Standardabweichung aus vorhandenen Lastparameterwerten, abgeleitet werden. Gleichzeitig können die ermittelten Regressionsgeradenparameter genutzt werden, um einen zukünftigen Trend für den Verlauf der Lastparameterkurve zu berechnen. Da aber die Lastparameterwerte häufige Schwan-

kungen aufweisen, wird diese Möglichkeit der Vorhersage der zukünftigen Belastung nicht genutzt.

Die Länge des Berechnungsintervalls

Der Berechnung der Regressionsgeraden werden die Meßwerte einer Belastungsgröße von einem zurückliegenden Zeitintervall zu Grunde gelegt. Dieses Intervall wird als Berechnungsintervall bezeichnet. Für den Verlauf der Regressionsgeraden und somit für ihren Anstieg ist die Intervallgröße, aus der Meßwerte einbezogen werden, von entscheidender Bedeutung. Die Regressionsgerade soll die Bedingung $\sum d^2 \rightarrow \min$ erfüllen. Je länger das zu berücksichtigende Zeitintervall ist, aus dem Meßwerte in die Berechnung einfließen, um so mehr Schwankungen im Meßwerteverlauf müssen ausgeglichen werden. Dies führt dazu, daß der Anstieg der Regressionsgeraden geringer wird, als bei Geraden, die auf kürzeren Zeitintervallen basieren. Bei kurzen Berechnungsintervallen paßt sich die Regressionsgerade stärker dem eigentlichen Kurvenverlauf an, da weniger Schwankungen auszubalancieren sind (siehe Abb. 22 - 25).

Mit der Länge des Berechnungsintervalls wird festgelegt, inwieweit bereits geringe Schwankungen des Meßwerteverlaufs Beachtung finden sollen. Bei der Festlegung der Intervallänge ist auch abzuwägen zwischen dem Aufwand, der durch jede Berechnung verursacht wird und den Anforderungen nach aussagekräftigen Informationen über die Belastung des Knotens zur weiteren Entscheidungsfindung. Unter Berücksichtigung dieser Bedingungen und aus bisherigen Untersuchungen zeigte sich ein Berechnungsintervall der Länge von 100 Sekunden als angemessen.

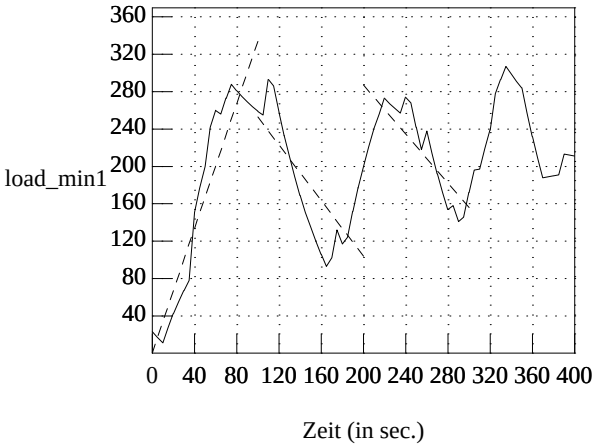

Abb. 22: Berechnungsintervallänge = 100 sec

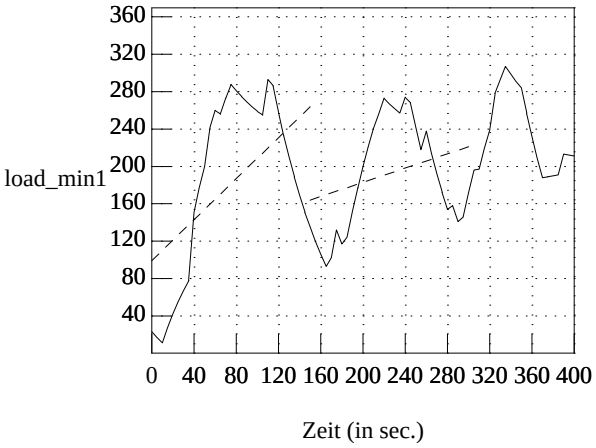

Abb. 24: Berechnungsintervallänge = 150 sec

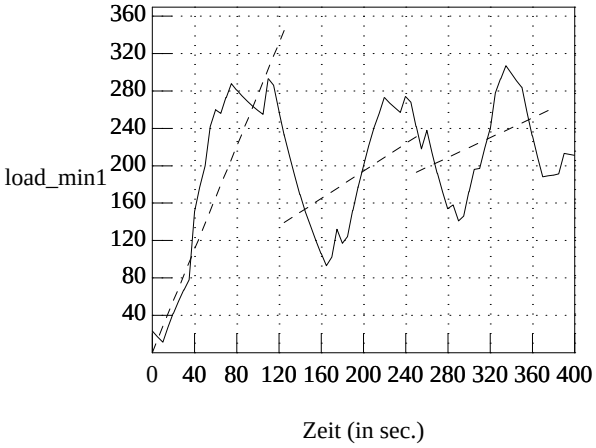

Abb. 23: Berechnungsintervallänge = 125 sec

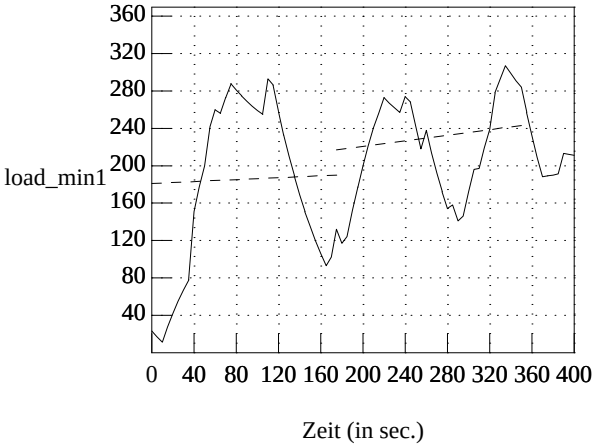

Abb. 25: Berechnungsintervallänge = 175 sec

Berechnung des Anstiegs der Regressionsgeraden

Für die Berechnung einer Regressionsgeraden werden die Meßwerte der letzten 100 Sekunden betrachtet. Dies bedeutet, daß erst nach Ablauf dieses Berechnungsintervalls der Anstieg der Geraden vorliegt. Da diese Zeitspanne den Anforderungen nicht gerecht wird, die die Lastbalancierung an die Verfügbarkeit von aussagekräftigen Lastinformationen stellt, erfolgt eine Untergliederung des Berechnungsintervalls in fünf Teilintervalle ($s[1], \ldots, s[5]$). Jedes Teilintervall enthält die Meßwerte von 20 Sekunden. Bei einer Meßintervallänge von fünf Sekunden entspricht dies vier Meßwerten pro Teilintervall und insgesamt 20 Werten pro Berechnungsintervall. Die 20 Meßwerte des Berechnungsintervalls sind wie folgt auf die fünf Teilintervalle aufgeteilt (siehe Tab.).

Meßwert zum Zeitpunkt t_i	y_{t1}	y_{t2}	y_{t3}	y_{t4}	y_{t5}	y_{t6}	y_{t7}	y_{t8}	y_{t9}	y_{t10}	y_{t11}	y_{t12}	y_{t13}	y_{t14}	y_{t15}	y_{t16}	y_{t17}	y_{t18}	y_{t19}	y_{t20}
Teilintervall	$s[1]$				$s[2]$				$s[3]$				$s[4]$				$s[5]$			

Im Teilintervall $s[5]$ befinden sich die vier zu letzt gemessenen und in $s[1]$ die vier ältesten Meßwerte des gesamten betrachteten Berechnungsintervalls. Durch diese Unterteilung ist es möglich, das Berechnungsintervall gleitend zu erneuern. Dabei werden die Meßwerte der Teilintervalle stets nach 20 Sekunden verschoben ($s[i] = s[i+1]$, $i = 1, \ldots 4$), um die neuesten Meßwerte dieser Zeitperiode ins Berechnungsintervall aufzunehmen. Bei diesem Vorgehen fallen die ältesten Meßwerte ($s[1]$ vor der Verschiebung) aus dem Berechnungsintervall heraus. Die aktuellsten Werte, die während der letzten 20 Sekunden gemessen wurden, bilden das neue Teilintervall $s[5]$ (siehe Tab.). Dies bedeutet, daß bereits nach der Dauer eines Teilintervalls eine neue Regressionsgerade berechnet werden kann, die die aktuellsten Meßwerte der letzten 20 Sekunden im gesamten Berechnungsintervall von 100 Sekunden berücksichtigt. Nach Ausführung der Regressionsanalyse steht mit dem Anstieg der Geraden eine Größe zur Verfügung, die Aufschluß darüber gibt, wie stark sich die betrachtete Meßgröße im zurückliegenden Zeitintervall von 100 Sekunden verändert hat. In Abbildung 26 sind die Regressionsgeraden im Meßgrößenverlauf ohne die Untergliederung des Berechnungsintervalls in Teilintervalle dargestellt. Abbildung 27 verdeutlicht im Vergleich

Meßwert zum Zeitpunkt t_i	y_{t1}	y_{t2}	y_{t3}	y_{t4}	y_{t5}	y_{t6}	y_{t7}	y_{t8}	y_{t9}	y_{t10}	y_{t11}	y_{t12}	y_{t13}	y_{t14}	y_{t15}	y_{t16}	y_{t17}	y_{t18}	y_{t19}	y_{t20}	y_{t21}	y_{t22}	y_{t23}	y_{t24}
Teilintervall v. Verschieb.	s[1]				s[2]				s[3]				s[4]				s[5]				werden neu gemessen			
Teilintervall n. Verschieb.	entfallen				s[1]				s[2]				s[3]				s[4]				s[5]			

dazu den Verlauf der Regressionsgeraden, wenn bereits nach Ablauf eines Teilintervalls eine neue Berechnung der Regressionsgerade erfolgt (aus Gründen der Übersichtlichkeit sind nur die Regressionsgeraden nach jeweils 40 Sekunden dargestellt).

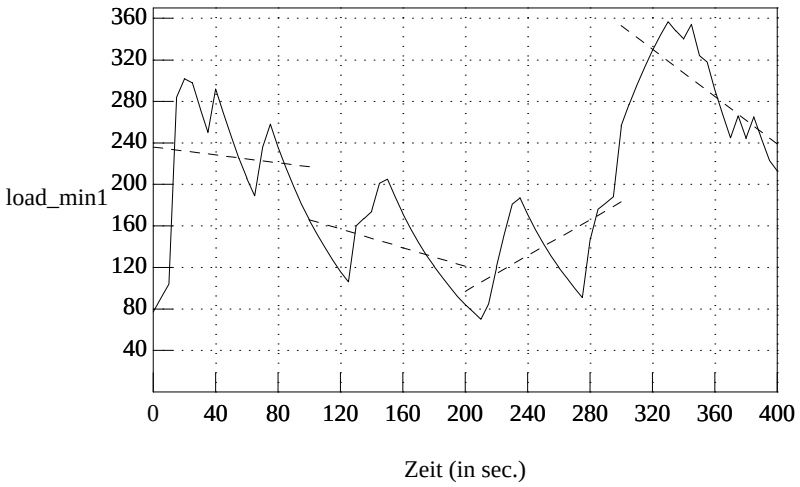

Abb. 26: Regressionsgeraden ohne Teilintervalle

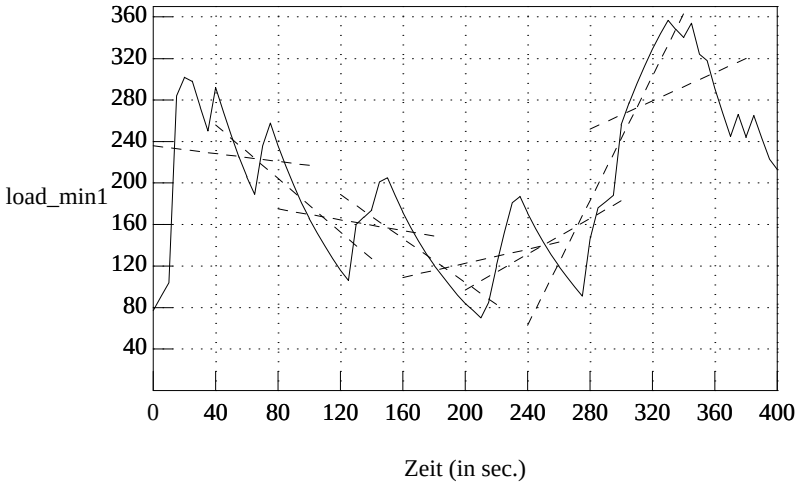

Abb. 27: Regressionsgeraden mit Teilintervallen

Die Durchführung einer Regressionsanalyse und somit die Ermittlung der Regressionsgeradenparameter erfolgt für jeden der im Lastdeskriptor enthaltenen Lastparameter. Die dabei berechneten Werte, wie Anstieg *b* und *t*-Wert, werden im Lastdeskriptor mit den zugehörigen Absolutwerten abgespeichert. Die Klasse `mval` definiert die Struktur eines einzelnen Lastparameters, die auch die berechneten Werten der Regressionsanalyse umfaßt.

```
class mval {

public:

      long        val;
      double      b;
      double      t;
      ...
};
```

val enthält den Absolutwert des gemessenen Lastparameters.
b ist der Anstieg der Regressionsgerade, die aus den Lastparameterwerten des zurückliegenden Berechnungsintervalls ermittelt wurde.
t repräsentiert den während des *t*-Test berechneten $\hat{t}$-Wert bzgl. der Überprüfung, ob der berechnete Anstieg *b* der Regressionsgeraden statistisch abgesichert ist.

Die Klasse `loadindex` enthält die Struktur des Lastdeskriptors.

```
class loadindex {

public:
      ...
      char       *host;
      long       time;
      mval       load_min1;
      mval       freemem;
      mval       collisions;
      mval       ipackets;
      mval       opackets;
      ...
};
```

5.4. Informationsmanagement

Die während der Messung und der Informationsverdichtung ermittelten Größen über die lokale Belastung des Knotens müssen für die Lastbalancierungsentscheidungen nutzbar sein. Dies erfordert den ***Zugriff auf die Lastinformationen*** mittels Anfragen nach festgelegten Regeln und das Halten der Daten auf einem angemessenen aktuellen Stand. Da das Lastbalancierungsverfahren kooperativ ist, also Belastungsinformationen anderer Knoten in die Verteilungsentscheidung einzubeziehen sind, muß die Versendung und der ***Austausch von Informationen*** zwischen den Rechnerknoten realisiert werden. Ebenso ist die ***Verwaltung der Daten*** über die Belastung der Knoten erforderlich.

Das Informationsmanagement erfordert eine Kommunikation zwischen den Rechnerknoten. Daher werden einige grundlegende Ausführungen zum vorhandenen Netzwerk und zu den möglichen Arten der Kommunikation vorangestellt.

5.4.1. Grundlagen

Netzwerk

Ein lokales Netz auf **Ethernet**-Basis IEEE 802.3 mit einer maximalen Übertragungsrate

von 10 MBit/s bildet die Grundlage für die Kommunikation zwischen den Rechnerknoten bzw. deren Lastkomponenten (Resource Analyzer) (siehe Abb. 28).

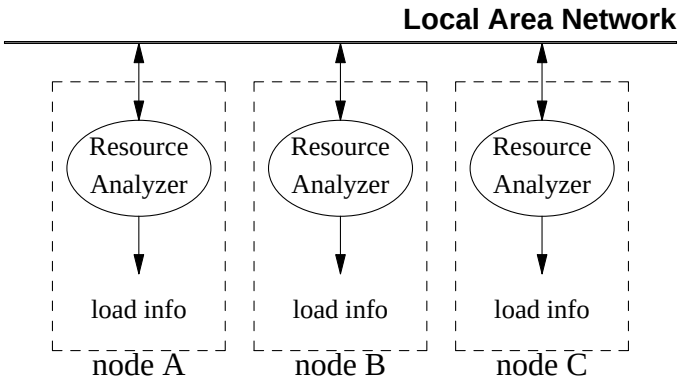

Abb. 28: Das Netzwerk

Untersuchungen in [Flac93b] bzgl. dem Übertragungszeitverhalten bei Ethernet ergaben, daß das dabei zur Anwendung kommende CSMA/CD-Netzwerk-Zugriffsverfahren günstig bei zeitlich und mengenmäßig stark schwankenden sowie unregelmäßigen Netzwerk-Verkehrsverhalten innerhalb einer niedrigen Netzwerklast (< 60%) ist. Weiterhin wurde nur eine geringe Kollisionsrate festgestellt, so daß ein CSMA/CD-System selbst bei hoher Netzlast nicht zusammenbricht. Dadurch steht mit Ethernet ein Kommunikationsmedium zur Verfügung, das die Kommunikation unabhängiger Geräte und Systeme innerhalb eines territorial begrenzten Bereiches mit hoher Übertragungsgeschwindigkeit gewährleistet.

Arten der Kommunikation

Die Kommunikation zwischen den Rechnerknoten bzw. zwischen den Prozessen erfolgt über ein lokales Netz (Ethernet), auf das die einzelnen Stationen Zugriff haben. Jeder an das Kommunikationsmedium angeschlossene Rechnerknoten liest alle im Netzwerk übertragenen Pakete bzw. Nachrichten. Diese Eigenschaft ist die Voraussetzung dafür, daß eine Nachricht von mehr als einem Knoten empfangen werden kann. Jeder Rechner vergleicht die Zieladresse mit der eigenen Adresse. Liegt eine Übereinstimmung der Adressen vor, übernimmt der Rechner als Empfänger die vollständige Nachricht. Ein Sender muß daher den oder die Empfänger als Ziel seiner Nachricht spezifizieren. Ent-

sprechend der dabei gewählten Anzahl von Empfängern läßt sich die Kommunikation unterteilen in:

- ***Unicast***
- ***Multicast***
- ***Broadcast***

Unicast

Die überwiegende Anzahl von Nachrichtenübertragungen können als Unicast eingestuft werden. Dabei wird eine Nachricht von einem einzelnen Sender an einen einzelnen Empfänger verschickt [Stev91]. Diese Art von Kommunikation wird auch als Punkt-zu-Punkt-Kommunikation bezeichnet. Eine Unicast-Nachricht, die für mehrere Empfänger bestimmt ist, muß an jeden Empfänger einzeln versendet werden (siehe Abb. 29).

Darstellung: 1 : 1 (Sender : Empfänger)

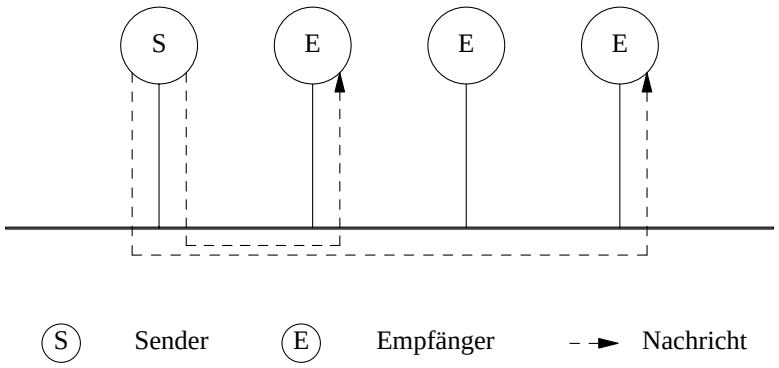

Abb. 29: Unicast-Nachrichtenversendung

Multicast

Eine Multicast-Nachricht wird von einem Sender an eine bestimmte Gruppe von Empfängern im Netz versendet. Voraussetzung dafür ist, daß ein Teil der möglichen Empfänger zu einer Multicast-Gruppe zusammen gefaßt wird. Diese Art der Kom-

munikation ermöglicht es, daß durch Versenden einer einzigen Nachricht alle Mitglieder der Gruppe erreicht werden (siehe Abb. 30). Der Sender der Nachricht besitzt dabei keine Kenntnis darüber, wie viele Empfänger es in der Gruppe gibt bzw. wer diese Empfänger sind [Tane81]. Diese Form der Kommunikation wird realisiert, indem für jede Gruppe von Empfängern eine Multicast-Adresse festgelegt wird. Nachrichten, die mit dieser Adresse versendet werden (bei Ethernet: höchstes Adreßbit = 1), werden dann nur von den Gruppenmitgliedern empfangen. Dabei wird die Nachricht aus dem Strom der Mitteilungen an Hand der Adresse herausgefiltert [Aute90].

Darstellung: 1 : K (Sender : Empfänger; K < N, N = Anzahl aller Knoten im Netz)

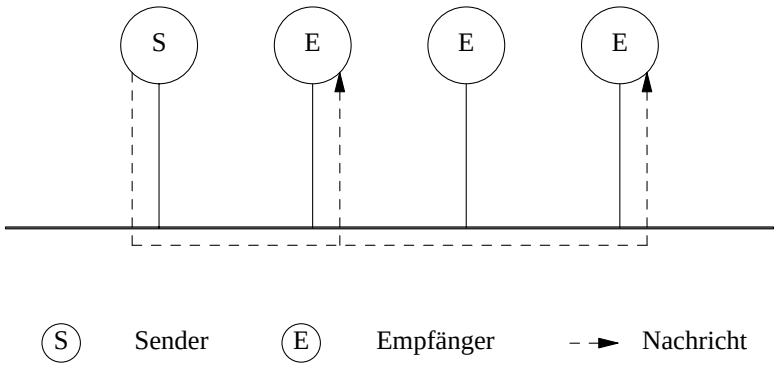

Abb. 30: Multicast-Nachrichtenversendung

Broadcast

Mittels Broadcasting wird von einem Sender eine Nachricht an alle vorhandenen Empfängerknoten im Netz verschickt. Diese Form der Kommunikation wird auch als Rundsendung bezeichnet [Tane81]. Die Zieladresse einer Broadcast-Nachricht hat eine besondere Form, die sie von allen anderen Adressen unterscheidet (bei Ethernet: alle Adreßbits sind auf 1 gesetzt). Durch das Versenden einer einzigen Nachricht werden alle im Netz vorhandenen Knoten erreicht. Voraussetzung aber ist, daß sich auf den Rechnerknoten ein aktiver Empfangsprozeß befindet (siehe Abb. 31). Bei einem broadcastfähigen LAN wie dem Ethernet sind die Übertragungskosten einer Broadcast-Nachricht ähnlich denen einer Unicast-Nachricht [Aute90]. Als Einschrän-

kung für die möglichen Empfänger einer Broadcast-Nachricht muß berücksichtigt werden, daß diese Form von Nachrichten nicht in andere Netze bzw. Teilnetze von Gateways weitergeleitet werden [Stev91].

Darstellung: 1 : N (Sender : Empfänger; N = Anzahl aller Knoten im Netz bzw. Teilnetz)

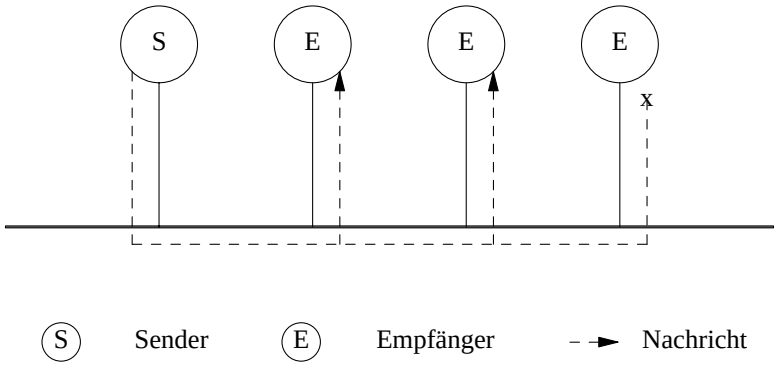

Abb. 31: Broadcast-Nachrichtenversendung

Die Vorteile von Multicast und Broadcast lassen sich wie folgt zusammenfassen:

- Nur eine physische Nachricht ist erforderlich, um an mehrere Empfänger eine Information zu versenden.
- Auf Grund nur einer physischen Nachricht wird die Netzwerkbelastung reduziert im Vergleich zu mehreren Einzelnachrichten, die notwendig sind, um mehrere Empfänger mittels Unicast zu erreichen.
- Es erfolgt eine Reduzierung der Anzahl von notwendigen Sendeprozessen, da insgesamt weniger Nachrichten zu verschicken sind.
- Die Möglichkeit von parallelen Ausführungen in einem verteilten System wird erhöht, da die Empfänger die Nachricht gleichzeitig und schneller erhalten.

Domäne

Eine Gruppe von Rechnerknoten, auf die der Lastinformationsaustausch von Rechnern beschränkt bleibt, wird als Domäne bezeichnet. Um eine Domäne zu bilden, ist es er-

forderlich, die Rechner des Netzwerks in Gruppen zu zerlegen. Bei dieser Unterteilung wird sich an den bestehenden Teilnetzen in der Netzwerktopologie orientiert. Teilnetze, die aus einer großen Anzahl von Rechnerknoten bestehen, sind in mehrere kleine Gruppen zu untergliedern (siehe Abb. 32). Die Verbindung zwischen zwei Domänen wird durch einen Gatewayknoten (siehe nachfolgenden Abschnitt) gewährleistet.

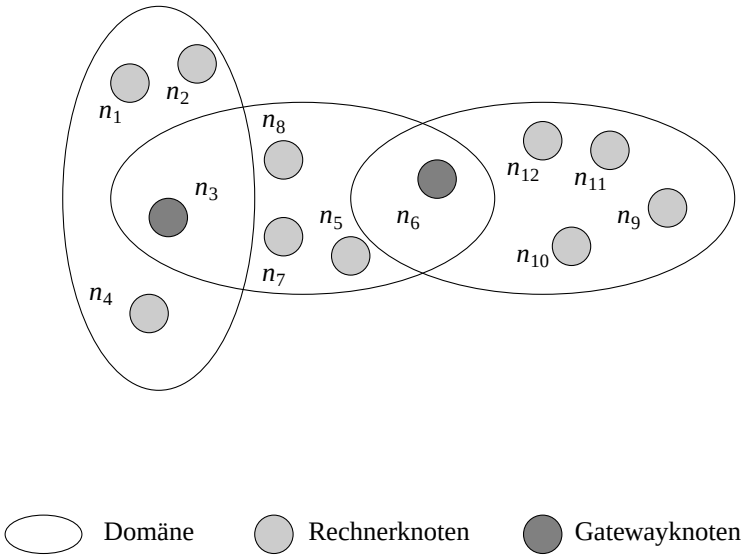

Abb. 32: Domänen und Gateways im Rechnernetz [Flac93a]

Gateway

Zur Realisierung des Lastausgleichs muß ein Rechnerknoten auch auf die Informationen über die Belastung von Rechnern außerhalb der eigenen Domäne zugreifen können. Dazu werden die Eigenschaften von Gatewayknoten genutzt [Come92]. Dies sind entsprechend ausgezeichnete Knoten, die die Schnittstelle zwischen zwei Teilnetzen bzw. die Verbindung zwischen zwei Domänen darstellen (siehe Abb. 32). Über einen Gatewayknoten können Nachrichten von einem Rechnerknoten der einen Domäne an Rechner der anderen Domäne weiter gegeben werden. Ein Gateway besitzt die Lastinformationen aus beiden Domänen, die er miteinander verbindet. Bei der Wahl des Gatewayknotens einer Domäne wird sich ebenfalls nach der gegebenen Struktur des Netzwerks gerichtet. Gatewayrechner zwischen Teilnetzen werden auch Gatewayknoten

zwischen zwei Domänen. Erfolgt eine Unterteilung großer Subnetze in mehrere Domänen, so ist die Leistungsfähigkeit des zufällig als Gateway ausgewählten Knotens zu berücksichtigen.

Kommunikationsprotokolle

Von der Wahl der Kommunikationsprotokolle hängt ab, wie sicher die Kommunikation realisiert werden kann. Durch die Kommunikationsprotokolle im OSI-7-Schichtenmodell (Open Systems Interconnection - OSI) wird ein virtueller Datenfluß zwischen den gleichen Schichten der beiden am Nachrichtenaustausch beteiligten Rechner definiert (siehe Abb. 33).

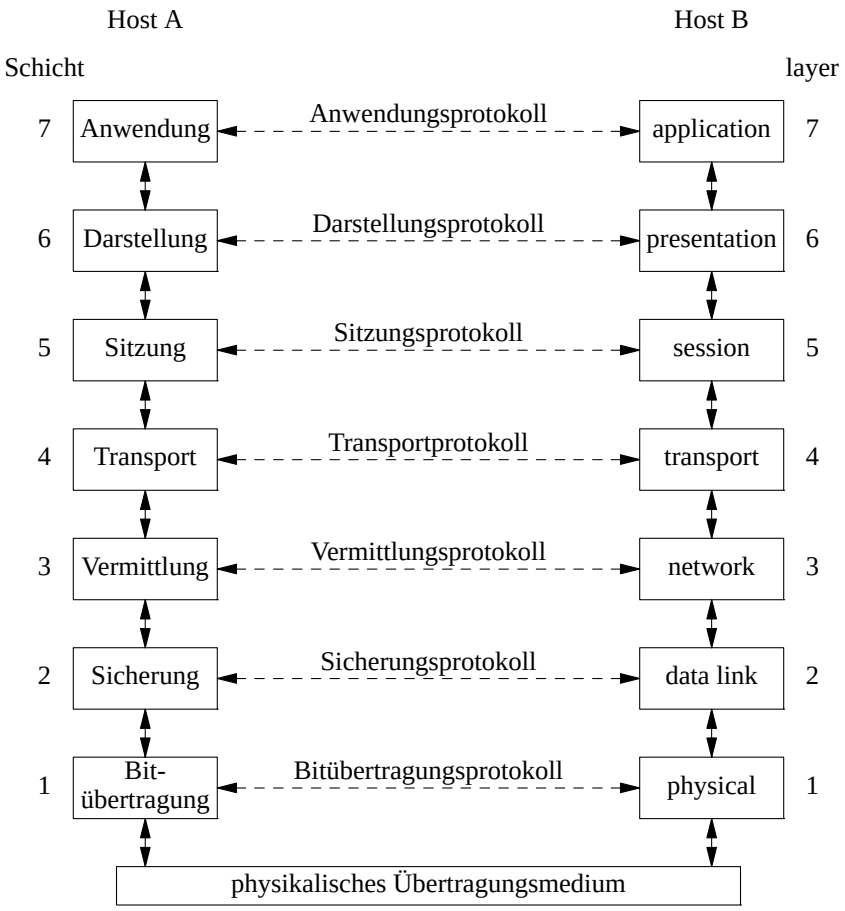

Abb. 33: Das OSI-7-Schichtenmodell [Tane81]

Die Schichten 1 - 4 können als transportdienstleistende Schichten bezeichnet werden. Die Schichten 5 - 7 bilden die transportdienstnutzenden Schichten. Der Transportschicht (Schicht 4) als Bindeglied zwischen nachrichtenübertragungsorientierten und anwendungsorientierten Instanzen kommt eine große Bedeutung zu. Sie bildet den Kern der gesamten Protokollhierarchie [Tane81]. Die Transportschicht ist im Schichtenmodell die letzte Instanz, die sich mit der Qualitätssicherung des Nachrichtentransports bezogen auf netz- und übertragungsspezifische Fehler befaßt [Kauf91]. Damit hat die Transportschicht bzw. das zugehörige Transportprotokoll entscheidenden Einfluß auf die Zuverlässigkeit und Effektivität des zu realisierenden Datentransfers zwischen Quell- und Zielrechner. Gleichzeitig stellt die Transportschicht für die oberen Schichten (5 - 7) einen netzwerkunabhängigen Transportdienst bereit [Chyl88].

Aufbauend auf dem OSI-7-Schichtenmodell sind für die Transportschicht fünf OSI-Transportprotokolle (TP0 - TP4) bereitgestellt. Diese verbindungsorientierten Protokolle bauen entsprechend dem Schichtenmodell auf den Diensten der Vermittlungsschicht auf. Durch die Vermittlungsschicht können drei verschiedene Typen von Netzdiensten bereitgestellt werden [Stev91]:

Typ A: zuverlässiger, fehlerfreier Dienst, da alle Fehler abgefangen werden

Typ B: zuverlässiger Dienst, bei dem auftretende Fehler erkannt werden

Typ C: unzuverlässiger Dienst

Entsprechend dem zugrundeliegenden Diensttyp der Vermittlungsschicht unterscheiden sich die fünf Transportprotokolle in den Maßnahmen zur Gewährleistung der Kommunikationssicherheit (siehe Tab.). (Multiplexing = Fähigkeit, über eine einzelne Netzwerkverbindung zwei oder mehrere Transportverbindungen aufrechtzuhalten)

Ein weiteres OSI - Protokoll der Transportschicht, das auf dem unzuverlässigen Dienst der Vermittlungsschicht aufbaut, liefert einen verbindungsunabhängigen Service. Dieses Transportprotokoll stellt nicht sicher, daß die Datenpakete in der selben Reihenfolge empfangen werden, wie sie versendet wurden.

Eine ebenfalls große Verbreitung bei der Rechnerkommunikation haben die von ARPANET entwickelten Transportprotokolle TCP (Transmission Control Protocol) [Post81a] und UDP (User Datagram Protocol) [Post80].

TP	Netzdiensttyp	Fehlererkennung	Fehlerbeseitigung	Multiplexing
0	A	nein	nein	nein
1	B	nein	ja	nein
2	A	nein	nein	ja
3	B	nein	ja	ja
4	C	ja	ja	ja

Beide Protokolle bauen auf den Diensten des verbindungsunabhängigen Vermittlungsprotokolls IP (Internet Protocol) [Post81b] auf. IP ist ein unzuverlässiges Protokoll, das Datenpakete in Form von Datagrammen ohne Beibehaltung der Sendereihenfolge und ohne Empfangsbestätigung an andere Rechnerknoten über das Netzwerk transportiert.

TCP ist ein verbindungsorientiertes Transportprotokoll, das einen zuverlässigen Transportservice zwischen zwei Kommunikationsendpunkten liefert. Ausdruck der Zuverlässigkeit ist:

- Für jede empfangene Nachrichteneinheit (hier Segment) wird an den Sender eine Empfangsbestätigung verschickt.
- Läuft beim Sender nach dem Verschicken einer Nachricht ein definiertes Zeitintervall (Timeout) ab, ohne das eine Empfangsbestätigung eingetroffen ist, wird die Nachricht nochmals versendet.
- Die empfangenen Daten sind in der Reihenfolge ihrer Versendung bzw. werden richtig sortiert.
- Es erfolgt ein Ausschluß von doppelt empfangenen Nachrichten.
- Durch eine Flußkontrolle wird sichergestellt, daß der Sender die Daten nur so schnell versendet, daß der Empfänger die ankommenden Nachrichten ohne Datenverlust verarbeiten kann.

Diese Sicherheiten werden aber durch einen zusätzlichen Overhead erreicht, der zusammen mit der Nachrichtenübertragung in Form von Byte-Strömen in einer langsamen und kostenintensiven Kommunikation resultiert.

UDP als ein verbindungsunabhängiges Transportprotokoll liefert hingegen einen schnellen und effektiven Datagramm-Dienst. Dieses Transportprotokoll gewährleistet aber nicht die Zuverlässigkeit der Nachrichtenübertragung wie TCP.

Entsprechend den verschiedenen spezifischen Anwendungsanforderungen stellt ARPANET ein weiteres Transportprotokoll zur Verfügung. RDP (Reliable Data Protocol) [Velt84] ist ein Transportprotokoll, das für eine verbindungsorientierte, zuverlässige und nachrichtenorientierte Datenübertragung entwickelt wurde. Im Vergleich zu TCP ist RDP ein weniger komplexes und zugleich effizienteres Protokoll.

XNS (Xerox Network System) [Slom89] entwickelte Kommunikationsprotokolle konkret für den Einsatz in lokalen Netzen und speziell für Ethernet. Das IDP (Internet Datagram Protocol) ist ein verbindungsloses, nicht zuverlässiges Protokoll für den Transfer von Datagrammen auf der Vermittlungsschicht. IDP stellt somit den Datagrammübertragungsdienst für die Transportprotokolle SPP (Sequenced Packet Protocol) und PEX (Packet Exchange Protocol) bereit.

SPP liefert eine zuverlässige (Flußkontrolle, Datenverlustsicherung) und verbindungsorientierte Transportverbindung. Im Vergleich zu TCP, das nur byteorientierte Datenströme überträgt, bietet SPP drei Formen des Datentransfers als Schnittstellen für den Nutzerprozeß an [Stev91]:

- byteorientierte Datenströme
- paketorientierte Datenströme
- zuverlässige Datagramm-Übertragung (es besteht keine volle Garantie für die Einhaltung der Reihenfolge der Datagramme beim Transfer)

PEX ist ein verbindungsunabhängiges, unzuverlässiges Datagramm-Transportprotokoll. Im Gegensatz zu UDP besitzt PEX die Möglichkeit einer Wiederholung der Nachrichtenübertragung nach Ablauf eines Timeouts.

In der Tabelle sind die vorgestellten Vermittlungs- und Transportprotokolle zusammenfassend dargestellt.

Datagramme sind Pakete von Informationen, die für einen, mehrere oder alle Rechnerknoten als Ziel bestimmt sein können und die allein und unabhängig von anderen

Protokoll	OSI-Netzdiensttyp	verbindungsorient.	zuverlässig	sequentiell	nicht sequentiell	Bytestrom	Datagramm	Verfügbarkeit
IP	C	nein	nein	nein	ja	nein	ja	Standard in
TCP	C	ja	ja	ja	nein	ja	nein	SVR4, 4.3BSD
UDP	C	nein	nein	nein	ja	nein	ja	u.a.
RDP	C	ja	ja	optional	ja	ja	ja	optional
IDP	C	nein	nein	nein	ja	nein	ja	optional in
SPP	C	ja	ja	ja	ja	ja	ja	SVR4, 4.3BSD
PEX	C	nein	nein	nein	ja	nein	ja	u.a.
TP0	A	ja	ja	ja	nein	ja	nein	optional
TP1	B	ja	ja	ja	nein	ja	nein	durch
TP2	A	ja	ja	ja	nein	ja	nein	ISODE-
TP3	B	ja	ja	ja	nein	ja	nein	Paket
TP4	C	ja	ja	ja	nein	ja	nein	
*	C	nein	nein	nein	ja	nein	ja	

Datagrammen übertragen werden. Daher muß jedes Datagramm die für die Übertragung notwendigen Informationen, z.B. die Zieladresse, beinhalten. Die Datagramme können dabei entsprechend der Anzahl von Zielknoten Unicast-, Multicast- oder Broadcast-Zieladressen besitzen.

5.4.2. Lastinformationsaustausch

Im Rahmen des Lastinformationsaustausches sollen von jedem Rechnerknoten Kenntnisse über den aktuellen lokalen Belastungszustand an andere Rechner, die an der Lastbalancierung beteiligt sind, versendet werden. Ein Rechnerknoten kann in die Lastbalancierung einbezogen werden, wenn auf dem Knoten ein Lastdämon (siehe Abs. 5.4.6.) aktiviert wurde, der die notwendigen Lastmessungen und Aufgaben im Zusammenhang mit dem Lastinformationsmanagement realisiert. Auskunft über die Belastung eines Rechnerknotens gibt dessen aktueller Lastdeskriptor, der während der Lastbestimmung und der anschließenden Aufbereitung der Lastinformationen ermittelt wurde.

Anforderungen an die Kommunikation

Ein aktueller Lastdeskriptor wird beim Lastinformationsaustausch vom lokalen Rechnerknoten an die an der Lastbalancierung beteiligten Rechner verschickt. Durch

* verbindungsunabhängiges OSI-Transportprotokoll

das effiziente Versenden einer einzigen Nachricht über die aktuelle Lastsituation des lokalen Rechnerknotens ist die Gruppe von Rechnern zu informieren, auf denen ebenfalls ein Lastdämon läuft und die daher in den Lastinformationsaustausch einzubeziehen sind. Die erforderliche Art der Kommunikation beim Lastinformationsaustausch entspricht somit dem **Multicasting**.

Die lokalen und die während des Lastinformationsaustausch empfangenen Lastinformationen bilden die Basis für die zu treffenden Lastbalancierungsentscheidungen. Um dieser Bedeutung der Kenntnisse bzgl. der Rechnerbelastung gerecht zu werden, ist eine **sichere Kommunikation** anzustreben. Grundlage dafür ist ein entsprechendes Kommunikationsprotokoll, nach dessen Regeln und Definitionen die Kommunikation zwischen den Kommunikationspartnern erfolgt.

Außerdem ist eine schnelle Übertragung der Nachrichten zwischen den Rechnerknoten notwendig, da aktuelle Lastinformationen für eine effektive Lastbalancierung erforderlich sind. Aus diesem Grund sind die Lastinformationen mittels eines schnellen **Datagramm-Dienstes** an Stelle von langsameren sequentiellen Byte-Strömen zu übertragen.

Realisierung des Lastinformationsaustausches

Die Entscheidungen hinsichtlich der Realisierung des Lastinformationsaustausches wurden unter Abwägung der vorhandenen Grundlagen und Möglichkeiten der Kommunikation (siehe Abs. 5.4.1.) gegenüber den Anforderungen an die Kommunikation getroffen.

Das verwendete Transportprotokoll

Auf Grund der geforderten Multicast-Kommunikation ist nur ein verbindungsloser Transportdienst sinnvoll und realisierbar. Um einen einzelnen Lastdeskriptor an mehrere Ziele zu versenden, wäre bei einem verbindungsorientierten Vorgehen ein enormer Aufwand notwendig. Für die Nachrichtenübermittlung vom Quellrechner zu jedem einzelnen Zielrechner müßte eine Punkt-zu-Punkt-Verbindung (Unicast) aufgebaut, eine Nachricht generiert und übertragen sowie die bestehende Verbindung wieder abgebaut werden. Dieses Vorgehen benötigt im Vergleich zur Versendung mittels Broadcast bzw.

Multicast mehr Zeit und beansprucht die Rechnerressourcen und das Netzwerk stärker. Bei einer verbindungslosen Kommunikation werden Datagramme über das Netzwerk versendet, ohne das ein Overhead zum Verbindungsaufbau und -abbau verursacht wird.

Zum Versenden der Lastinformationen an die an der Lastbalancierung beteiligten Rechnerknoten wird daher das **UDP-Protokoll** [Sant92] als verbindungsloses Transportprotokoll genutzt. Dieses nicht zuverlässige Datagramm-Dienst-Protokoll kann den Anforderungen der Kommunikation unter Berücksichtigung der in der HE$_A$D - Systemumgebung zur Verfügung stehenden Transportprotokollen am ehesten gerecht werden. Es wird häufig genutzt, um von einem Knoten in kurzen Intervallen Informationen an mehrere Zielrechner zu versenden [Wash93]. Ein möglicher Datenverlust wird dabei durch das Versenden einer aktuellen Nachricht nach Ablauf eines festen Zeitintervalls ausgeglichen. Die Einfachheit des Protokolls und der Verzicht auf Sicherungsmaßnahmen bei der Nachrichtenübertragung bewirken, daß UDP effizient arbeitet (geringer Overhead) und für Anwendungen geeignet ist, die eine kurze Übertragungszeit erfordern. Dabei wird das UDP-Protokoll häufig gleichzeitig mit einem schnellen und sicheren Übertragungsmedium wie z.B. LAN eingesetzt [Sant92]. Um eine annähernd sichere Kommunikation (Reaktion auf Nachrichtenverlust, Fehlererkennung, Duplikaterkennung, Flußkontrolle, Reihenfolge der Nachrichten) trotz des Einsatzes von UDP zu gewährleisten, sind eigene Maßnahmen und Sicherungsvorkehrungen zu treffen.

Von UDP wird nicht garantiert, daß die zu übertragenen Datagramme in der gleichen Reihenfolge ankommen, wie sie versendet wurden. Jeder verschickte Lastdeskriptor enthält einen "Zeitstempel", der den Zeitpunkt der Lastbestimmung angibt. An Hand dieses Zeitstempels kann eine korrekte Reihenfolge der empfangenen Lastdeskriptoren garantiert werden. Gleichzeitig kann die Aktualität der im Lastdeskriptor enthaltenen Belastungsinformationen abgeleitet werden.

Bei einem UDP-Protokoll gibt es keine Sicherheit bzw. keine Empfangsbestätigung dafür, daß die versendete Nachricht den Zielknoten erreicht hat. Um den Schaden aus einem eventuellen Verlust eines versendeten Lastdeskriptors gering zu halten, wird sichergestellt, daß spätestens nach Ablauf eines festgelegten Zeitintervalls (*freshtime*) erneut ein aktueller Lastdeskriptor verschickt wird. Dazu wird in regelmäßigen Zeitabständen das Alter der zuletzt versendeten Belastungsinformation überprüft. Liegt das

Aussenden der Nachricht bzgl. der lokalen Rechnerknotenbelastung bereits den definierten Zeitraum *freshtime* oder sogar länger zurück, wird ein aktueller Lastdeskriptor verschickt. Durch dieses Vorgehen wird versucht, einen möglichen Lastinformationsverlust zu kompensieren.

Duplikate von möglicherweise doppelt gesendeten bzw. empfangenen Lastdeskriptoren werden verworfen, indem ein Doppelteintrag des gleichen Lastindex im Lastvektor (siehe Abs. 5.4.3.) verhindert wird.

Die Art der Kommunikation

Da der Lastinformationsaustausch auf die an der Lastbalancierung beteiligten Rechnerknoten der Domäne beschränkt werden soll (siehe folgenden Unterabschnitt), können die Lastinformationen auch mittels des einfacher zu realisierenden **Broadcasting** versendet werden. Dabei wird die standardisierte Broadcastadresse genutzt. Damit entfällt die sonst notwendige Vereinbarung einer Gruppenadresse (Multicastadresse) für die Lastinformationsversendung. Broadcast verursacht wie Multicast neben der Versendung der Broadcast-Nachricht keinen zusätzlichen Netzwerkverkehr. Alle Rechner in der Domäne empfangen und verarbeiten diese Broadcast-Nachricht. Es wird erst auf der Ebene des Transportprotokolls festgestellt, ob ein Prozeß vorhanden ist, der die Zielportnummer der empfangenen Nachricht nutzt. Ist auf einem Rechnerknoten kein Lastdämon als Empfangsprozeß aktiv, wird die Nachricht verworfen. Somit verursacht die Broadcast-Nachricht auf einem Rechner ohne aktiven Lastdämon einen zusätzlichen Verarbeitungsaufwand. Da dieser Rechnerknoten aber nicht an der Lastbalancierung beteiligt ist, wird dessen Belastung als nicht ausschlaggebend betrachtet. Durch Broadcast als Art der Kommunikation bei der Lastinformationsversendung ist gewährleistet, daß alle Rechner der Domäne und selbst der lokale Rechnerknoten den nur einmal versendeten aktuellen Lastdeskriptor empfangen. Jeder lokale Rechnerknoten hat somit eine globale Sicht auf die momentane Belastung der Rechner innerhalb der Domäne.

Die Lastinformationsversendung bzgl. einer Domäne erfolgt mittels Broadcasting. Auf das gesamte Systems bezogen, wird der Lastinformationsaustausch als Multicasting realisiert (Multicast-Gruppe = Rechnerknoten der Domäne).

Minimierung des Kommunikationsoverhead beim Lastinformationsaustausch

Unter Beachtung der Effektivität der Lastbalancierung sind stets der erforderliche Aufwand und der mögliche Nutzen abzuwägen. Im Zusammenhang mit dem Lastinformationsaustausch ist daher zu klären:

- Welche Anzahl von Rechnerknoten ist in den Lastinformationsaustausch einzubeziehen?
- Wann ist eine Benachrichtigung der am Lastausgleich beteiligten Rechnerknoten über eine lokale Belastungsänderung erforderlich?

Umfang des Lastinformationsaustausches

Über das bestehende Netzwerk sollen im Zusammenhang mit dem Lastinformationsaustausch Kenntnisse über die Rechnerknotenbelastung zwischen den Lastkomponenten der Knoten ausgetauscht werden. Durch jede Übertragung wird das Kommunikationsmedium beansprucht. Mit zunehmender Anzahl an Rechnerknoten steigt der Netzwerkverkehr und somit dessen Belastung an. Daher ist der Aufwand für den Informationsaustausch und für die -verwaltung zu begrenzen. Zur Wahrung der Effektivität und unter Berücksichtigung des mit der Lastbalancierung angestrebten Optimierungsziels kann folgende Einschränkung getroffen werden. Jeder Rechnerknoten muß nicht Kenntnisse über alle anderen Knoten im Netz besitzen. Es wird als ausreichend angesehen, wenn der Informationsaustausch auf die Rechnerknoten der Domäne begrenzt wird. Der Lastinformationsaustausch ist daher mittels Broadcasting so organisiert, daß alle Rechnerknoten umfassende Kenntnisse über die zur eigenen Domäne gehörenden Knoten verfügen. Durch diese Beschränkung des Rahmen des Lastinformationsaustausches wird der Kommunikationsaufwand begrenzt, zumal innerhalb einer Domäne die Kommunikationskosten gering sind.

Anlaß für eine Lastinformationsversendung

Reaktionen der Lastkomponente auf kleinere Belastungsänderungen erscheinen als uneffektiv bei Abwägung des notwendigen Aufwands und des erzielbaren Nutzens [Ludw93]. Dies hat zur Folge, daß die Nachbarknoten über kleinere Belastungswechsel nicht benachrichtigt werden müssen. Außerdem ist der Informationsaustausch auf ein

erforderliches Maß zu beschränken, da durch jede Nachrichtenübertragung der Kommunikationskanal belastet wird. Demgegenüber steht die Forderung, daß jeder Knoten über ausreichend aktuelle Informationen zur Belastung der Nachbarknoten bei der Entscheidungsfindung im Rahmen der Lastbalancierung verfügen muß. Basieren die Entscheidungen auf unaktuellen Informationen, können falsche systembelastende Prozeßverlagerungen verursacht werden. Es ist erforderlich, einen Kompromiß zwischen den beiden gegensätzlichen Anforderungen zu finden.

Notwendigkeit des Informationsaustauschs < – > Aufwand für Informationsübertragung

Eine Grenze ist festzulegen, ab wann eine Laständerung als so bedeutend angesehen wird, daß darüber die anderen beteiligten Knoten zu verständigen sind. Über die Stärke der Lastschwankungen im zurückliegenden Berechnungsintervall gibt der Anstieg der Regressionsgeraden Auskunft (großer Anstieg → hoher Belastungswechsel; kleiner Anstieg → geringe Laständerung bzw. viele kleine Schwankungen)(siehe Abb. 34).

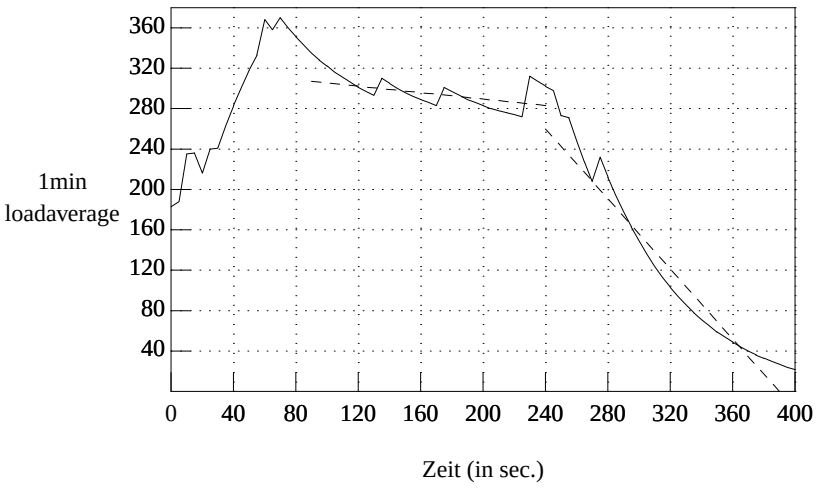

Abb. 34: Anstieg der Regressionsgeraden

Unter Ausnutzung dieses Zusammenhanges werden zwei Schwellwerte bzgl. des statistisch abgesicherten Anstiegs der Regressionsgeraden fixiert. Ein positiver Schwellwert bestimmt, wie stark die Gerade und somit die Last tendenziell ansteigen darf, ohne daß darüber die anderen Rechnerknoten informiert werden müssen. Ähnlich wird der negative Schwellwert festgelegt, der den Spielraum für eine fallende Gerade bzw.

absinkende Last abgrenzt. Wird von einem Lastparameter des Lastdeskriptors einer der Schwellwerte übertreten, so erfolgt die Versendung des gesamten Lastdeskriptors an die anderen Netzknoten der Domäne mittels Broadcasting.

Damit die Nachbarknoten Kenntnis darüber haben, wie stark ein neu hinzugekommener Rechner und somit in die Lastbalancierung einzubeziehender Knoten, belastet ist, wird nach der ersten Berechnung des Regressionskoeffizienten grundsätzlich der Lastdeskriptor verschickt. Dies erfolgt unabhängig davon, ob ein Schwellwert überschritten wurde. Ereignen sich auf dem lokalen Knoten nur geringe Laständerungen, so werden keine Nachrichten über einen Belastungswechsel versendet (eine Schwellwertüberschreitung liegt nicht vor). Um den anderen Rechnerknoten aber trotzdem mitzuteilen, daß der lokale Knoten weiterhin bei der Lastbalancierung zu berücksichtigen ist, wird nach einem definierten Zeitintervall (*freshtime*) seit der letzten Lastinformationsversendung ein aktueller Lastdeskriptor verschickt.

5.4.3. Verwaltung der Informationen

Die empfangenen Lastinformationen sowohl des eigenen als auch der anderen Knoten der Domäne müssen von jedem Rechnerknoten verwaltet werden. Zu diesem Zweck verfügt jeder Knoten über einen Lastvektor. Dieser Vektor ist als eine doppelt verkettete Liste realisiert. Die Einträge im Lastvektor sind nach aufsteigender Belastung sortiert, wobei der 1min-Lastdurchschnitt das entscheidende Ordnungskriterium ist. Empfängt ein Knoten einen Lastdeskriptor, wird die Liste aktualisiert. Dabei wird der empfangene Lastdeskriptor eines Knotens entsprechend dem Sortierkriterium in die Liste eingeordnet. Ist in der Liste bereits ein Eintrag zu dem Knoten vorhanden, so wird dieser nun unaktuelle Lastdeskriptor entfernt. Durch diese Vorgehensweise ist sichergestellt, daß jeder Knoten nur durch seinen zuletzt gesendeten bzw. aktuellsten Lastdeskriptor in der Liste repräsentiert wird. In regelmäßigen Zeitintervallen (*refresh*) erfolgt eine Überprüfung aller Listeneinträge, um festzustellen, welche Knoten noch für die Lastbalancierung zu berücksichtigen sind. Dabei wird die Liste nach Einträgen durchsucht, die älter als ein ganzzahliges Vielfaches der *freshtime* sind. Im Normalfall verschickt jeder Knoten nach Ablauf des Zeitintervalls *freshtime* seit der letzten Versendung von Lastinformationen einen aktuellen Lastdeskriptor. Wird ein älterer Listeneintrag gefunden, erfolgt die Streichung aus der Liste mit der Annahme, daß der

Knoten nicht mehr für die Lastbalancierung zur Verfügung steht.

5.4.4. Zugriff auf die Informationen des Lastvektors

Die im Lastvektor enthaltenen Informationen über die Belastungszustände des eigenen und der Nachbarknoten der Domäne bilden eine Grundlage für die Lastbalancierungsentscheidungen. Um diese Informationen in die Entscheidungsfindung einbeziehen zu können, ist der Zugriff auf diese Daten notwendig. Durch folgende drei Anfragetypen wird die Nutzung der Informationen aus dem Lastvektor ermöglicht:

1. Anfrage nach dem Lastdeskriptor eines Knotens der eigenen Domäne
2. Anfrage nach den Lastdeskriptoren aller Knoten der eigenen Domäne
3. Anfrage nach dem Lastdeskriptor eines Knotens aus einer anderen Domäne

Eine Anfrage wird grundsätzlich von einem Nutzerprozeß generiert und an die lokale Lastkomponente auf dem Rechnerknoten versendet.

Anforderungen an die Kommunikation

Bei der Kommunikation im Rahmen der Anfragebearbeitung sind grundsätzlich nur Punkt-zu-Punkt-Kommunikationen erforderlich. Sowohl die Nachricht zur Anfrageübermittlung als auch die Ergebnisnachrichten werden nur zwischen dem konkreten Nutzerprozeß und der bearbeitenden Lastkomponente verschickt. Die Kommunikation im Fall der Weiterleitung der Anfrage von der Lastkomponente des lokalen Rechnerknotens an die Lastkomponente eines anderen Rechners erfolgt ebenfalls nur zwischen zwei Punkten (ein Sendeprozeß - ein Empfangsprozeß). Daher wird bei der Anfragebearbeitung eine **Unicast-Kommunikation** angestrebt.

Bei der Bearbeitung von Anfragen ist ebenfalls eine **sichere Kommunikation** durch ein entsprechendes Transportprotokoll zu garantieren. Durch den schnellen Zugriff auf die Lastdeskriptoren des Lastvektors mittels eines **Datagramm-Dienstes** wird die Informationsbasis für die Lastbalancierungsentscheidungen geliefert. Dabei ist zu beachten, daß Entscheidungen, die auf falschen oder unaktuellen Lastinformationen beruhen, einen negativen Einfluß auf die Systemleistung verursachen können.

Das verwendete Transportprotokoll

Eine sichere und nachrichtenorientierte Übertragung von Daten bietet nur das verbindungsorientierte RDP-Transportprotokoll. Da aber RDP in der HE$_A$D - Systemumgebung nicht verfügbar ist, wird wiederum das **UDP-Transportprotokoll** als nachrichtenorientiertes Transportprotokoll gewählt. Da dieses Datagramm-Dienst-Protokoll aber unzuverlässig arbeitet, sind im Zusammenhang mit der Kommunikation bei der Bearbeitung von Anfragen zum Zugriff auf die im Lastvektor enthaltenen Informationen einige Funktionen zur Verbesserung der Sicherheit bei der Kommunikation realisiert.

Nach dem Versenden der Anfrage an die Lastkomponente geht der Nutzerprozeß in Empfangsbereitschaft und wartet ein definiertes Zeitintervall auf den Erhalt einer Antwort. Empfängt der Nutzerprozeß innerhalb des Zeitintervalls keine Antwort auf die Anfrage, erfolgt ein Timeout und die Anfragebearbeitung wird beendet. Dies erfolgt in der Annahme, daß die Anfragenachricht (siehe Abs. 5.4.5.) bei der Lastkomponente nicht angekommen oder daß die Antwortnachricht verloren gegangen ist. Der Nutzer kann auf den Nachrichtenverlust reagieren, indem die Anfrage nochmals generiert und versendet wird.

Jeder Anfrage wird während der Generierung ein eindeutiger Identifikator zugewiesen, der bei allen Nachrichten bzgl. der Anfragebearbeitung mit übertragen wird. Dadurch kann eine Duplikaterkennung einer nicht vom Nutzer verursachten doppelten Anfrageversendung gewährleistet werden (Verursacher = Protokoll unterhalb der Transportschicht).

Die Anfragebearbeitung wird von der Lastkomponente mit dem Versenden einer "End of answer"- Nachricht (siehe Abs. 5.4.5.) beendet. In dieser Nachricht wird gleichzeitig die Anzahl der gefundenen und an den Sender der Anfrage verschickten Antworten übermittelt. Durch den Vergleich der Anzahl der tatsächlich empfangenen Antworten mit der Anzahl der ermittelten Antworten in der "End of answer"- Nachricht kann festgestellt werden, ob bei der Antwortübermittlung ein Nachrichtenverlust aufgetreten ist. Mit einer erneuten Anfrageversendung kann der Nutzer auf diesen Nachrichtenverlust reagieren.

Die Art der Kommunikation

Die in Abschnitt 5.4.5. detailliert dargestellten Nachrichten im Zusammenhang mit der Anfragebearbeitung werden entsprechend den Anforderungen an die dazu erforderliche Kommunikation mittels **Unicasting** realisiert.

Die Bearbeitung der Anfragen kann entsprechend dem Ausführungsort der damit verbundenen Operationen in **lokale** und **entfernte** Bearbeitung unterteilt werden:

Lokale Anfragebearbeitung

Da der Lastinformationsaustausch durch Broadcasting auf die Domäne beschränkt ist, verfügt jeder Rechnerknoten über die Lastdeskriptoren aller Knoten aus der eigenen Domäne (Voraussetzung: Lastkomponente auf den Rechnern ist aktiviert). Jeder Rechnerknoten besitzt daher die erforderlichen Informationen, um die ersten beiden Anfragetypen lokal beantworten zu können. Im Rahmen der Anfragebearbeitung ist nur eine Durchsuchung des lokalen Lastvektors notwendig, um den oder die geforderten Lastdeskriptoren zu finden. Die Ergebnisse der Anfragebearbeitung werden abschließend an den Nutzerprozeß, der die Anfrage generierte, übermittelt.

Entfernte Anfragebearbeitung

Mittels des 3. Anfragetypes wird der Zugriff auf Belastungsinformationen von Knoten außerhalb der eigenen Domäne realisiert. Dabei werden die Vorteile eines Gatewayknotens genutzt. Ein Gatewayknoten ist am Lastinformationsaustausch in beiden Domänen beteiligt, zwischen denen er die Schnittstelle bildet. Dieser Rechnerknoten verfügt daher über die Lastdeskriptoren aller Knoten aus beiden Teilnetzen. Wird an einen Nicht-Gatewayknoten (Rechnerknoten, der nicht die Eigenschaft eines Gatewayknoten besitzt) eine Anfrage des 3.Typs gerichtet, sendet dieser Knoten die Frage an einen Gatewayknoten der lokalen Domäne weiter. Es erfolgt eine entfernte Bearbeitung der Anfrage. Ist kein Gatewayknoten vorhanden, kann die Anfrage nicht beantwortet werden. Verfügt der Gatewayknoten über den gesuchten Lastdeskriptor in seinem Lastvektor, wird die Antwort an den Ausgangsknoten der Anfrage zurück gesendet. Ist die Beantwortung nicht möglich, versucht der Gatewayknoten die Anfrage an einen anderen Gatewayknoten, mit dem er in Broadcast-Verbindung steht, weiterzu-

leiten. Dieser Gatewayknoten bearbeitet dann die Anfrage. Eine Weitersendung der Anfrage erfolgt entweder bis eine Antwort gefunden wurde oder solange ein weiterer Gatewayknoten als Bearbeitungsknoten vorhanden ist. Die Anfrageabarbeitung ist rekursiv implementiert. Jeder Rechnerknoten sendet das Ergebnis der Anfragebearbeitung an den Rechner bzw. Prozeß, von dem er die Anfrage zur Bearbeitung empfangen hat. Grundvoraussetzung für die Realisierung dieser Anfrage ist, daß jeder Knoten darüber Kenntnis hat, ob er Gatewayknoten ist und welche Gatewayknoten in seiner Domäne vorhanden sind.

Die genaue Realisierung der Anfragen bzw. die erforderlichen Reaktionen der Knoten auf eine Anfrage werden im Zusammenhang mit der damit verbundenen Nachricht erläutert.

5.4.5. Nachrichten beim Informationsmanagement

Bei der Kommunikation der Knoten zum Zwecke des Lastinformationsaustausches sind neben dem eigentlichen Versenden des Lastdeskriptors weitere Nachrichten notwendig, um eine geordnete Kommunikation nach festen Regeln zu gewährleisten. Dabei gilt es, neben dem Austausch von Belastungsinformationen auch die Beantwortung von Anfragen sicherzustellen.

I’m alive - Nachricht des Nicht-Gatewayknotens

Jeder Nicht-Gatewayknoten, dessen Lastkomponente aktiviert wird, sendet einmalig als erstes diese Nachricht mittels Broadcasting an alle Nachbarknoten in der Domäne. Dadurch werden alle Empfänger veranlaßt, dem neu hinzugekommenen Knoten mit aktuellen Informationen über ihre lokale Belastung zu unterrichten. Rechnerknoten, die selbst noch keinen aktuellen Lastzustand (Regressionskoeffizient) ermittelt haben, senden erst im Anschluß an die Vorlage der ersten Ergebnisse der Regressionsanalyse. Die Versendung der Belastungsinformationen erfolgt mittels einer "New value" - Nachricht.

Empfängt ein Gatewayknoten eine "I’m alive" - Nachricht, versendet er außerdem eine "I’m gateway" - Nachricht.

Format: *A : <hostname>*

Bedeutung: *A* = Nachrichtentyp

< *hostname* > = Hostname des Nicht-Gatewayknotens, dessen Lastkomponente aktiviert wurde

I’m alive - Nachricht des Gatewayknotens

Ein Gatewayknoten versendet diese Nachricht mittels Broadcasting einmalig zu Beginn der Aktivierung der Lastkomponente. Die Empfänger verhalten sich wie bei der "I’m alive" - Nachricht des Nicht-Gatewayknotens. Zusätzlich speichern die Empfänger den Namen des Gatewayknotens (Sender der Nachricht) in einer Liste (= Gatewayknoten-Liste) ab. Damit verfügen die einzelnen Knoten in der Domäne über die Information, welcher Rechnerknoten ein Gateway ist und somit die Schnittstelle zu einem anderen Teilnetz darstellt. Es wird sichergestellt, daß in der Liste jeder Gatewayknoten nur einmal vorhanden ist.

Format: *a : <hostname>*

Bedeutung: *a* = Nachrichtentyp

< *hostname* > = Hostname des Gatewayknotens, dessen Lastkomponente aktiviert wurde

I’m gateway - Nachricht des Gatewayknotens

Empfängt ein Gatewayknoten eine "I’m alive"-Nachricht von einem beliebigen Knoten, kann daraus abgeleitet werden: Der Knoten, der die "I’m alive" - Nachricht sendete, ist erst nach dem Gatewayknoten aktiviert worden. Der Knoten hat somit die "I’m alive" - Nachricht des Gatewayknotens nicht empfangen können. Damit der Knoten dennoch die Information erhält, welcher Rechnerknoten ein Gateway in der Domäne ist, wird die "I’m gateway" - Nachricht mittels Broadcasting versendet. Die Empfänger der Nachricht speichern den Namen des Gateways in der Gatewayknoten-Liste ab, wobei Doppeleintragungen vermieden werden.

Format: *G : <hostname>*

Bedeutung: *G* = Nachrichtentyp

< *hostname* > = Hostname des Gatewayknotens

I’m dead - Nachricht

Wird die Lastkomponente eines Rechnerknotens deaktiviert, müssen darüber die anderen Knoten in der Domäne verständigt werden, da dieser Rechner nun nicht mehr für die Lastbalancierung verfügbar ist. Daher wird die "I’m dead" - Nachricht per Broadcast ausgesendet, wenn die Lastkomponente normal deaktiviert wird oder Signale eintreten, die die Beendigung der Arbeit der Lastkomponente zur Folge haben. Empfänger dieser Nachricht streichen den entsprechenden Knoteneintrag aus dem Lastvektor. Ist der Knoten in der Liste der Gatewayknoten enthalten, wird er darin ebenfalls gelöscht.

Format: *D : <hostname>*

Bedeutung: *D* = Nachrichtentyp

< *hostname* > = Hostname des Rechnerknotens, dessen Lastkomponente deaktiviert wurde

New value - Nachricht

Diese Nachricht wird mittels Broadcasting versendet um:

- andere Rechnerknoten über Belastungswechsel zu informieren
- anderen Rechnerknoten einen aktuellen Lastdeskriptor zu übermitteln

Inhalt der Nachricht ist ein Lastdeskriptor. Im Zusammenhang mit dem Lastinformationsaustausch fügen die Empfänger der Nachricht den Lastdeskriptor in den lokalen Lastvektor ein, wenn noch keine Belastungsinformationen über den Knoten (Sender der Nachricht) vorhanden sind. Ist bereits im Lastvektor ein Eintrag zum Lastzustand des Knotens enthalten, so erfolgt eine Aktualisierung des Rechnerknoteneintrages.

Format: *V : <host> : <time> : <load_min1.val> : <load_min1.b> : <load_min1.t> : <freemem.val> : <freemem.b> : <freemem.t> : <collisions.val> : <collisions.b> : <collisions.t> : <ipackets.val> : <ipackets.b> : <ipackets.t> : <opackets.val> : <opackets.b> : <opackets.t>*

Bedeutung: *V* = Nachrichtentyp

<host> : <time> : ... : <opackets.t> = Lastdeskriptor des Knotens

Request - Nachrichten

Diese Nachrichten beinhalten eine an die Lastkomponente des lokalen Rechnerknotens gestellte Anfrage. Das Anfrageformat ist über eine Schnittstellendefinition (*loadif.h*) festgelegt und bestimmt das Nachrichtenformat. Die Ergebnisübermittlung der Anfragebearbeitung wird für die drei möglichen Anfragetypen einheitlich realisiert. Wird eine Antwort auf die Anfrage gefunden, so erfolgt deren Ablage in einer Liste (Ergebnisliste). Der oder die Lastdeskriptoren in dieser Liste werden am Ende der Anfragebearbeitung zurück an den Ausgangspunkt der Anfrage mittels einer oder mehrerer "Answer" - Nachrichten verschickt. Um dem Sender der Anfrage das Ende der Anfragebehandlung mitzuteilen, wird abschließend eine "End of answer" - Nachricht versendet ("Answer" - und "End of answer" - Nachricht siehe nachfolgende Unterabschnitte). Konnte die Anfrage nicht beantwortet werden, wird nur die "End of answer" - Nachricht als Endekennzeichen verschickt. Um die an die Lastkomponente gerichteten Anfragen zu unterscheiden bzw. um eine korrekte Bearbeitung der Anfragen zu gewährleisten, wird jeder Frage ein Identifikator (Zeitpunkt der Anfragegenerierung) zugeordnet. Die Zuteilung dieses Kennzeichens zur Anfrage erfolgt einmalig im Nutzerprozeß, der die Anfrage generiert. Jeder Rechnerknoten, der eine Anfrage zur Bearbeitung an einen anderen Rechner weiterleitet, speichert den Anfrageidentifikator in einem lokalen Vektor (= Anfragevektor) ab. Zusammen mit dem Anfrageidentifikator werden der Hostname, dessen Lastdeskriptor gesucht ist, sowie der Hostname des Senders der Anfrage bzw. die Portnummer des Nutzerprozesses, der die Anfrage generierte und versendete, im Anfragevektor abgelegt. Dieser Vektor enthält alle die Anfragen, die vom Rechnerknoten noch nicht abschließend bearbeitet wurden. Erst wenn die "End of answer" - Nachricht als Kennzeichen der beendeten Anfragebehandlung an den Sender der Anfrage verschickt wurde, wird der Eintrag zur Anfrage aus dem Anfragevektor des

Rechnerknotens gestrichen. Da drei mögliche Anfragetypen implementiert wurden, ergibt sich folgende Unterteilung:

Local host-Request - Nachricht

Diese Nachricht entspricht dem 1. Anfragetyp (Anfrage nach dem Lastdeskriptor eines Knoten der lokalen Domäne). Der Empfänger der Nachricht durchsucht seinen lokalen Lastvektor nach dem geforderten Lastdeskriptor. Die Antwort wird in der Ergebnisliste abgelegt, die anschließend durch eine "Answer" - Nachricht an den Nutzerprozeß übermittelt wird. Zum Abschluß der Anfragebearbeitung wird eine "End of answer" - Nachricht ebenfalls an den Nutzerprozeß, der die Anfrage versendete, verschickt.

Format: *R : L : <request-id> : <req_hostname>*

Bedeutung: *R* = Nachrichtentyp

L = Anfragetyp ("L" = *Local host-Request*)

< *request−id* > = Anfrageidentifikator

< *req_hostname* > = Hostname, dessen Lastdeskriptor gesucht ist

Hosts from local domain-Request - Nachricht

Durch diese Nachricht wird eine Anfrage 2. Typs an die Lastkomponente des lokalen Knotens übermittelt. Da die Lastdeskriptoren aller Knoten der lokalen Domäne erfragt werden, ist eine lokale Beantwortung möglich. Dazu werden alle Einträge des Lastvektors in der Ergebnisliste abgelegt, die anschließend mittels "Answer" - Nachrichten an den Nutzerprozeß gesendet werden. Die Anfragebearbeitung wird mit einer "End of answer" - Nachricht abgeschlossen, die ebenfalls an den Nutzerprozeß verschickt wird.

Format: *R : D : <request-id>*

Bedeutung: *R* = Nachrichtentyp

D = Anfragetyp ("D" = *Hosts from local domain-Request*)

< *request−id* > = Anfrageidentifikator

Host from other domain-Request - Nachricht

Diese Nachricht repräsentiert den 3. Anfragetyp (Anfrage nach dem Lastdeskriptor eines Knoten aus einer anderen Domäne). Der Empfänger dieser Nachricht versucht zunächst, die Anfrage lokal zu beantworten. Dazu wird der lokale Lastvektor nach dem geforderten Lastdeskriptor durchsucht. Wird keine Antwort gefunden, so leitet der Rechnerknoten die Anfrage an einen Gatewayknoten weiter, mit dem er in Broadcast-Verbindung steht. Bei der Auswahl des neuen Rechnerknotens für die Anfragebearbeitung werden folgende Informationen berücksichtigt.

Bei der Weiterleitung der Anfrage werden grundsätzlich alle Rechnerknotennamen mit übertragen, die die Anfrage bereits erfolglos bearbeitet haben. Beim Empfang einer Anfrage werden diese Namen in einer Liste (= *public_host*-Liste) abgespeichert. Der neue Zielknoten für die Anfragebearbeitung wird aus der lokalen Liste der Gatewayknoten ausgewählt, wobei dieser aber nicht in der *public_host*-Liste vorhanden sein darf. Dadurch wird verhindert, daß die Anfrage an Knoten nochmals weitergeleitet wird. Bevor die Anfrage weiter versendet wird, erfolgt die Eintragung der Anfrage in den Anfragevektor. Der Vorgang der Anfrageweiterleitung wird solange wiederholt, bis entweder die Anfrage beantwortet werden kann oder bis kein weiterer Zielknoten für die Anfragebearbeitung gefunden wird. Im ersten Fall wird der Antwortlastdeskriptor in die Ergebnisliste eingetragen. Die Anfragebearbeitung wird auf dem Knoten mit einer "End of answer" - Nachricht an den Sender der Anfrage beendet. Kann die bisher nicht beantwortete Anfrage von dem bearbeitenden Rechnerknoten an keinen neuen Zielknoten weitergeleitet werden, erfolgt die Versendung einer *erweiterten* "End of answer" - Nachricht an den Sender der Anfrage. Hierbei werden zusätzlich die Hostnamen der Rechnerknoten mit übertragen, die die Anfrage erfolglos beantwortet haben. Diese Informationen werden vom Empfänger bei der Auswahl des nächsten Zielknotens für die Weiterleitung der Anfrage berücksichtigt. Nach dem Versenden der "End of answer" - Nachricht ist für den Rechnerknoten die Bearbeitung der Anfrage beendet. Daher wird abschließend der Eintrag zur Anfrage im Anfragevektor des Rechnerknotens gelöscht.

Format: *R : H : <request-id> : <req_hostname> : <maxpublic> {:<hostname>}*

Bedeutung: *R* = Nachrichtentyp

H = Anfragetyp ("H" = *Host from other domain-Request*)

< *request−id* > = Anfrageidentifikator

< *req_hostname* > = Hostname, dessen Lastdeskriptor gesucht ist

< *maxpublic* > = Anzahl der Knoten, die die Anfrage bereits erfolglos bearbeitet haben (Anfangswert = 0)

< *hostname* > = (bei *maxpublic* > 0) Namen der Knoten, die die Anfrage nicht beantworten konnten

Answer - Nachricht

Die während der Anfragebearbeitung in die Ergebnisliste eingetragenen Lastdeskriptoren werden mittels dieser Nachricht vor dem Versenden der "End of answer" - Nachricht an den Sender der Anfrage verschickt. Hauptinhalt der "Answer" - Nachricht ist ein Lastdeskriptor. Der Rechnerknoten, der die Anfrage bearbeitete, sendet den Ergebnislastdeskriptor an den Rechner bzw. Prozeß, von dem er die Anfrage empfangen hat. Wurde keine Antwort auf die Anfrage ermittelt, entfällt diese Nachricht. Empfängt ein Rechnerknoten (= Empfangsrechnerknoten) eine "Answer" - Nachricht von einem Rechner, an den die Anfrage weiter geleitet wurde, wird der Lastdeskriptor in die Ergebnisliste eingetragen. Anschließend erfolgt wieder die Versendung des Ergebnislastdeskriports in Form einer "Answer" - Nachricht. Ziel der Ergebnisweiterleitung ist nun der Rechnerknoten bzw. Prozeß von dem der Empfangsrechnerknoten die Anfrage empfangen hat (rekursive Anfragebearbeitung).

Format: *R : V : <request-id> : <host> : <time> : <load_min1.val> : <load_min1.b> : <load_min1.t> : <freemem.val> : <freemem.b> : <freemem.t> : <collisions.val> : <collisions.b> : <collisions.t> : <ipackets.val> : <ipackets.b> : <ipackets.t> : <opackets.val> : <opackets.b> : <opackets.t>*

Bedeutung: *R*: *V* = Nachrichtentyp

< *request−id* > = Anfrageidentifikator

<host> : <time> : ... : <opackets.t> = Ergebnislastdeskriptor der Anfrage

End of answer - Nachricht

Diese Nachricht wird genutzt, um dem Sender der Anfrage die Beendigung der Anfragebearbeitung auf einem Rechnerknoten mitzuteilen. Gleichzeitig wird die Anzahl der bei der abgeschlossenen Anfragebehandlung gefundenen Ergebnislastdeskriptoren übermittelt.

Bei der Bearbeitung einer Anfrage des 3. Typs werden für den Fall, daß keine Antwort auf die Anfrage ermittelt werden konnte, zusätzlich die Rechnerknotennamen mit übertragen, die die Anfrage bereits erfolglos bearbeitet haben (= *erweiterte* "End of answer" - Nachricht). Die "End of answer" - Nachricht wird zum Abschluß der Anfragebearbeitung auf einem Rechnerknoten an den Sender der Anfrage verschickt. Der Empfangsrechnerknoten einer "End of answer" - Nachricht reagiert in Abhängigkeit von der Anzahl der Ergebnislastdeskriptoren (*result_number*). Wurde keine Antwort auf die Anfrage durch deren Weiterleitung gefunden (*result_number* = 0), wird ein neuer Zielknoten für die Anfrageversendung ausgewählt. Ist diese erneute Weiterleitung nicht möglich, wird die Anfragebehandlung auf diesem Rechnerknoten mit der Versendung der *erweiterten* "End of answer" - Nachricht beendet. Wurde ein Ergebnislastdeskriptor für die Anfrage ermittelt (*result_number* = 1), wird vom Empfangsrechnerknoten die Anfragebearbeitung auf diesem Rechner unmittelbar mit der Versendung der "End of answer" - Nachricht an den Sender der Anfrage abgeschlossen.

Format: *R : E : <request-id> : <result_number> {: <value>} {: <hostname> }*

Bedeutung: *R*: *E* = Nachrichtentyp

< *request−id* > = Anfrageidentifikator

< *result_number* > = Anzahl der für die Anfrage gefundenen Ergebnislastdeskriptoren

< *value* > = nur bei *erweiterter* "End of answer" - Nachricht -> Anzahl der Knoten, die die Anfrage bereits erfolglos bearbeitet haben

< *hostname* > = nur bei *erweiterter* "End of answer" - Nachricht -> Namen der Knoten, die die Anfrage nicht beantworten konnten

5.4.6. Implementierungsaspekte

Die beschriebenen Aufgaben der Lastkomponente bzgl. der Lastbestimmung, der Aufbereitung der Lastinformationen und des Lastinformationsmanagement werden auf jedem Rechnerknoten durch einen Lastdämon *loadd* realisiert. Dieser Dämon läuft im Hintergrund für den Benutzer nicht sichtbar und verwaltet die für die Lastbalancierung notwendigen Lastinformationen sowohl des lokalen Rechnerknotens als auch der Rechner der Domäne. Voraussetzung dafür, daß von einem Rechnerknoten Informationen bzgl. der Belastung verfügbar sind, ist ein aktiver Lastdämon auf dem Rechner.

Zur Implementierung der Kommunikation im Rahmen des Lastinformationsaustausches und der Anfragebearbeitung wird die Klassenbibliothek *libipc* [Meye92] genutzt. Die Klassenbibliothek enthält eine Klassenhierarchie für die Interprozeßkommunikation (IPC - Interprocess Communication) mit verschiedenen Transportprotokollen. In der Bibliothek wird die Basisklasse *ipc* als abstrakter Datentyp für die Interprozeßkommunikation definiert. Von dieser Basisklasse, die protokollunabhängig ist, kann eine Klasse *sock* für die Kommunikation mittels Sockets abgeleitet werden. Die Sockets dienen als Kommunikationsendpunkte für die Kommunikation zwischen Prozessen auf einem bzw. auf verschiedenen Rechnern. Bei der Instanzenbildung von der abgeleiteten Klasse *sock* ist der Sockettyp (SOCK_DRAM) in Abhängigkeit vom genutzten Transportprotokoll (UDP) anzugeben. Mit den Methoden der Klasseninstanz kann das Senden und Empfangen von Daten realisiert werden. Damit Daten von bzw. zu einem konkreten Prozeß der Lastkomponente eines lokalen oder entfernten Rechnerknotens übertragen werden können, werden Ports (Verbindungsendpunkte) erzeugt. Die Kommunikation im Zusammenhang mit dem Lastinformationsaustausch erfolgt über das vom Lastdämon erzeugte Port *HeaDgra*. Anfragen an den Lastdämon werden an das vom Dämon erzeugte Port *HeaDlra* gesendet. Beide Ports sind in HEAD speziell definierte Verbindungsendpunkte†. Erfolgt die Weiterleitung einer Anfrage, da der lokale Lastdämon die Anfrage nicht beantworten konnte, wird dazu ein temporäres Port ge-

† Portnummer wird durch */etc/services* oder analogen Mechanismus übersetzt

nutzt.

Die Klasse `loadif` beschreibt die Schnittstelle zur Realisierung von Anfragen eines Nutzers an den Lastdämon des lokalen Rechnerknotens.

```
class loadif {

private:
      sock        *server;
      sock        *local;

public:
                  loadif();
                  ~loadif();
      int         get(char *reqhost, char *qual, loadindex vec[], int maxidx);
};
```

Im Konstruktor *loadif::loadif()* dieser Klasse wird das Empfangsport der Anfrage *HeaDlra* des lokalen Lastdämons eröffnet. Außerdem wird durch den Konstruktor ein Port für den Empfang der Antworten auf die Anfrage angelegt.

Mit der Methode *loadif::get* kann eine Anfrage an den Lastdämon des lokalen Rechnerknotens generiert werden.
reqhost ist der Hostname des Rechnerknotens, dessen Lastdeskriptor gesucht ist.
qual repräsentiert den Typ der Anfrage (L,D,H) (siehe Abs. 5.4.5.).
vec[] ist ein vom Nutzer definiertes Array (Ergebnisvektor) vom Typ *loadindex*, in das die gefundenen Antwortlastdeskriptoren eingetragen werden.
maxidx gibt die Dimension des Ergebnisvektors an bzw. die Anzahl an Antwortlastdeskriptoren, die in das Array eingetragen werden können.

Als Ergebnis der Funktion wird die Anzahl der während der Anfragebearbeitung gefundenen Antwortlastdeskriptoren zurückgegeben. Wurde keine Antwort gefunden, beträgt der Rückgabewert 0. Im Fehlerfall wird ein Wert kleiner Null zurückgegeben. Durch den Nutzer ist nach Beendigung der Funktion zu überprüfen, ob ein ausreichend großes Array für die Eintragung der Antwortlastdeskriptoren zur Verfügung gestellt worden ist und ob somit keine Antworten verloren gegangen sind (*maxidx* ≥

Funktionsrückgabewert). Wurde die Dimension des Ergebnisvektor *vec[]* zu klein gewählt, sind die überzähligen Antwortlastdeskriptoren durch die Funktion verworfen worden. Der Nutzer kann die Anfrage erneut generieren und einen entsprechend großen Ergebnisvektor (*vec[Rückgabewert des ersten Anfrageversuchs]*) bereitstellen. Ein Auszug aus einem Beispielprogramm ist zur Verdeutlichung der Anwendung der Funktion *loadif::get* im folgenden dargestellt.

```
#include "load.h"           //declaration of class loadif and class loadindex

const MAXENTRIES = 5;
loadindex  vec[MAXENTRIES];
char *reqhost, *reqtyp;
int   maxidx = MAXENTRIES, result_number;
loadif *g;

main ()
{
      ...
      g = new loadif();
      result_number = g->get(reqhost, reqtyp, vec, maxidx);
      if (result_number > 0) {
            if (result_number > maxidx) {
                  cout <<"ATTENTION: result_number > maxidx\n";
            }
            else {
                  for (int i = 0; i < result_number; ++i) {
                        cout << "host=" << vec[i].host << ", ";
                        cout << "time=" << vec[i].time << ", ";
                        cout << "loadavg=" << vec[i].load_min1;
                        cout <<"\n";
                  }
            }
      }
      else {
            if (result_number == 0) {
                  cout<<"no load index are found\n";
            }
            else {
                  cout<<"error by loadif::get()\n";
```

```
                }
            }
            . . .
        }
```

5.5. Betrachtung des durch die Lastkomponente erzeugten Overhead

Die Operationen, die im Zusammenhang mit der Lastmessung, der Meßwerteverdichtung und dem Lastinformationsaustausch auszuführen sind, verursachen selbst eine Last auf den Rechnerknoten sowie dem Netzwerk und erzeugen einen entsprechenden Overhead. Um den zusätzlichen Aufwand auf einem geringen Niveau zu halten, wurden bereits bei der Realisierung der Aufgaben der Lastkomponente folgende Maßnahmen zur Verringerung des Overheads umgesetzt:

- Beschränkung der Lastparameter auf eine geringe Anzahl von aussagekräftigen Größen, die vom UNIX-System selbständig gemessen werden
- Anpassung der Meß- und der Berechnungsintervallänge an die Lastwechselzyklen
- Begrenzung des Lastinformationsaustausches auf die Rechnerknoten der Domäne
- Anlaß zur Lastinformationsversendung sind nur signifikante Belastungsänderungen des Rechnerknotens oder der Ablauf eines festgelegten Zeitintervalls seit der letzten Benachrichtigung der anderen Knoten über den aktuellen Lastzustand des lokalen Rechners

Zur Überprüfung des von der Lastkomponente erzeugten Overheads wurden Untersuchungen auf drei Workstations (Sun-SPARCstation) durchgeführt. Zwei Workstations (n_2, n_6) befanden sich in verschiedenen Domänen. Die beiden Domänen wurden durch den 3. Rechnerknoten (n_3 = Gatewayknoten) miteinander verbunden (siehe Abb. 35).

Auf den drei Workstations wurde jeweils der Lastdämon für 12 Stunden aktiviert und Messungen mit den UNIX-Werkzeugen *prof* und *time* realisiert. Jeder Lastdämon führte die notwendigen Operationen im Rahmen der Lastmessung auf dem lokalen Rechnerknoten aus und sendete die aktuellen Lastinformationen an die auf den anderen Rechnerknoten der Domäne aktivierten Lastdämons.

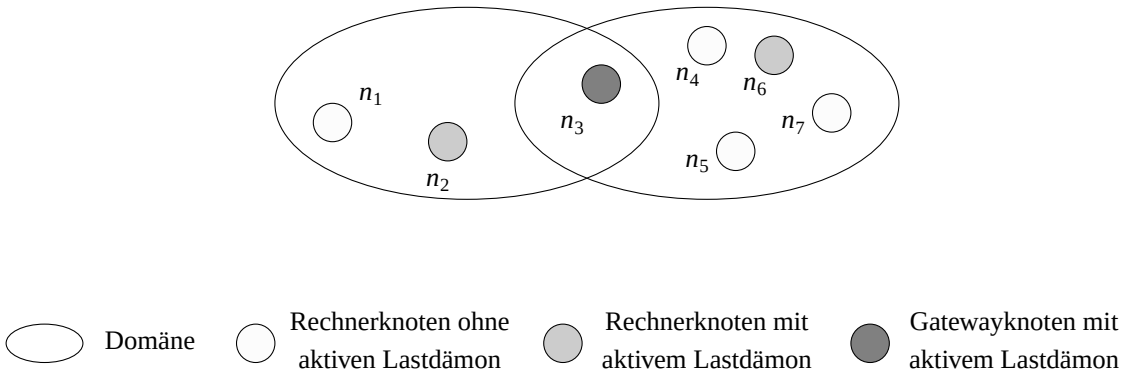

Abb. 35: Versuchsaufbau

Aussagen über den von einem Lastdämon erzeugten Overhead sind aus dem Verhältnis von der Zeit, die der Lastdämon die CPU (CPU-Zeit im Benutzermodus + CPU-Zeit im Systemmodus) nutzte, und der Gesamtlaufzeit des Dämons (elapsed time) abzuleiten (siehe Tab.).

Work-station	**vom Dämon verbrauchte CPU-Zeit**	**Gesamt-lauf-zeit**	**prozentual verbrauch-te CPU-Zeit bezogen auf Gesamtlaufzeit**
n_2	10,40 sec	12 h	0,02%
n_3	29,85 sec	12 h	0,07%
n_6	4,76 sec	12 h	0,01%

Im Vergleich zu den anderen Rechnerknoten beansprucht der Lastdämon des Gatewayknotens (n_3) die CPU stärker, weil der Gatewayknoten am Lastinformationsaustausch in beiden Domänen beteiligt ist. Dadurch ist ein größerer Aufwand für die Informationsversendung und die Informationsverwaltung erforderlich. Aus den Meßergebnissen kann folgende wichtige Erkenntnis abgeleitet werden: Die Operationen des Lastdämons erzeugen auf den Rechnerknoten einen unbedeutenden Overhead.

In weiteren Untersuchungen wurde überprüft, wieviel Zeit die einzelnen Operationen der Lastmessung, der Meßwerteverdichtung und des Lastinformationsaustausches eines Lastdämons für die Ausführung benötigen. In Tabelle sind beispielgebend einige

Operationen und deren Ausführungszeiten angeben. Alle Operationen des Lastdämons beanspruchten zusammen eine Gesamtausführungszeit von 29,85 sec während einer Gesamtlaufzeit des Dämons von 12 Stunden.

Operation	**% Zeit**	**Ausführungs-Zeit in sec**	**Anzahl der Aufrufe**	**msec/Aufruf**
poll	3,6	1,07	266622	0,04
recvfrom	2,7	0,81	6225	0,13
strtod	2,1	0,62	61580	0.01
atof	2,1	0,62	61580	0.01
split_ipc	1,1	0,32		
sendto	1,0	0,29	3206	0,09
nlist	0,5	0,16	2	80,00
loadd_bcastsock	0,5	0,15	1	150,00
gettimeday	0,3	0,10	21549	0,00
ipc::poll	0,3	0,08		
ipc::elect	0,2	0,07		
open	0,0	0,01	14	0,71

Die notwendigen Operationen zur Realisierung der Lastinformationsversendung und des Informationsempfang (z.B.: *poll*, *ipc*: : *poll*, *ipc*: : *elect*, *loadd_bcastsock*, *open*, *sendto*, *recvfrom*) verursachen annähernd 10% der Gesamtausführungszeit. Eine Verringerung dieses Anteils ist nur durch eine weitere Begrenzung der Häufigkeit bzw. der Notwendigkeit für eine Lastinformationsversendung erzielbar. Ebenso ist die Häufigkeit der erforderlichen Operationen im Zusammenhang mit der Lastmessung (z.B.: *nlist*, *gettimeofday*) und des damit verbundenen Aufwands nur durch die Verlängerung des Meßintervalls zu vermindern. Diese Einschränkungen sind aber grundsätzlich nur unter Berücksichtigung der Anforderungen der Lastbalancierung an die Aktualität der Lastinformationen realisierbar.

Folgender Ansatzpunkt bietet sich, um den zusätzlichen Aufwand im Rahmen des Lastinformationsaustausches zu reduzieren: Indem die Nachrichten nicht in Form von ASCII-Zeichenketten, sondern in einer externen Datenrepräsentation (maschinen- und netzwerkunabhängig), z.B. ASN.1 (Abstract Syntax Notation One), XDR (External Data Representation), übertragen werden, entfällt der Aufwand zur Konvertierung und Dekonvertierung der Nachrichten (z.B.: *strtod*, *atof*, *split_ipc*).

6. Ausblick

Alle Probleme und Fragen im Zusammenhang mit der Lastmessung und dem Lastinformationsaustausch in einem heterogenen, verteilten DBS konnten in der vorliegenden Arbeit nicht behandelt bzw. ausführlich erörtert werden. Gründe dafür sind einerseits, daß der Prototyp des HEAD - Projektes sich in der Entwicklungsphase befindet und daher einige Systemkomponenten noch nicht verfügbar sind. Andererseits wurde innerhalb des bestehenden zeitlichen Rahmens dieser Arbeit versucht, auf die grundlegenden Herangehensweisen der Lastmessung und -balancierung einzugehen.

Die dynamische Lastbalancierung während der Abarbeitung der Teilanfragen wird durch den Local Query Manager realisiert. Da diese Komponente (LQM) zum derzeitigen Entwicklungsstand des HEAD - Projektes noch nicht zur Anwendung bereitsteht, konnte die konkrete Nutzung der ermittelten Lastinformationen bei der Lastbalancierung und somit deren Einfluß auf die Effizienz der Lastausgleichsentscheidungen bisher nicht überprüft werden. Es sind daher keine Kenntnisse vorhanden, ob die Häufigkeit der Messungen und demzufolge die Aktualität der verfügbaren Lastinformationen für die dynamische Lastbalancierung ausreichend ist.

Um einen effizienten Lastausgleich zu realisieren, kann eine Angleichung von Parametern der Lastbestimmungs- und der Lastinformationsaustauschmethode erforderlich sein. Die Länge des Meß- und des Berechnungsintervalls sowie die Schwellwerte bzgl. der Notwendigkeit eines Informationsaustausches und die Häufigkeit der Versendung von Lastinformationen sind entsprechend den Anforderungen der Lastbalancierung unter gleichzeitiger Berücksichtigung des damit verbundenen Overhead anzupassen. Indem diese Parameter als Kommandozeilen-Argumente bei der Aktivierung an den Lastdämon übergeben werden, kann die Parameteranpassung an sich ändernde Forderungen leicht erfolgen.

Die Menge der während der Lastmessung ermittelten Lastparameter ist ohne großen Aufwand erweiterbar bzw. zu verändern und somit an aktuelle Anforderungen anzugleichen. Durch Ergänzung der Liste von Parametern, die aus dem UNIX-Kern ausgelesen werden, können zusätzliche Informationen über die Belastung des Rechnerknotens und des Netzwerks erhalten werden.

Aus Effizienzgründen ist der Lastinformationsaustausch auf die Rechnerknoten der Domäne beschränkt, so daß alle Rechner vollständig über die Lastzustände der Knoten in der eigenen Domäne informiert sind. Es sind bisher keine Untersuchungen durchgeführt worden, welche Anzahl von Rechnerknoten in einer Domäne maximal zusammengefaßt werden sollten. In weitergehenden Arbeiten ist daher zu überprüfen, welchen Einfluß die Größe der Domäne (= Anzahl der Rechnerknoten in der Domäne) auf die erforderlichen Kosten für die Kommunikation und die Informationsverwaltung im Zusammenhang mit dem Lastinformationsaustausch besitzt.

Aus den in einem Lastdeskriptor eines Rechnerknotens enthaltenen Lastwerten können Kenntnisse über die Belastung der CPU und des Speichers eines Rechners sowie über die Netzwerkbelastung abgeleitet werden. Wird ein einziger aussagekräftiger Lastparameter benötigt, ist eine Zusammenfassung der Lastwerte der drei Ressourcen erforderlich. Dabei sind die einzelnen Größen des Lastdeskriptors entsprechend der Bedeutung der Ressource bei der Anfragebearbeitung in einem verteilten System zu wichten. Gleichzeitig ist die Heterogenität des Systems zu beachten, um einen Vergleich der zusammenfassenden Lastparameter zwischen den Rechnerknoten zu ermöglichen.

Ein Ansatzpunkt für weitergehende Arbeiten bzgl. der Minimierung des Overhead beim Lastausgleich wäre die Überprüfung, inwieweit es erforderlich ist, daß die Lastinformationen aller an der Lastbalancierung beteiligten Rechnerknoten im Lastvektor verwaltet werden. Für die Lastausgleichsentscheidung sind die Lastinformationen der gering belasteten Rechnerknoten als mögliche Zielknoten der Prozeßverlagerung von Bedeutung. Indem die Größe des Lastvektors beschränkt wird, sind nur die Lastinformationen einer begrenzten Anzahl von Rechnerknoten zu verwalten. Der Lastvektor ist aufsteigend nach den Lastwerten der Rechner sortiert. Im Lastvektor sind daher grundsätzlich Lastinformationen der am geringsten belasteten Rechner des Systems verfügbar. Die Speicherung der Lastinformationen der schwer belasteten Knoten, die bei der Zielrechnersuche für einen Prozeßtransfer ohnehin nicht zu berücksichtigen sind, entfällt infolgedessen.

Die Betrachtung der Transportprotokolle für die Kommunikation im Rahmen der Anfragebearbeitung ergab, daß RDP das am besten geeignete Protokoll ist. Da aber RDP derzeitig in der HEAD - Systemumgebung nicht verfügbar ist, wird das Transportpro-

tokoll UDP genutzt. Dieses Protokoll gewährleistet keine sichere Nachrichtenübertragung. Aus diesem Grund wurden zusätzlich eigene Funktionen realisiert, um die Zuverlässigkeit der Nachrichtenkommunikation zu erhöhen. Wird RDP anstelle von UDP verwendet, kann durch den Einsatz dieses zuverlässigen Transportprotokolls der zusätzliche Aufwand für eine sichere Kommunikation reduziert und gleichzeitig die Kosten der Nachrichtenübertragung gesenkt werden.

Durch die Realisierung der Lastmessung und des Lastinformationsaustausches verfügt ein lokaler Rechner über die Belastungsinformationen der an der Lastbalancierung beteiligten Rechnerknoten. Diese Informationen bilden eine Grundlage für die weiteren noch zu realisierenden Operationen im Rahmen des Lastausgleiches. Die im Lastvektor des lokalen Rechnerknotens gehaltenen Lastinformationen sind bei der Zielrechnersuche für einen Prozeßtransfer einzubeziehen. Durch den stetigen Lastinformationsaustausch ist kein zusätzlicher Kommunikationsaufwand erforderlich, um über aktuelle Belastungsinformationen der potentiellen Zielknoten bei der Lastbalancierungsentscheidung zu verfügen.

Neben den Lastinformationen sind auch Kenntnisse über die Leistungsfähigkeit der vorhandenen Rechnerknoten und des Netzwerkes bei der Prozeßtransferentscheidung zu beachten. Dadurch erfolgt entsprechend der heterogenen Systemumgebung von HE$_A$D eine Normalisierung der aktuellen Belastungsinformationen gemäß dem unterschiedlichen Leistungsvermögen der Rechner. Vorhandenes Wissen über die zu bearbeitenden Aufträge und Prozesse, wie z.B. deren Rechenzeitbedarf und bestehende Reihenfolgebeziehungen, sind bei der Wahl des zu transferierenden Prozesses und des Zielrechners für die entfernte Prozeßausführung zu berücksichtigen.

Im Zusammenhang mit der Wahl eines Zielknotens für die Bearbeitung von Prozessen erfolgt auf der Basis des Kostenmodells eine Abschätzung der entstehenden Kosten einer eventuellen Verlagerung von Prozessen auf andere Rechnerknoten.

In HE$_A$D wird eine senderinitiierte Lastbalancierung angestrebt. Daher ist ein Schwellwert festzulegen, ab wann ein Rechner als schwer belastet gilt. Überschreitet die Last des lokalen Rechnerknotens den Schwellwert, ist der Anlaß für einen Prozeßtransfer gegeben. Wird der Schwellwert in Abhängigkeit von der Systemlast dynamisch fixiert, erfolgt eine stetige Anpassung an das bestehende Lastniveau im System.

Die erforderlichen Informationen über die Lastzustände der anderen Knoten zur Festlegung des Schwellwertes können aus dem Lastvektor des lokalen Rechners entnommen werden.

Um eine dynamische Lastbalancierung zu erzielen, sind die angedeuteten Aufgaben bzgl. des Prozeßtransfers und der Zielrechnersuche zu realisieren.

Lang92.

Lang92.

Förs79.

Förs79.

Aute90.

Autenrieth, K., Dappa, H., Grevel, M., Kubalski, W., and Bartsch, T., “Technik verteilter Betriebssysteme,” *Huthig Buch Verlag,* Heidelberg (1990).

Bara85.

Barak, A. and Shiloh, A., “A Distributed Load Balancing Policy for a Multicomputer,” *Software - Practice and Experience,* 15, 9, pp. 901-913 (Sept. 1985).

Bara93.

Barak, A., Guday, S., and Wheeler, R. G., *The MOSIX Distributed Operating System*, Springer Verlag, Berlin, Heidelberg (1993).

Baum89.

Baumgartner, K. M. and Wah, B. W., “GAMMON: A Load Balancing Strategy for Local Computer Systems with Multiaccess Networks,” *IEEE Transactions on Computers,* 38, 8, pp. 1098-1109 (Aug. 1989).

Beck92.

Becker, W., “Lastbalancierung in heterogenen Client-Server Architekturen,” Fakultatsbericht Nr. 1992/1, Fakultat fur Informatik, Universitat Stuttgart (1992).

Beye88.

Beyer, O., Hackel, H., Pieper, V., and Tiedge, J., “Wahrscheinlichkeitsrechnung

und mathematische Statistik," *BSB B.G. Teubner Verlagsgesellschaft,* Leipzig (1988).

Bono89.

Bonomi, F., Flemming, P. J., and Steinberg, P., "An Adaptive Join-the-biased-Queue Rule for Load Sharing on distributed Computer Systems" in *IEEE Proceedings of the 28th Conference on Decision and Control,* Tampa, Florida, USA (Dez. 1989).

Brya81.

Bryant, R. and Finkel, R. A., "A Stable Distributed Scheduling Algorithm" in *Proceedings of the 2nd International Conference on Computing Systems,* pp. 314-323, IEEE, Paris (April 1981).

Casa87.

Casavant, T. L. and Kuhl, J. G., "Analysis of three dynamic distributed load-balancing strategies with varying global information requirements" in *Proceeding of the 7th IEEE International Conference on Distributed Computing Systems,* pp. 185-192 (Sept. 1987).

Ceri85.

Ceri, S. and Pelagatti, G., "Distributed Databases: Principles and Systems," *McGraw-Hill,* New York (1985).

Chou82.

Chou, T. C. K. and Abraham, J. A., "Load Balancing in Distributed Systems," *IEEE Transactions on Software Engineering,* SE-8, 4, pp. 401-412 (Jul. 1982).

Chow79.

Chow, Y. Ch. and Kohler, W. H., "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Transactions on Computers,* C-28, 5, pp. 354-361 (May 1979).

Chwd90.

Chowdhury, S., "The Greedy Load Sharing Algorithm," *Journal of Parallel and Distributed Computing,* 9, 1, pp. 93-99 (1990).

Chu80.

Chu, W. W., Holloway, L. J., Lan, M. T., and Efe, K., “Task allocation in distributed data processing,” *IEEE Computer,* pp. 57-68 (Nov. 1980).

Chyl88.

Chylla, P. and Hegering, H.-G., “Ethernet - LANs: Planung, Realisierung und Netzmanagement,” *Datacom-Buchverlag, Lipinski,* Pulheim (1988).

Cici92.

Ciciani, B., Dias, D. M., and Yu, P. S., “Dynamic and static loadsharing in hybrid distributed centralized database systems,” *Computer Systems Science & Engineering,* 7, 1, pp. 25-41 (Jan. 1992).

Come92.

Comer, D. M., *Internetworking with TCP/IP,* Englewood Cliffs, NJ (1992).

Eage86a.

Eager, D. L., Lazowska, E. D., and Zahorjan, J., “A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing,” *Performance Evaluation,* 6, pp. 53-68 (March 1986).

Eage86b.

Eager, D. L., Lazowska, E. D., and Zahorjan, J., “Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Transactions on Software Engineering,* SE-12, 5, pp. 662-675 (May 1986).

Ferr83.

Ferrari, D., Serazzi, G., and Zeigner, A., “Measurement and turing of computer systems,” *Prentice-Hall,* Englewood Cliffs, NJ, USA (1983).

Ferr85.

Ferrari, D., “A Study of Load Indices for Load Balancing Schemes,” UCB / CSD Report 85/262, Computer Science Division, University of California Berkeley, USA (1985).

Flac93a.

Flach, G., Langer, U., and Meyer, H., “Das Projekt HEaD,” *Forschungsbericht (DFG),* Universität Rostock, Fachbereich Informatik, Arbeitsgruppe

Datenbanken, Rostock (1993).

Flac93b.

Flach, G., “Konzeption eines Kostenmodells für ein verteiltes Datenbanksystem,” *Diplomarbeit,* Universität Rostock, Fachbereich Informatik, Rostock (Mai 1993).

Gosc91.

Goscinski, A., “Distributed Operating Systems: The Logical Design,” *Addison-Wesley Publishing Company,* Sydney (1991).

Hac86.

Hac, A. and Johnson, T. J., “A Study of Dynamic Load Balancing in a Distributed System” in *Proc. ACM SIGCOMM Symposium on Communication, Architecture, and Protocols,* pp. 348-356, Stowe, Vermont, USA (Aug. 1986).

Hac89a.

Hac, A. and Jin, X., “Dynamic load balancing in a distributed system using a decentralized algorithm” in *Proceedings of the 7th IEEE International Conference on Distributed Computing Systems,* pp. 170-177 (Sept. 1989).

Hac89b.

Hac, A., “A distributed algorithm for performance improvement through file replication, file migration and process migration,” *IEEE Transaction on Software Engineering,* 15, 11 (Nov. 1989).

Harg90.

Harget, A. J. and Johnson, J. D., “Load Balancing Algorithms in loosely-coupled Distributed Systems: a Survey” in *Zedan, H. S. M. "Distributed Computer Systems", Butterworths,* pp. 85-108, London (1990).

Heis88.

Heiss, H.-U., “Überlast in Rechensystemen,” *Springer Verlag,* Berlin (1988).

Heis91.

Heiss, H.-U., “Classification of problems and algorithms for processor allocation in parallel systems,” Interner Bericht Nr. 7/91, Fakultaẗ fuẗr Informatik, Universitaẗ Karlsruhe (1991).

Heis92.

Heiss, H.-U. and Schmitz, M., “Distributed Load Balancing Using a Physical Analogy,” Interner Bericht Nr. 5/92, Fakultaẗ fuŕ Informatik, Universitaẗ Karlsruhe (1992).

Jain91.

Jain, R., “The Art of Computer Systems Performance Analysis,” *John Willey & Sons, INC,* New York, Chichester (1991).

Kauf91.

Kauffels, F. J., “Rechnernetzwerk - Systemarchitektur und Datenkommunikation,” *Wissenschaftsverlag,* Mannheim, Wien, Zuŕich (1991).

Krue84.

Krueger, P. and Finkel, R. A., “An Adaptive Load Balancing Algorithm for a Multicomputer,” Tech Rep. 539, Department of Computer Science, University of Wisconsin-Madison, USA (April 1984).

Krue88.

Krueger, P. and Livny, M., “A comparison of preemptive and non-preemptive load distributing” in *Proceedings of the 8th International Conference on Distributed Computer Systems,* pp. 123-130 (June 1988).

Lee86.

Lee, K. J. and Towsley, D., “A Comparison of Priority Based Decentralized Load Balancing Policies” in *ACM Performance Evaluation Review: Proc. Performance ’86 and ACM SIGMETRICS 1986. Vol. 14,* pp. 70-77 (May 1986).

Leff90.

Leffler, S. J., McKusick, M. K., Karels, M. J., and Quaterman, J. S., “Das 4.3 BSD UNIX Betriebssystem,” *Addison-Wesley Publishing Company,* Bonn, Muñchen (1990).

Lela86.

Leland, W. E. and Ott, T. J., “Load-balancing Heuristics and Process Behavior,” *ACM SIGMETRICS Performance Evaluation Review,* 14, 1, pp. 54-69 (May 1986).

Lin92.

Lin, H. C. and Raghavendra, C. S., “A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC),” *IEEE Transactions on Software Engineering,* 18, 2 (Feb. 1992).

Livn82.

Livny, M. and Melman, M., “Load Balancing in Homogeneous Broadcast Distributed Systems” in *Proceedings Modeling Performance Evaluation Computing Systems, ACM SIGMETRICS,* pp. 47-55 (April 1982).

Ludw93.

Ludwig, T., “Load Management on Multiprocessor Systems” in *Bode, A., Cin, M. D., "Parallel Computer Architectures", Springer Verlag,* pp. 87-101, Berlin, Heidelberg (1993).

McGr75.

McGregor, P. V. and Boorstyn, R. R., “Optimal Load Sharing in a Computer Network” in *Procedings International Conference on Communications,* 3, pp. 41.14-41.19 (1975).

Meye92.

Meyer, H., “`libipc` − A C++-class library for process interaction and communication,” *Technical Report,* 2-92, University of Rostock, Dept. of Computer Science, Database Research Group (1992).

Mikk88.

Mikkilineni, K. P. and Su, S. Y. W., “An Evaluation of Relational Join Algorithms in a Pipelined Query Processing Environment,” *IEEE Trans. on Software Eng.,* 14, 6nd, pp. 838-848 (Jun. 1988).

Mirc89.

Mirchandaney, R., Towsley, D., and Stankovic, J. A., “Adaptive Load Sharing in Heterogeneous Systems” in *Proceedings of the 9th International Conference on Distributed Computing Systems,* pp. 298-306, Newport Beach, USA (June 1989).

Ni85.

Ni, L. M., Xu, C. W., and Gendreau, T. B., “A Distributed Drafting Algorithm for

Load Balancing," *IEEE Transactions on Software Engineering,* SE-11, 10, pp. 1153-1161 (Oct. 1985).

Osse92.

Osser, W., "Automatic Process Selection for Load Balancing," *Master of Science in Computer Engineering,* University of California of Santa Cruz (June 1992).

Ozsu91.

Ozsu, M.T. and Valduriez, P., "Principles of distributed database systems" in *Prentice-Hall, Inc., Englewood Cliffs* (1991).

Post80.

Postel, J., "User Datagram Protocol," *RFC 768* (Aug. 1980).

Post81a.

Postel, J., "Transmission Control Protocol," *RFC 793* (Sept. 1981).

Post81b.

Postel, J., "Internet Protocol," *RFC 791* (Sept. 1981).

Rahm93.

Rahm, E., "Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems" in *Proceedings of the 19th VLDB Conference,* Dublin, Ireland (1993).

Sach92.

Sachs, L., "Angewandte Statistik," *Springer Verlag,* Berlin (1992).

Sant92.

Santifaller, M., "TCP/IP und NFS in Theorie und Praxis," *Addison-Wesley Publishing Company* (1992).

Scha92.

Schabernack, J., "Lastausgleichsverfahren in verteilten Systemen - Überblick und Klassifikation," *Informationstechnik,* 34, 5, pp. 280-295 (1992).

Shen85.

Shen, C. C. and Tsai, W. H., "A graph matching approach to optimal task asignment in distributed computing systems using a minimax criterion," *IEEE*

Transaction on Computers, 34, 3, pp. 197-203 (1985).

Shen88.

Shen, S., "Cooperative Distributed Dynamic Load Balancing," *Acta Informatica,* 25, pp. 663-676 (1988).

Shoj91.

Shoja, G. C., "A Distributed Facility for Load Sharing and Parallel Processing Among Workstations," *The Journal of Systems and Software,* 14, 3, pp. 163-172 (March 1991).

Slom89.

Sloman, M. and Kramer, J., "Verteilte Systeme und Rechnernetze," *Carl Hanser Verlag,* Wien, Muñchen (1989).

Stan84.

Stankovic, J. A., "Simulations of Three Adaptive Decentralized Controllled, Task Scheduling Algorithms," *Computer Networks,* 8, 3, pp. 199-217 (June 1984).

Stan85a.

Stankovic, J. A., "An application of bayesian decision theory to decentralized control of job scheduling," *IEEE Transactions on Computers,* 34, 2, pp. 117-130 (1985).

Stan85b.

Stankovic, J. A., "Stability and distributed scheduling algorithms," *IEEE Transaction on Software Engineering,* 11, 10, pp. 1141-1152 (Oct. 1985).

Stev91.

Stevens, R., "UNIX Network Programming," *Addison-Wesley,* Reading, Massachusetts (1991).

Ston77.

Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Transaction on Software Engineering,* SE-3, pp. 85-93 (January 1977).

Ston86.

Stonebraker, M., "The Case for Shared-Nothing," *IEEE Database Engineering,* 9,

1 (1986).

Tane81.

Tanenbaum, A. S., “Computer Networks,” *Prentice Hall*, Eaglewood Cliffs, NJ (1981).

Tant85.

Tantawi, A. N. and Towsley, D., “Optimal Static Load Balancing in Distributed Computer Systems,” *Journal of the ACM*, 32, 2, pp. 445-465 (April 1985).

Velt84.

Velten, D., Hinden, R., and Sae, J., “Reliable Data Protocol,” *RFC 908* (July 1984).

Wang85.

Wang, Y. T. and Morris, R. J. T., “Load Sharing in Distributed Systems,” *IEEE Transactions on Computers*, C-34, 3, pp. 204-217 (March 1985).

Wash93.

Washburn, K., “TCP/IP - Running a Successful Network,” *Addison Wesley Publishing Company*, Bonn (1993).

Wiso90.

Wisotzki, A., “Parallele Anfrageverarbeitung in HEaD,” *Forschungsbericht*, 1, Universiat Rostock, Fachbereich Informatik (1990).

Zhou86.

Zhou, S, “An Experimental Assessment of Resource Queue Lengths as Load Indices,” UCB / CSD Rep. 86/298, Computer Science Division, University of California Berkeley, USA (June 1986).

Zhou87.

Zhou, S. and Ferrari, D., “A Measurement Study of Load Balancing Performance” in *Proceedings 7th IEEE International Conference on Distributed Computing Systems*, pp. 490-497 (Sept. 1987).

Zhou88.

Zhou, S., “A Trace-Driven Simulation Study of Dynamic Load Balancing,” *IEEE Transactions on Software Engineering*, 14, 9, pp. 1327-1341 (Sept. 1988).

Zhou93.

Zhou, S., Zheng, X., Wang, J., and Delisle, P., “Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems,” *Software - Practice and Experience*, 23, 12, pp. 1305-1336 (Dec. 1993).

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich erkläre, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 20.05.1994

Inhaltsverzeichnis