INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Konzeption des Managements von Lastbalancierung und Datenmigration in Workstation-Farms

Jakob Blätte

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dr. Bernhard Neumair, Markus Gutschmidt
Abgabedatum:	15. August 1994

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15. August 1994

Kapitel 1

Kurzfassung

Workstation-Farms (bzw. Workstation-Cluster) sind Ansammlungen von Computern im Netz, die als eine einzelne Computer-Ressource verwendet werden können mittels der Verwendung zusätzlicher verteilter Betriebsystem-Software, insbesondere mittels Lastverwaltungs- und Datenmigrations-Systemen. Es läßt sich die Beobachtung machen, daß die Ressourcen von Workstations in einem Rechner-Netz auch zu Zeiten hoher Belastung zu einem Großteil ungenutzt sind. Durch die Verwendung von geeigneten Lastbalancierungs- und Batch-Queueing-Systemen können diese zusätzlichen Ressourcen besser ausgenutzt werden. Lastbalancierung kann definiert werden als der Prozeß des Teilens von Rechner-Ressourcen mittels der transparenten Verteilung von System-Arbeitslast. Die Systemleistung kann dabei verbessert werden, indem Prozesse von stark ausgelasteten Knoten auf solche verschoben werden, die nur gering belastet sind. Entsprechend umfaßt Datenmigration den Prozeß des Teilens von Speicher-Ressourcen mittels der transparenten Verteilung von Datensätzen zur Optimierung von Speicheraufteilung und Gesamt-Systemleistung.

Lastverwaltungs-Mechanismen bedürfen des Managements, d.h. der Planung, Konfiguration, Steuerung, Überwachung, Fehlerbehebung und Verwaltung im Rechnernetz bzw. innerhalb des verteilten Systems. Es soll in dieser Arbeit ein Konzept zur Realisierung eines derartigen Managements von Lastbalancierung und Datenmigration vorgestellt werden. Nach einer einführenden Betrachtung von Lastverwaltungs-Systemen wird ein einheitlicher Beschreibungs-Rahmen für Lastbalancierungs-Algorithmen vorgestellt. Eine Analyse verschiedener Algorithmen und Lastbalancierungs-Produkte wird als Grundlage der Formulierung eines generischen Lastverwaltungs-Modells verwendet. Verschiedene Einsatz-Szenarien runden die Analyse von existierenden Methoden und Verfahren der Lastbalancierung ab und liefern die Grundlage, auf der das Management-Konzept entworfen werden soll. Innerhalb diese Konzeptes werden insbesondere das Informations-Modell vorgestellt, in dem eine Beschreibung der Management-Objekte erfolgt, sowie das Funktions-Modell, in dem näher auf Management-Aufgaben bezüglich Konfiguration, Fehler, Leistungsverhalten, Abrechnung und Sicherheit eingegangen wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 2

Einführung in die Problematik

2.1 Lastverwaltung in verteilten Systemen

Die Nützlichkeit und Bedienerfreundlichkeit von Computersystemen genauso wie deren effektiver und effizienter Einsatz sind entscheidend von hochentwickelter Betriebssystem-Software und dem Einsatz geeigneter Management-Systeme abhängig. Die als Down- und Rightsizing bekannte Entwicklung im Bereich großer Rechensysteme führte zu neuen Fragestellungen und Aufgaben bezüglich der Entwicklung von verteilten Rechensystemen, wie sie in zentralen Architekturen bisher nicht aufgetreten waren. Aber nicht nur die Entwicklung von Betriebssystemen gewann eine neue Dimension, auch das Management dieser verteilten Systeme wurde zunehmend wichtiger. Neue Herausforderungen stellten sich den Betreibern von Rechnernetzen und verteilten Systemen bei der Verwaltung und Pflege ihrer Systeme, deren Kapazitäten und Ressourcen nicht mehr zentral gewartet werden konnten, sondern über möglicherweise große Entfernungen verteilt waren.
Die Vorteile solcher Systeme liegen unter anderem darin, daß deren Kapazitäten trotz ihrer dezentralen Anordnung insgesamt oft größer sind, als die von zentralen Großrechen-Anlagen. Darüberhinaus sind verteilte Systeme leicht erweiterbar und damit flexibler als zentrale Systeme. Auch aufgrund vieler anderer Faktoren ist die Wirtschaftlichkeit dezentraler Rechenanlagen nicht zu übersehen.
Obwohl die Gesamtkapazität sogenannter Workstation-Cluster bzw. Workstation-Farms zur Bearbeitung großer verteilter Anwendungen durchaus geeignet wäre, wird dies heute noch unzureichend durch entsprechende System-Software unterstützt. Es läßt sich nämlich beobachten, daß ein Großteil der Kapazitäten eines Rechnernetzes z.B. in einem Unternehmen oder einer wissenschaftlichen Einrichtung auch in Zeiten starker Beanspruchung der Rechner ungenutzt sind. Diese zusätzlichen Kapazitäten ließen sich durch geeignete Unterstützung von verteilter Betriebssystem-Software derart nutzen, daß Anwendern Kapazitäten zur Verfügung gestellt werden könnten, die jene ihrer lokalen Rechner bei weitem übersteigen. Damit könnte es auch möglich gemacht werden, komplexe Probleme durch verteilte Anwendungen, die die Kapazitäten lokaler Systeme bei weitem übersteigen, optimal zu lösen. In gewissem Umfang wird eine derartige Möglichkeit heute bereits durch sogenannte Batch-Queueing Systeme zur Verfügung gestellt, wenn deren Schwerpunkt auch in erster Linie noch nicht im Bereich datenintensiver verteilter Anwendungen sondern vor allem in der Nutzung der im Cluster verfügbaren Rechenkapazität durch Prozesse und Prozess-Gruppen liegt.
Verteilte Systemsoftware bedarf genauso wie die Ressourcen, auf der diese arbeitet, des Ma-

nagements, also der Planung, Konfiguration, Steuerung, Überwachung und Fehlerbehebung sowie der Verwaltung. Insbesondere muß also auch ein System, das durch geeignete Verteilung von Prozessen und Daten auf das verteilte Rechensystem den Benutzern Kapazitäten zur Verfügung stellt, die über die lokal verfügbaren Kapazitäten hinausgehen, geeignet konfiguriert, gesteuert und überwacht werden, damit es innerhalb der Gesamtsystems seine Aufgabe effektiv und effizient erbringen kann.
In der vorliegenden Arbeit soll nun ein Konzept fürden effektiven und effizienten Einsatz eines Lastverwaltungs- und Datenmigrations-Systems durch den Einsatz geeigneter Management-Funktionalität vorgeschlagen werden.
Nach generellen Begriffsbestimmungen, einer Einführung in die Problematik der Lastverwaltungsaufgabe und der Vorstellung eines grundlegenden Modells der Lastbalancierung soll in Kapitel 3 zunächst ein einheitlicher Beschreibungsrahmen für existierende Lastbalancierungsalgorithmen vorgestellt werden. Die Analyse verschiedener Beispiele soll einen Überblick über das Spektrum von verschiedenen Algorithmen der Lastbalancierung bieten. Daran anschließend wird eine Analyse existierender Lastverwaltungs-Produkte vorgenommen, die die Grundlage für ein differenziertes Lastverwaltungs-Modell bildet, das in Kapitel 5 vorgestellt werden soll. Hier soll u.a. management-spezifische Merkmale von Produkten im Bereich der Lastbalancierung festgestellt werden und von solcher Funktionalität unterschieden werden, die lastverwaltungs-spezifisch ist. Dies bildet dann die Grundlage für das eigentliche Management-Konzept der Lastbalancierung und Datenmigration, das in Kapitel 6 vorgestellt werden soll.

2.2 Begriffsbestimmungen

Unter dem Begriff Netz- und System-Management versteht man die Summe aller Verfahren und Produkte zur Planung, Konfigurierung, Steuerung, Überwachung, Fehlerbehebung sowie Verwaltung von Rechnernetzen und verteilten Systemen. Dabei soll eine benutzerfreundliche und wirtschaftliche Unterstützung der Betreiber und Nutzer bei ihrer Arbeit mit dem Netz und seiner Komponenten erreicht werden. Netz- und System-Management umfaßt also die Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung des effektiven und effizienten Einsatzes eines Rechnernetzes bzw. eines verteilten Systems. ([Hege93] S68).
Die Dimensionen der Management-Szenarien umfassen das Komponenten-Management, das System-Management, das Anwendungs-Management sowie das Enterprise-Management. Während das Komponenten-Management in erster Linie das Management der Netzkomponenten betrifft und das Anwendungs-Management das Management der auf den Ressourcen operierenden verteilten Anwendungen zum Gegenstand hat, besteht die Aufgabe des System-Managements darin, "aufbauend auf den Systemressourcen die verteilt erbrachten Systemfunktionen so zu organisieren, daß die vernetzten Einzelsysteme nach außen wie ein (virtuelles) System erscheinen". ([Hege93] S100) Das Enterprise-Management zielt auf die Betrachtung der Teilbereiche Komponenten-Management, System-Management und Anwendungs-Management im Unternehmenskontext ab und ist demzufolge als übergeordnetes Management-Szenario anzusehen.
Allgemein kann festgehalten werden, daß immer dann von Netzmanagement gesprochen werden muß, wenn bei einem Verbund von Rechensystemen Management-Aspekte im Vordergrund stehen, die Ressourcen betreffen, die primär der Kommunikations-Möglichkeit zwischen Rechensystemen dienen, wohingegen Management-Aspekte, die den Betrieb des Ver-

bundes von Rechensystemen als verteiltes System mit entsprechend verteilter Diensterbringung betreffen, unter dem Begriff System-Management zusammengefaßt werden ([Hege93] S103).
In diesem Sinne beschäftigt sich das System-Management insbesondere mit dem Management von verteilter Betriebssystem-Software und u.a. auch mit dem Management von Lastbalancierung und Datenmigration.

[Gosc91] definiert ein verteiltes System als eine Ansammlung mehrerer autonomer Computersysteme (Prozessoren und Datenspeicher, die Prozesse unterstützen), die über eine bestimmte Art von Kommunikations-Subsystem miteinander verbunden sind und logisch in verschiedenen Graden durch ein verteiltes Betriebssystem und / oder ein verteiltes Datenbank-System integriert sind, um ein bestimmtes übergeordnetes Gesamtziel zu erreichen ([Gosc91] S40).
Insbesondere kann also der Begriff "Workstation-Farm" bzw. "Workstation-Cluster" unter dem Begriff "verteiltes System" eingeordnet werden. [KaNe93a] stellt fest, daß "eine lose Ansammlung von Workstations im Netz (das physische und politische Grenzen überschreiten kann) noch kein "Cluster" darstellt. Verschiedene Bezeichnungen wurden verwendet, um eine solche Aktivität zu definieren: "cluster", "farm" oder sogar "kluster". (...) Zur Vermeidung von Unklarheiten definieren wir als "Cluster" eine Ansammlung von Computern im Netz, die als eine einzelne Computer-Ressource funktionieren kann mittels der Verwendung zusätzlicher System-Management Software" ([KaNe93a] S1).
Das verteilte Betriebssystem eines verteilten Systems kann durch den Vergleich mit einem herkömmlichen Betriebssystem wie z.B. UNIX definiert werden. [CDK94] stellt fest, daß ein verteiltes Betriebssystem ähnlich wie ein herkömmliches Betriebssystem eine Ansammlung von Software-Komponenten darstellt, die die Programmierung von Software vereinfacht und die größtmögliche Vielfalt an Anwendungen unterstützt. Anders als gewöhnliche Betriebssysteme sind verteilte Betriebssysteme modular aufgebaut und erweiterbar. Die Modularität des verteilten Betriebssystems basiert dabei auf seiner grundlegenden Fähigkeit, Kommunikation zwischen den einzelnen Modulen zu unterstützen ([CDK94] S5).

Eines der logischen Komponenten eines verteilten Betriebssystems stellt die Ressourcen-Allokation dar. [Gosc91] stellt fest, daß die Ressourcen-Allokation einer der drei Aspekte des Ressourcen-Managements in Betriebssystemen ist (die anderen beiden Aspekte sind Deadlock-Erkennung sowie Ressourcen- und Kommunikations-Sicherheit). In verteilten Umgebungen kann es vorkommen, daß manche Computer "stillstehen", während die Ressourcen anderer Rechner stark ausgelastet sind. Die Migration von Prozessen und Daten von ausgelasteten Rechnern zu gering belasteten Rechnern ermöglicht die Verbesserung der Leistung des verteilten Systems und wird mit Hilfe von Lastbalancierung und Datenmigrations-Systemen durchgeführt.

Lastbalancierung oder allgemeiner Lastverwaltung kann nach [Ludw92] definiert werden als die (kontinuierliche) Zuweisung von Aufträgen an Funktionseinheiten, die ihre Bearbeitung übernehmen, wobei für jeden Auftrag mindestens zwei alternative Funktionseinheiten vorhanden sein müssen, die ihn potentiell ausführen können ([Ludw92] S37). Analog kann Datenmigration als die Zuweisung von Datensätzen an Funktionseinheiten verstanden werden, die diese speichern, wobei für jeden Datensatz mindestens zwei alternative Funktionseinheiten zur Verfügung stehen müssen, die für die Speicherung potentiell in Frage kommen.
In Bezug auf das Leistungsverhalten des Systems kann Lastverwaltung wie bei [ELZ86b]

definiert werden als den Prozeß des Teilens von Rechnerressourcen mittels der transparenten Verteilung von System-Arbeitslast. Die Systemleistung kann dabei verbessert werden, indem Last von stark ausgelastetetn Knoten auf solche verschoben wird, die nur gering belastet sind ([ELZ86b]). Entsprechend umfaßt Datenmigration den Prozeß des Teilens von Speicherressourcen mittels der transparenten Verteilung von Datensätzen zur Optimierung von Speicheraufteilung und Gesamtsystemleistung.

2.3 Lösungs-Problematik der Lastbalancierungsaufgabe

Eine Lastverwaltungskomponente muß eine Optimierungs-Aufgabe erbringen, die als untergeordnete Aufgabe laufend Entscheidungsaufgaben beinhaltet, welche wiederum oft zeitkritisch sind. Eine optimale Verteilung insbesondere von Auftragsgruppen setzt Kenntnisse über zukünftige Entwicklungen voraus, die nicht oder nur unzureichend vorhanden sind. Entscheidungsgrundlagen für optimale Entscheidungen setzen Information voraus, die nicht immer oder genügend schnell eingeholt werden können. Dies führt in der Praxis zu einer Suche nach Lösungen für die Lastbalancierungsaufgabe, die eine optimale Lösung zumindest gut annähern. Eine der Konsequenzen dieser Tatsache ist, daß unterschiedlichste Strategien und Verfahren der Lastverwaltung entwickelt wurden, deren einheitliche Betrachtungsweise durch verschiedenste Strukturierungskriterien und Zielsetzungen erschwert ist. Im Kapitel 2 soll daher zunächst ein einheitlicher Beschreibungsrahmen vorgestellt werden, der eine Einordnung bestehender Lastbalancierungsalgorithmen in bestehende Klassifikationsschemata erlaubt und eine logische Strukturierung vornimmt. Zunächst soll aber ein Lastbalancierungsmodell vorgestellt werden, in das bis auf spezielle Verfahren alle gängigen Algorithmen und Methoden eingeordnet werden können - das Steuerungs- und Regelungsmodell.

2.4 Das Steuerungs- und Regelungsmodell der Lastbalancierung

Nach [Hein90] zeichnen sich kybernetische Systeme vor allem dadurch aus, daß sie nach Störungen, die ihr Gleichgewicht beeinträchtigen, unter bestimmten Bedingungen wieder in einen Gleichgewichtszustand zurückkehren bzw. einen neuen Gleichgewichts-Zustand entwickeln. Zwei dieser kybernetischen Systeme sind das Regelungsmodell sowie das Steuerungsmodell.
Regelung charakterisiert dabei eine Störungskompensation, die auf Rückkoppelung beruht, wogegen Steuerung eine Störungskompensation ohne Rückkoppelung darstellt. Rückkoppelung wird dabei verstanden als ein Prinzip, nach dem das Ergebnis eines Prozesses gemessen und mit dem gewünschten Sollzustand verglichen wird. Eine Korrektur wird eingeleitet, wenn Soll- und Ist-Zustand voneinander abweichen.
Im Falle des Regelungsmodells wird also Information über die zu regelnde Umgebung über einen Rezeptor aufgenommen und einer Vergleichsstelle zugeführt, die die gewonnenen Werte mit einem Soll-Wert vergleicht und das Ergebnis (Abweichung oder Übereinstimmung) einem Regler weiterleitet. Dieser leitet im Falle der Abweichung vom gewünschten Zustand eine Korrekturmaßnahme ein, um das System wieder in den Gleichgewichtszustand über-

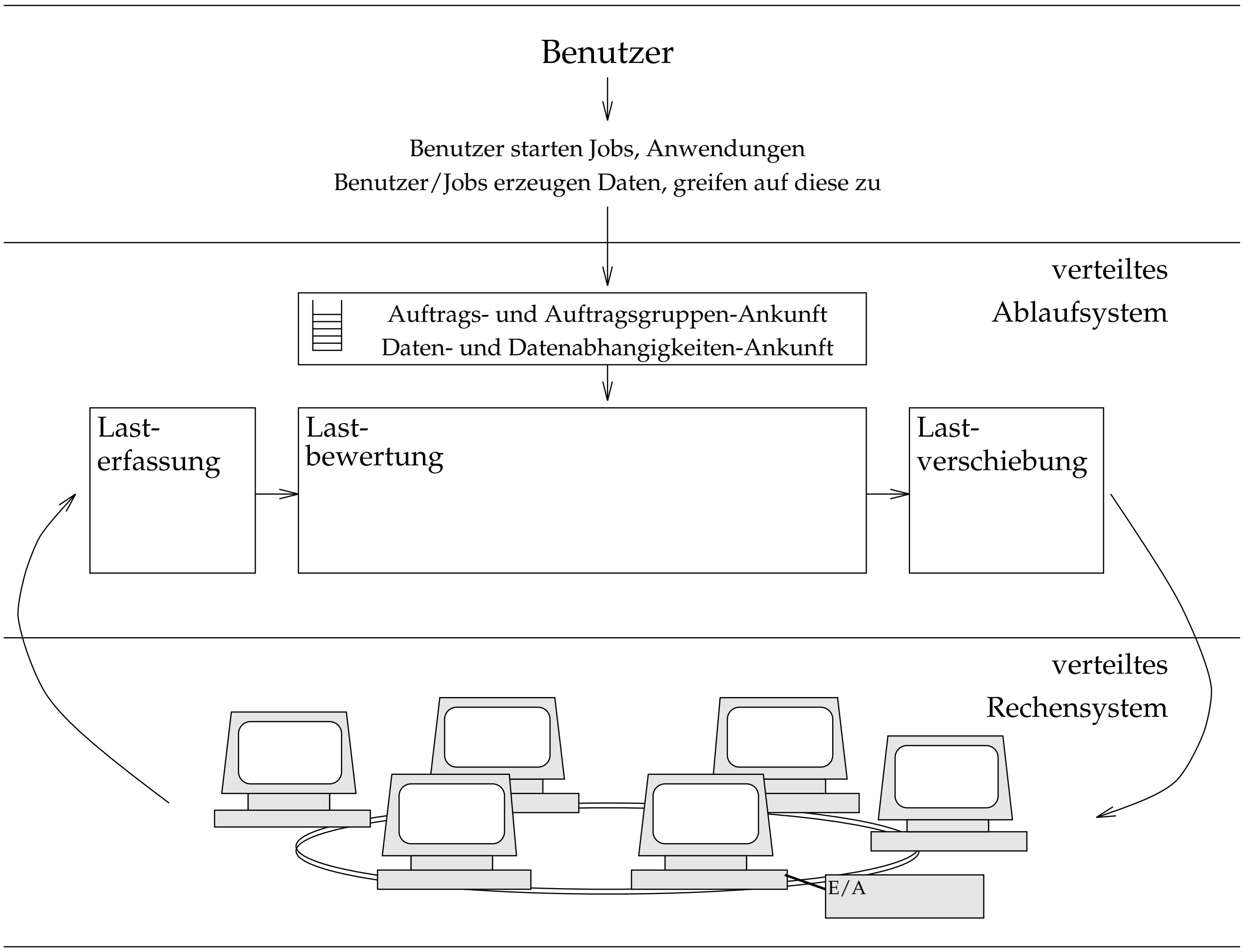

Abbildung 2.1: Regelkreiskomponenten der Lastbalancierung (nach [Ludw92])

zuführen.
Im Falle des Steuerungsmodells wird kein Rückkoppelungsmechanismus verwendet, der Korrekturmaßnahmen zum Zeitpunkt der Störungsentstehung einleitet, sondern mit Hilfe eines Reaktionsprogramms oder Reaktionsmodells Korrekturmaßnahmen vorgenommen, wenn jenes Programm oder Modell das bzgl. bestimmter Informationen aus der Umwelt vorsieht ([Hein90] S21ff). So können Umweltänderungen bereits im voraus kompensiert werden, allerdings nur mit der Genauigkeit, mit der die entsprechenden Reaktionsprogramme und Modelle die zukünftigen Entwicklungen wiederspiegeln.
Kybernetische Modelle wie Regelung und Steuerung können nun, wie dies von [Ludw92] durchgeführt wird, auf das Prinzip der Lastbalancierung und Datenmigration übertragen werden. [Ludw92] bezeichnet die Komponenten Rezeptor, Vergleichsstelle und Regler für die Lastverwaltungs-Aufgabe als "Lasterfassung", "Lastbewertung" und "Lastverschiebung" (vgl. Abbildung 2.1).

Das zu regelnde und steuernde System stellt das Workstation-Cluster zusammen mit dessen Betriebssystem und den Prozessen und Daten dar. Die Beobachtung des Systems erfolgt durch die Lasterfassungskomponente, die die empfangene Information der Lastbewertungskomponente weiterleitet. Diese ist nun aufgrund dieser Information in der Lage, Entscheidungen bezüglich der Verlagerung von Prozessen und Daten im System auszulösen. Eine Lastverschiebekomponente nimmt anhand der getroffenen Entscheidungen entsprechende Aktionen im verteilten Rechensystem vor, um einen die Verteilung von Last betreffenden Gleichgewichtszustand wiederherzustellen bzw. aufrechtzuerhalten.

Kapitel 3

Analyse von Lastbalancierungs - Algorithmen und - Systemen

3.1 Überblick und Analyse verschiedener Lastbalancierungs-Algorithmen

3.1.1 Entwicklung eines einheitlichen Beschreibungsrahmens für Lastbalancierungs-Algorithmen

3.1.1.1 Übersicht

Es soll nun ein einheitlicher Beschreibungsrahmen für Lastbalancierungs- Algorithmen vorgestellt werden, der eine einfache und übersichtliche Klassifizierung verschiedenster in der Literatur vorgestellter Algorithmen erlaubt.
Lastbalancierungs-Mechanismen werden in Anlehnung an [Ezza86] in die drei grundsätzliche Aspekte "Zielfunktion", "verwendete Taktik/Strategie" (Policy) und "Struktur des Lastbalancierungs-Mechanismus" und einen zusätzlichen Aspekt "Annahmen über die Systemumgebung" aufgeteilt. Die Zielfunktion beschreibt, auf welche Kostenfuktion hin der spezifische Algorithmus entwickelt worden ist. Die verwendete Taktik/Strategie umfaßt das eigentliche Vorgehen des Algorithmus bei der Verteilung von Prozessen im System. Durch die Struktur des Lastbalancierungs-Algorithmus schließlich wird beschrieben, welche konkrete Ausprägung der Algorithmus im verteilten System hat.
Da der Lastbalancierungs-Algorithmus grundsätzlich für eine bestimmte System-Umgebung entwickelt worden ist, gibt der Aspekt "Annahmen über die Systemumgebung" die entsprechenden Randbedingungen wieder.

3.1.1.2 Zielfunktion

Da Lastbalancierung immer nur hinsichtlich einer Zielsetzung konzipiert werden kann, stellt diese die erste der drei grundsätzlichen Aspekte der Lastbalancierung dar.Die Zielfunktion repräsentiert die Kostenfunktion des Algorithmus, auf die die Lastbalancierungs-Strategie hin entwickelt worden ist. Generell ist das Ziel der Lastbalancierungs-Strategie, die Rechnerlast transparent für den Benutzer zu verteilen, um die bestmögliche Rechnerleistung zu erzielen und die Performance-Erwartungen der Benutzer zu erfüllen. Diese Aufgabe ist allerdings

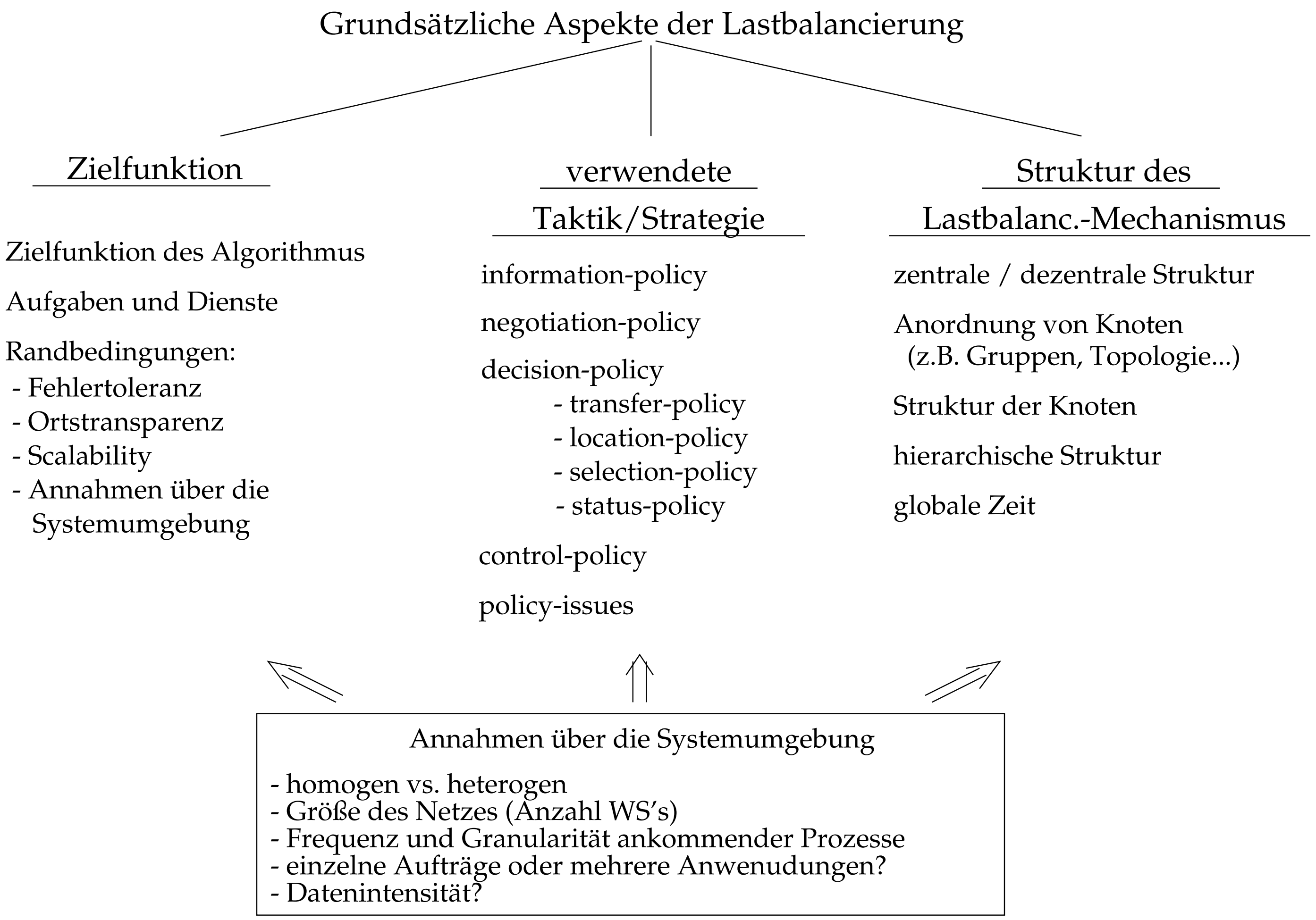

Abbildung 3.1: Einheitlicher Beschreibungsrahmen für Lastbalancierungs-Algorithmen

nicht einfach zu bewältigen, da es einerseits keine unabhängig existierenden Leistungs-Ziele im System gibt und andererseits die Erwartungen der Benutzer bzgl. Performance nicht leicht zu bestimmen sind. ([Gosc91] S495)
[Ludw92] nennt als Minimalziel aller Lastverwaltungsverfahren die Lastverteilung ("load sharing"), bei der "zu keinem Zeitpunkt ein Rechnerknoten untätig werden darf, wenn zu gleicher Zeit auf anderen Knoten Aufträge zur Bearbeitung anstehen." ([Ludw92] S38). Lastausgleich ("load balancing") definiert [Ludw92] als grundsätzliches Lastbalancierungs-Ziel, bei dem gefordert wird, daß darüberhinaus "die Last zwischen allen Rechnerknoten zu jedem Zeitpunkt bezüglich eines gegebenen quantitativen Maßes ausgeglichen ist."
Neben diesen grundsätzlichen Zielfunktionen können folgende Zielfunktionen eine Grundlage für Lastbalancierungs-Algorithmen darstellen:

- Minimierung der Antwortzeit von Aufträgen / Jobs unabhängig voneinander
- Minimierung der System-Antwortzeit ([ELZ86a])
- Minimierung des Mittels der Summe aller Ausführungszeiten der Aufträge einer verteilten Anwendung ([Beck92] bezeichnet ein derartiges Vorgehen in einem Lastbalancierungs-System als "soziale Lastbalancierung")
- Minimierung der Kommunikations-Kosten ([ThLa89])
- Maximierung der Betriebsmittel-Auslastung ([Gosc91])
- Maximierung des Durchsatzes ([Gosc91])
- Minimierung der Auswahlzeitdauer für einen Zielrechner ([ThLa89])
- Minimierung des Overheads für die Lastbalancierung ([ThLa89], [EfGr88])
- Minimierung der Migrationskosten von Prozessen (oder Daten) ([Beck93] S6)

Daneben muß ein Lastbalancierungs-System verschiedene Randbedingungen berücksichtigen, wie z.B. Fehlertoleranz, maximal zu bewältigende Systemgröße, Arten von Aufträgen und Anwendungen, Häufigkeit der Auftrags-Ankunft o.ä. (vgl. auch: "Annahmen zur Systemumgebung", Kapitel 3.1.1.5)

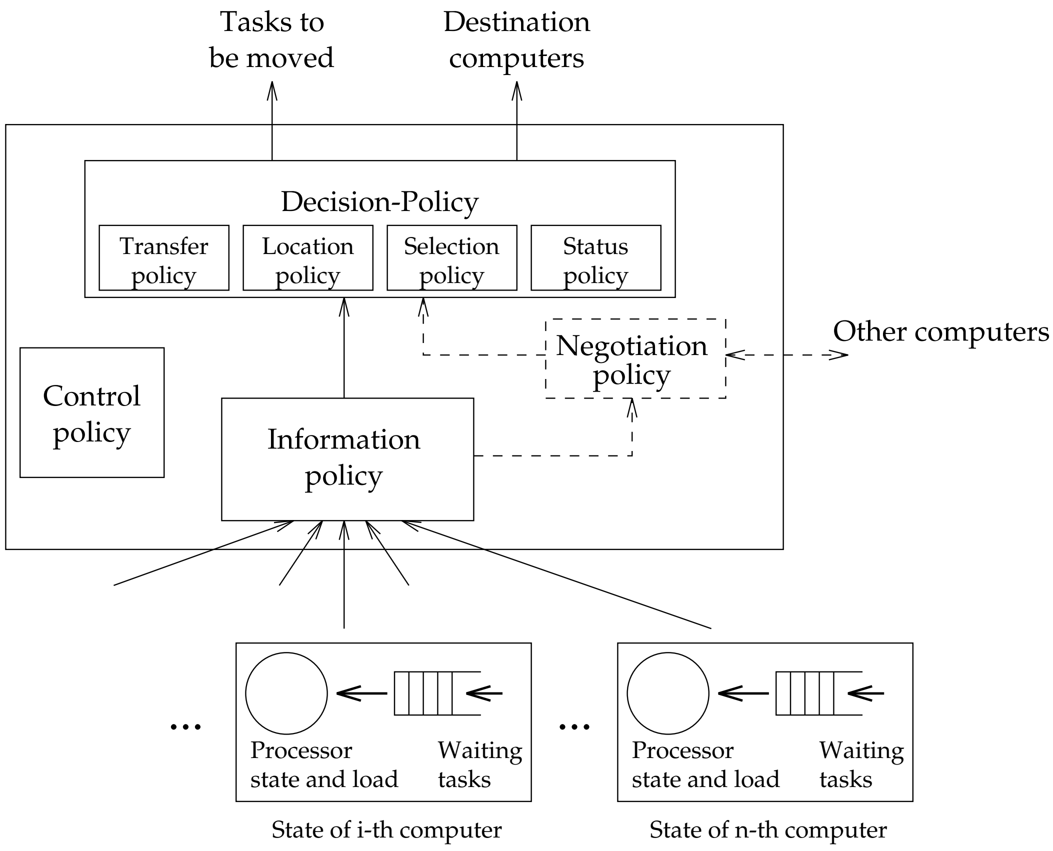

Abbildung 3.2: Strategie-Komponenten der Lastbalancierung (nach [Gosc91] S499)

3.1.1.3 Verwendete Taktik/Strategie

Der wesentliche Teil des Lastbalancierungs-Algorithmus wird durch die in ihm verwendete Taktik (Policy) beschrieben. Folgende Teilstrategien lassen sich gegeneinander abgrenzen:

1. **information-policy**
   Die "information-policy" legt alle Aktivitäten des Algorithmus fest, die die für die Lastbalancierungs-Entscheidung notwendige Information betreffen. Im einzelnen werden hier folgende Fragen beantwortet:

   - Über welche Größen soll Information eingeholt werden? Wie detailliert soll die Information sein?
   - Soll Information aus der Vergangenheit berücksichtigt werden?
   - Wo soll die Information gespeichert werden?
   - Wie soll Information über das System verbreitet werden?
   - Wie soll Information über den Status des Systems eingeholt werden?
   - Wann und wie oft soll System-Information erneuert werden?

   Folgende Klassifizierungen existieren innerhalb der "information-policy":

- Statische versus dynamische Verfahren (1)
(auch: Lastabhängige bzw. Lastunabhängige Verfahren, vgl. [FeZh86]) Nach [Ludw92] und [ELZ86b] versteht man unter dynamischer (adaptiver) Lastverwaltung (1) alle Methoden, die bei der Regelung auf Wissen über den aktuellen Zustand des Systems zugreifen; unter statischen Verfahren (1) versteht man hingegen alle Verfahren, die ohne Wissen über den Systemzustand arbeiten [1].

- Verfahren, die abhängig/unabhängig von globaler Information sind
Nach [Kale88] lassen sich dezentrale Verfahren, die bei ihrer Entscheidung abhängig von globaler Information sind unterscheiden von Verfahren, die lediglich die Information ihrer Nachbarn berücksichtigen.

- Verfahren, die Datenabhängigkeiten berücksichtigen bzw. nicht berücksichtigen
[NiSt93] berücksichtigt Datenabhängigkeiten von Aufträgen, da die Placierung von Tasks relativ zu ihren Daten die Kommunikations-Kosten und damit die Gesamt-Performance der Anwendung beeinflußt.

2. **negotiation-policy**
In dieser logischen Komponente wird die Rollenverteilung und Interaktion zwischen Rechnerknoten bei der Entscheidungsfindung festgelegt und je nach Algorithmus der "decision-policy" eine weitere Entscheidungs-Grundlage geliefert.

- Kooperative versus Nicht-Kooperative Verfahren:
Kooperieren die verteilten Komponenten bei der Entscheidungsfindung, spricht man von Kooperativen Verfahren; treffen die Rechnerknoten unabhängig voneinander ihre Entscheidungen (wobei sich diese Entscheidungen dann gegenseitig stören und Ressourcen-Konflikte verursachen können), so spricht man von Nicht-Kooperativen Verfahren ([Gosc91] S486). Kooperation ist insbesondere wünschenswert in dezentralen Verfahren. (vgl. Kap 3.1.1.4)

- Quellknoteninitiierte Verfahren, Zielknoteninitiierte Verfahren, bzw. Quellknoten- und Zielknoteninitiierte Verfahren sowie Überwacherinitiierte Lastbewertungsverfahren (nach [Ludw92])
Bei Quellknoteninitiierten Lastbewertungsverfahren erfolt der Anstoß zum Durchlauf des vollen Regelkreises stets vom überlasteten Knoten aus. Die Lastbewertungskomponente eines überlasteten Knotens initiiert also die Auswahl möglicher Zielknoten und zur Verschiebung geeigneter Objekte.
Bei Zieknoteninitiierten Verfahren erfolgt der Anstoß zum Durchlauf des vollen Regelkreis-Schemas stets vom unterbelasteten Knoten aus. Bei Quell- und Zielknoteninitiierten Verfahren kann die Initiative sowohl von den überlasteten wie auch von den unterbelastetn Knoten aus erfolgen; bei überwacherinitiierten Verfahren geht die Initiative von dem Knoten aus, der die Lastungleichheit zuerst erkennt, wobei dieser Knoten nicht unbedingt identisch mit den überlasteten oder unterlasteten Knoten identisch sein muß ([Ludw92] S45).

- Quellknotengesteuerte versus Zielknotengesteuerte Verfahren, sowie Quell- und Zielknotengesteuerte bzw. Überwachergesteuerte Lastverschiebungsverfahren

[1] Aufgrund uneinheitlicher Definition der Begriffe "statische Verfahren" und "dynamische Verfahren" werden die Verfahren mit statisch (1), statisch (2) bzw. dynamisch (1), dynamisch (2) - siehe weiter unten - gekennzeichnet.

Man spricht von Quellknotengesteuerten Verfahren, "wenn die Durchführung der Lastverschiebung stets vom überlasteten Knoten aus erfolgt" und von Zielknotengesteuerten Verfahren, " wenn die Durchführung der Laststeuerung stets von einem unterlasteten Knoten aus erfolgt". Entsprechend spricht man von Quell- und Zielknotengesteuerten Verfahren, wenn die Steuerung von beiden Seiten aus erfolgt, und von Überwachergesteuerten Verfahren, wenn die Steuerung auch von anderen als den überlasteten bzw. unterlasteten Knoten aus erfolgt.([Ludw92] S47)

3. **decision-policy**
   Die eigentliche Entscheidung über Prozessverlagerungen wird in der "decision-policy"-Komponente getroffen. Vier Teilaspekte lassen sich unterscheiden:

   - **transfer-policy** ([ELZ86a], [Gosc91] S498)
     Diese Komponente trifft die Entscheidung darüber, ob ein Prozess lokal oder auf einem entfernten Knoten ausgeführt werden soll.
   - **location-policy** ([ELZ86a], [Gosc91] S498)
     Hier wird die Entscheidung über den Zielknoten des zu verlagernden Prozesses getroffen.
   - **selection-policy** ([Gosc91] S389)
     Die Entscheidung darüber, welcher Prozess verlagert werden soll, wird in dieser Teilkomponente getroffen.
   - **status-policy**
     Entscheidungen über den Zeitpunkt der Prozeß-Ausführung, über das Anhalten und Starten sowie das Umverteilen (Checkpointing) von Prozessen werden hier getroffen.

   Jede der angegebenen Teilstrategien der "decision-policy" kann eine eigene "information-policy" und "negotiation-policy" besitzen.
   Für Algorithmen der Datenmigration lassen sich diese Strategie-komponenten vergleichbar behandeln. Innerhalb der "status-policy" würden Entscheidungen über Replikation und Partitionierung bzw. Löschung/Ungültigmachen von Datensätzen behandelt; die "selection-policy" würde Entscheidungen bzgl. Partitions-Größen und Datensatz-Auswahl behandeln; durch die "transfer-policy" würde entschieden werden, ob Daten verlagert werden sollen oder nicht; die "location-policy" schließlich würde den Zielrechner für den zu verlagernden Datensatz bestimmen.
   Lastbalancierungs-Algorithmen lassen sich innerhalb der "decision-policy" wie folgt klassifizieren:

   - Lastausgleichs-Verfahren versus Lastverteilungsverfahren:
     ("load-balancing" versus "load-sharing")
     Nach [Ludw92] spricht man von Lastverteilung ("load-sharing"), wenn "zu keinem Zeitpunkt ein Rechnerknoten untätig werden darf, wenn zur gleichen Zeit auf anderen Knoten Aufträge zur Bearbeitung anstehen.", wogegen bei Lastausgleichsverfahren ("load-balancing") "zu jedem Zeitpunkt bezüglich eines gegebenen quantitativen Maßes" die Last ausgeglichen sein muß ([Ludw92] S38)

- Statische versus Dynamische Verfahren (2) [2]
  ("Rechnerzuteilung", "mapping" nach [Ludw92] S39)
  Wird der Ort der Ausführung für alle Prozesse vor ihrem Start festgelegt, spricht man von statischer Lastbalancierung (2), oder nach [Ludw92] von Rechnerzuteilung; wird hingegen die Verteilung dynamisch während der Laufzeit des Systems vorgenommen (und dabei Laufzeitinformation berücksichtigt), spricht man von adaptiven oder dynamischen Verfahren (2). ([Kale88], [Beck92] S23ff)
- Optimale versus Suboptimale Verfahren:
  Ist der Algorithmus in der Lage, bei vollständig verfügbarer Information über Systemzustand und Betriebsmittelumfang bzgl. der Zielfunktion die Zuteilung optimal durchzuführen, spricht man von Optimalen Verfahren; kann aufgrund des erforderlichen Aufwandes keine optimale Lösung gefunden werden, sucht man stattdessen mit Hilfe eines suboptimalen Verfahrens nach einer "guten" Lösung.
- Approximative und heuristische Verfahren:
  Approximative und heuristische Verfahren sind zwei Klassen von Suboptimalen Verfahren. ([Gosc91] S484f)
- Deterministische und Probabilistische Verfahren:
  Spezialfälle für Statische Verfahren (1) sind deterministische und probabilistische Verfahren. Deterministische Verfahren verwalten "Lasten nach einem festen, deterministischen Algorithmus", während Probabilistische Verfahren Wahrscheinlichkeiten verwenden, "um den Zielknoten zur Zuteilung von Last auszuwählen." ([Ludw92] S40)
- Preemptive versus Nicht-preemptive Verfahren:
  Können laufende Prozesse vor ihrer Beendigung umverteilt werden werden, spricht man von Preemptiven Verfahren, sonst von Nicht-preemptiven Verfahren. ([FeZh86])

4. **control-policy**
   Innerhalb dieser Teilkomponente werden Fragestellungen bzgl. Fehlertoleranz abgehandelt, wie z.B. mögliche Instabilität des Algorithmus ("processor-thrashing", vgl. [Ezza86] S1140, [ELZ86a]). Dies kann als rudimentäre, implementierte Management-Funktionalität betrachtet werden.

5. **policy-issues**
   Neben diesen Teilaspekten der in Lastbalancierungs-Verfahren verwendeten Strategien existieren noch Gesichtspunkte, die für diese Komponente insgesamt gelten:

   - Fairness ([ThLa89], [LiLi88] S107, [KaNe93a] 4.2)
     Der faire Zugriff auf entfernte Ressourcen betrifft vor allem die Berücksichtigung der Arbeitslast der Benutzer, um zu verhindern, daß kurze Aufträge lange hinter langlaufenden Jobs auf Ausführung warten müssen.
   - Maß an Komplexität ([ELZ86a])
     In [ELZ86a] untersuchen Eager, Lazowska und Zahorjan den geeigneten Level an Komplexität für Lastverteilungsverfahren. (z.B. Maß an benötigter Last-Informationen oder Komplexität des Algorithmus) Ihr Ergebnis ist dabei, daß

[2] vgl. statische und dynamische Verfahren (1): Aufgrund uneinheitlicher Definitionen in der Literatur sollen die Klassifizierungen durch (1) bzw. (2) unterschieden werden

einfache Strategien in der Praxis die größte Leistungssteigerung im Verhältnis zum Aufwand bieten.

- Kosten der Lastbalancierung
  Welchen Aufwand an Ressourcen verbraucht der Lastbalancierungs- Algorithmus für seine Arbeit? (gemessen in Zeiteinheiten, Kommunikations-Kosten, Speicher-Bedarf und CPU-Auslastung)

3.1.1.4 Struktur des Lastbalancierungs-Mechanismus

Die Struktur des Lastbalancierungs-Mechanismus gibt seine Realisierung im verteilten System wieder. Folgende Struktur-Merkmale sind von besonderer Bedeutung:

1. Zentrale versus Dezentrale Verfahren
   Je nachdem, ob die Informationshaltung und Entscheidungfindung über die zu verteilenden Prozesse zentral von einem einzelnen Rechnerknoten aus erfolgt oder dezentral durch mehrere Rechner des Systems, spricht man von zentralen bzw. dezentralen Verfahren.
   Vorteile und Nachteile zentraler Verfahren gegenüber dezentralen Verfahren (nach [Beck93] S23 sowie [LiLi88] S105f):
   Eine zentrale Informations- und Entscheidungsstelle bietet den Vorteil einer global optimalen Balancierung. Bei diesem Ansatz kann effektiv entschieden werden, welcher Job als nächstes verteilt werden soll, da jeder zu verschickende Job im zentralen Koordinator registriert ist. Allerdings kann eine zentrale Verteilungsstelle leicht zum Engpaß werden, der maximal verwaltbare Pool an Rechnerknoten ist nicht leicht erweiterbar und das System ist leichter fehleranfällig, da eine einzige Komponente für die Verwaltung des Clusters verantwortlich ist.
   Demgegenüber ist ein dezentrales Verfahren fehlertoleranter und leichter auf eine große Zahl an Rechnerknoten ausdehnbar. Dafür sind diese Algorithmen nicht in der Lage, eine globale Optimierung zu erzielen, da die einzelnen an der Entscheidung beteiligten Knoten nur lokal operieren und sich auf einfache Meß- und Entscheidungsgrößen beschränken. Auch können keine komplexeren Zusammenhänge wie Reihenfolgebeziehungen oder soziale Lastbalancierung erfasst werden. ([Beck93] S23)

2. hierarchische oder nicht-hierarchische Strukturen
   Bei großen Systemen kann eine hierarchische Anordnung der Lastbalancierung vorteilhaft sein, bei der lokale Teilkomponenten autonom jeweils ein Teilnetz verwalten und eine übergeordnete Lastbalancierungskomponente als verbindendes Element dient, indem jede lokale Teilkomponente diese als weitere Komponente ihres Teilsystems betrachtet. ([Beck93] S23)

3. Anordnung der Konten
   Bei dezentralen Verfahren spielt häufig die Anordnung der Knoten für die Prozeß-Verteilung eine wesentliche Rolle. Beispiele für mögliche Anordnungen sind Netz ("grid"), Hülle ("wrap"), Würfel oder Quader ("cube") bzw. mehrdimensionaler Quader ("hypercube"). ([LiKe87]) [Ludw92] spricht in diesem Zusammenhang von minimalem, begrenztem, oder maximalem Wirkungsbereich. Dabei wird "Wirkungsbereich einer lastbewertenden Komponente" definiert als derjenige Teil des Systems, innerhalb dessen diese Komponente Lastverwaltung durchführen kann ([Ludw92] S42). Kann

die Komponente auf nur zwei Rechnerknoten lastverwaltend zugreifen (logische Nachbarn), so ist der Wirkungsbereich minimal, kann sie auf das gesamte System einwirken, ist er maximal; in allen anderen Fällen ist der Wirkungsbereich begrenzt.

4. Struktur der Knoten
   Wesentlich für die Beschreibung der Struktur eines Lastbalancierungs- Mechanismus ist weiterhin, wie jeder Knoten aus der Sicht der Lastbalancierung aufgebaut ist. Beispiele für Strukturmerkmale sind Aufbau von Master-Scheduler und Clients eines zentralen Verfahrens, Algorithmen-spezifische Teilkomponenten, Warteschlangen, besondere Variablen o.ä.

5. Globale Zeit: Synchrone versus Asynchrone Verfahren ([HLM+90])
   In einem synchronen Netzwerk hat jeder Knoten eine Uhr, die mit allen anderen Uhren aller anderer Knoten synchronisiert ist. Das Netzwerk hat eine globale Zeit. In einem asynchronen Netzwerk wird eine solche globale Zeit nicht vorausgesetzt. (DCE beispielsweise bietet mit dem Distributed Time Service eine globale Zeit an)

3.1.1.5 Annahmen über die System-Umgebung

Da ein Lastbalancierungs-Algorithmus immer unter bestimmten Annahmen über die Systemumgebung entwickelt worden ist, bilden folgende Angaben die Rahmenbedingungen und Beschränkungen, unter denen der Algorithmus arbeiten kann.

1. Homogene versus Hegerogene Umgebung
   Viele der in der Literatur beschriebenen Algorithmen gehen implizit von einer homogenen Systemumgebung aus (z.B. [Kale88], [BrFi81], [ELZ86a] etc.) und nur wenige Authoren machen Angaben über die Anpassungsmöglichkeiten an heterogene Umgebungen (z.B. [LiKe87], [StSi84]). Eine heterogene Umgebung setzt einerseits Wissen über die System-Umgebung voraus (Prozessoren, Speicher, Betriebssysteme, Kommunikations-Verbindungen), andererseits auch über die von den Prozessen benötigten Ressourcen, damit eine geeignete Verteilung möglich ist.

2. Größe des Rechner-Pools/Anzahl verwaltbarer Rechnerknoten
   Je nach der Konzeption der Algorithmen ist eine Erweiterung des an der Lastverteilung beteiligten Rechner-Pools leicht möglich bzw. existiert eine obere Schranke an maximal verwaltbaren Rechner-Knoten (vgl. v.a. dezentrale und zentrale Verfahren).

3. Frequenz der am Lastbalancierungs-System ankommenden Prozesse/Durchschnittliche zu bewältigende Last
   Insbesondere Unterschiede in der Performance zeigen, ob Lastbalancierungs-Algorithmen nicht nur bei geringer oder mäßiger Last gut arbeiten, sondern auch bei durchschnittlich sehr hoher Auslastung der Prozessorknoten ([EfGr88]).

4. Granularität der Prozesse
   Je nachdem, ob Größe und Ausführungsdauer der zu balancierenden Prozesse stark schwankt oder ob von einer relativ homogenen Struktur der Prozesse ausgegangen wird, sind entsprechende Berücksichtigungen in der Algorithmenstruktur vorzunehmen, bzw. weisen unterschiedliche Algorithmen entsprechende Performance-Unterschiede auf.

5. Art der zu verarbeitenden Aufträge: Prozesse, Auftrags-Gruppen, verteilte Anwendungen
   Sollen nicht nur unabhängige Aufträge auf das System verteilt werden, sondern verteilte Anwendungen, so sind Reihenfolgebeziehungen und Abhängigkeiten zwischen Prozessen zu berücksichtigen, können wiederholte Ausführungen von Aufträgen in die Balancierung miteinbezogen werden und kann allgemein die Lastverwaltungs-Strategie globaler auf die Minimierung der Gesamtlaufzeit hin konzipiert werden.

6. Berücksichtigung von Datenintensität?
   Ähnlich wie die Berücksichtigung von Querbezieungen in verteilten Anwendungen ergibt die Balancierung von Aufträgen, die häufig auf Datensätze zugreifen oder große Datenmengen erzeugen, zusätzliche Aspekte, die für eine Leistungsoptimierung des Algorithmus wesentlich sein können.

7. Existenz einer globalen Zeit?
   Ist in einem Netzwerk eine globale Zeit vorhanden, können Verfahren eingesetzt werden, die eine solche Möglichkeit voraussetzen. (vgl. synchrone und asynchrone Verfahren: [HLM+90])

3.1.2 Beispiele verschiedener Lastbalancierungs-Algorithmen

3.1.2.1 "Random", "Threshold" und "Shortest"

Die von Eager, Lazowska und Zahorjan in [ELZ86a] vorgestellten Ansätze stellen eine der grundlegenden Arbeiten bezüglich Lastverwaltung dar.
Zentraler Aspekt der vorgestellten Algorithmen ist nicht eine spezielle Zielfunktion oder eine besondere Struktur des Lastbalancierungs-Mechanismus. Es wird in erster Linie die Auswirkung verschiedener Komplexitätsgrade der Algorithmen auf die System-Antwortzeit untersucht. Daher sind auch die Annahmen über die System-Umgebung extrem einfach und abstrakt gehalten. Alle Verfahren sind dezentral organisiert und arbeiten in einer homogenen Umgebung. Kommunikationskosten, Granularität von Prozessen o.ä werden nicht berücksichtigt. Es wird in erster Linie die Taktik/Strategie der Algorithmen untersucht, und hier vor allem die "decision-policy".

Die Algorithmen im einzelnen:

- **Random**

  Idee : Zufällige Auswahl von Knoten

  Strategie : Falls die Länge der Warteschlange größer als ein Schwellwert T ist, sende lokalen Job an zufällig gewählten Knoten

  Algorithmus :

  if local_node_queue_length < threshold T
    then process_task_locally;
    else select_node_at_random;

```
        transfer_task;
    fi
```

- der Empfänger-Knoten behandelt ankommende Aufträge in der selben Weise
- zur Beschränkung der maximal möglichen Verschiebungen wird zusätzlich das "transfer limit" L_t berücksichtigt

- **Threshold**

  Idee : Auswahl eines Knotens mit geringer Last

  Strategie : Falls die Länge der Warteschlange größer als ein Schwellwert T ist, suche zufällig (maximal L_p mal) Knoten, dessen Last geringer als ein Schwellwert T ist.

  Algorithmus :

```
if local_node_queue_length > threshold T
   then while (no_of_probes < probe_limit_Lp)
             do  select_node_at_random
                 if transfer_of_task_from_here_to_node ⇒ remote_queue_length > threshold T
                    then goto_while
                    else transfer_task; goto_end
                 fi
             od
   else process_locally
   :end
fi
```

- **Shortest**

  Idee : Auswahl des Knotens mit geringster Last

  Strategie : Falls die Länge der Warteschlange größer als ein Schwellwert T ist, wähle L_p mal Knoten zufällig aus; sende Job an Knoten mit geringster Last, falls diese kleiner als ein Schwellwert T ist.

  Algorithmus :

```
if local_node_queue_length > threshold T
   then while (no_of_probes < probe_limit_Lp)
             do  select_node_at_random
                 ql_i = remote_queue_length
             od
             if min(ql_i) < threshold T
                then transfer_task_to_node min(ql_i)
                else process_locally
             fi
   else process_locally
fi
```

Wichtige Resultate sind u.a., daß in vielen Fällen eine zufällige Knoten-Auswahl eine deutlich bessere System-Antwortzeit liefert als gar keine Lastverschiebung, daß der "Threshold" Algorithmus eine deutliche Verbesserung des Leistungsverhalten gegenüber dem "Random" Algorithmus zeitigt, sowie, daß die Leistung des "Shortest" Algorithmus nicht signifikant besser ist, als die des einfacheren "Threshold" Algorithmus. Darüberhinaus kann festgestellt werden, daß der Schwellwert T nicht stark von der Systemlast abhängt und man bei der Wahl der Größe L_p bzw. L_t schon mit kleinen Werten gute Leistung erzielen kann.
Einordnung in den Beschreibungsrahmen:

Algorithmus: Random, Threshold, Shortest	
Zielfunktion	Verbesserung der System-Antwortzeit **Verfahren: Lastausgleich (load sharing)**
Randbedingungen	keine Angaben zu Randbedingungen
Taktik/Strategie	
information policy	Information über die Warteschlangen-Länge der entfernten Rechnerknoten **Verfahren:** Random: **statisch (1)**, Threshold, Shortest: **dynamisch (1)**
negotiation policy	keine Kooperation zur Entscheidungsfindung **Verfahren: quellknoteninitiiert, quellknotengesteuert**
decision policy	**Verfahren: dynamisch (2), suboptimal, approximativ**
transfer	"threshold policy", die lediglich lokale Information benutzt; im Algorithmus durch die äuserste if-Klammer repräsentiert
location	durch das Innere der if-Klammerung repräsentiert
selection	keine Angaben zur selection-policy
status	keine Angaben zur status-policy
policy issues	unterschiedliche Algorithmen-Komplexität
Struktur-Merkmale	keine spezifischen Struktur-Merkmale **Verfahren: dezentral, asynchron**
Ann. über Systemumgeb.	Homogene, einfach gehaltene Systemumgebung

3.1.2.2 "Sender Policy", "Receiver-Policy", "Reservation-Policy"

In [ELZ86b] stellen Eager, Lazowska und Zahorjan drei Algorithmen vor, die in erster Linie zur vergleichenden Untersuchung von quellknoten-initiierten Verfahren mit zielknoten-initiierten Verfahren dienen. Demzufolge sind die Annahmen über die System-Umgebung entsprechend einfach gehalten. Die Algorithmen sind der Klasse der dezentralen Verfahren zuzurechnen.

- **Sender Policy**

  Idee : Sender belastet unterbelastete Knoten

  Strategie : Falls die Länge der Warteschlange des Sender-Knotens größer als ein Schwellwert T ist, suche einen Knoten, der so wenig Last hat, daß auch der Transfer nicht zu einer Last größer als T führt (Anzahl der Versuche: Test-Limit L_p)

Anmerkung Das Verfahren ist mit dem "Threshold"-Algorithmus des vorigen Kapitels identisch.

Algorithmus :

```
if my_node_queue_length > threshold T
   then while (no_of_probes < probe_limit_L_p)
            do  select_node
                if transfer_of_task_from_here_to_node ⇒ node_above_threshold_T
                   then goto_while
                   else transfer_task; goto_end
                fi
            od
          :end
fi
```

- **Receiver Policy**

  Idee : Empfänger entlastet überladene Knoten

  Strategie : Falls die Länge der Warteschlange des Empfänger-Knotens kleiner als ein Schwellwert T ist (nach der Beendigung eines Jobs), suche einen Knoten, der so viel Last hat, daß auch der Transfer nicht zu einer Last kleiner als T führt. (Anzahl der Versuche: Test-Limit L_p)

  Algorithmus :

```
if after_completing_task
   my_node_queue_length < threshold T
   then while (no_of_probes < probe_limit_L_p)
            do  select_node
                if transfer_of_remote_task_from_node_to_here ⇒ node_below_threshold_T
                   then goto_while
                   else transfer_task; goto_end
                fi
            od
          :end
fi
```

  Problem : Es ist notwendig, einen schon gestarteten Job zu transferieren.

- **Reservation Policy**

  Idee : Verbesserung des "Receiver Policy"-Algorithmus

  Strategie : Falls die Länge der Warteschlange des Empfänger-Knotens kleinder als ein Schwellwert T ist (nach der Beendigung eines Jobs), reserviere einen Knoten, der bei der Ankunftszeit des nächsten Jobs so viel Last hat, daß auch der Transfer nicht zu einer Last kleiner als T führt - und für den noch keine Reservierung vorliegt. (Anzahl der Versuche: Test-Limit L_p)

Algorithmus :

```
if after_completing_task
   my_node_queue_length < threshold T
   then while (no_of_probes < probe_limit_Lp)
            do  select_node
                if transfer_of_next_originating_remote_task_from_node_to_here
                      ⇒ node_below_threshold_T_at_arrival_time
                      or other_reservation_pending
                   then goto_while
                   else reserve_remote_task_for_me; goto_end
                fi
            od
         :end
fi
```

Einordnung in den Beschreibungsrahmen:

Algorithmus: Sender-Policy, Receiver-Policy, Reservation-Policy	
Zielfunktion	Verbesserung der durchschnittlichen Job-Antwortzeit **Verfahren: Lastausgleich (load sharing)**
Randbedingungen	keine Angaben zu Randbedingungen
Taktik/Strategie	
information policy	Information über die Warteschlangen-Länge der entfernten und lokalen Rechnerknoten (Reservation zusätzlich: "Reservierungen" des Testknotens) Einholen der Information je nach Initiierungs-Verfahren **Verfahren: dynamisch (1)**
negotiation policy	Keine Kooperation zur Entscheidungsfindung **Verfahren:** Sender-Policy: **quellknoteninitiiert, quellknotengesteuert** Receiver-Policy, Reservation-Policy: **zielknoteninitiiert, zielknotengesteuert**
decision policy	**Verfahren: dynamisch (2), suboptimal, approximativ**
transfer	"threshold policy", die lediglich lokale Information benutzt; im Algorithmus durch die äuserste if-Klammer repräsentiert
location	durch das Innere der if-Klammerung repräsentiert
selection	keine Angaben zur selection-policy
status	keine Angaben zur status-policy
policy issues	keine
Struktur-Merkmale	keine spezifischen Struktur-Merkmale **Verfahren: dezentral, asynchron**
Ann. über Systemumgeb.	Homogene, einfach gehaltene Systemumgebung

3.1.2.3 Das Gradientenverfahren von Lin/Keller

F. Lin und R. Keller stellen ein dezentrales, asynchrones, dynamisches Lastbalancierungs-Verfahren vor, das auf dem "Gradienten-Modell" beruht. Dabei werden neu entstehende Jobs über mehrere Rechnerknoten hinweg weitergereicht und einem Knoten mit geringer Ressorucenlast zugeführt, indem der Job mittels einer Gradientenoberfläche den kürzesten Weg innerhalb der vorhandenen (logischen) Knotentopologie wählt.

- Struktur des Lastbalancierungs-Mechanismus:
  Das Gradienten-Modell-Lastbalancierungs-Verfahren ist ein dezentrales Verfahren, bei dem die Rechnerknoten (logisch) als Netz, Hülle, Würfel, Quader oder mehrdimensionaler Quader etc. angeordnet sind.
  Eine Gradienten-Oberfläche $[w_1 ... w_n]$ wird über diese Topologie gelegt und gibt für jeden Konten i an, wie weit dieser vom nächsten gering ausgelasteten Rechner entfernt ist (Beispiel: Liegt zwischen einem Rechner i und einem leicht ausgelasteten Rechner j nur ein weiterer Rechner, so würde die Gradienten-Oberfläche für den Rechner i w_i=2, d.h. 2 Schritte angeben). Rechner mit geringer Ressourcenlast selbst erhalten w_i=0.
  Da jeder Knoten nur seine eigenen Werte der Gradienten-Oberfläche sowie die seiner Nachbarn kennt, die Oberfläche aber aufgrund von Prozess-Verschiebungen Änderungen unterworfen ist, müssen für die Konsistent-Haltung geeignete Verfahren gefunden werden. Man führt dazu eine Durckoberfläche $[p_1 ... p_n]$ ein, die eine Annäherung an die Gradientenoberfläche darstellt. Wird ein Knoten als gering belastet eingestuft und kann noch Aufträge annehmen, so erhält er den Durckwert 0. Alle Knoten beobachten die Druckwerte ihrer Nachbarn und setzen bei einer Änderung ihren Druckwert auf den um eins erhöhten Wert des Nachbarn mit dem kleinsten Druckwert. So breitet sich eine Änderung in der Druckoberfläche über alle Knoten hinweg aus und liefert als stationären Zustand genau die Gradientenoberfläche.

- Taktik/Strategie des Mechanismus:
  Die Entscheidung über die Verlagerung eines Tasks (transfer-policy) wird implizit über w_i=0 getroffen; die Entscheidung über den Zielknoten (location-policy) wird verteilt mittels der Druckoberfläche festgelegt. Gilt p_i=diam(Network) für alle Knoten i, findet keine Lastbalancierungsaktivität statt ("Saturation"). Eine "selection-policy" ist nicht vorhanden. Die Entscheidung über die Verlagerung von Prozessen wird mittels Kooperation mit den Nachbarknoten getroffen, die Lastbewertung wird vom Zielknoten initiiert und die Verlagerung wird vom Sender gesteuert.
  Wesentliche für die Entscheidung benötigte Größen sind dabei:

  - der Lastzustand auf jedem Knoten ("geringe Last" = kann zusätzlich Last aufnehmen; "mittelschwere Last", oder "ausgelastet" = möchte Last abgeben)
  - die Werte der Druckoberfläche
  - die Topologie des Rechnernetzes; die Nachbarn jedes Knotens

- Annahmen über die Systemumgebung:
  Das Grundmodell dieses Algorithmus arbeitet in homogener Umgebung und berücksichtigt keine weiteren (erschwerenden) System-Annahmen. Jedoch werden Angaben für die Anpassung des Modells an heterogene Umgebungen gemacht, die abgewandelt

auch für zusätzliche Kriterien wie Granularität der Prozesse oder Berücksichtigung von Datenabhängigkeiten geeignet sein könnten:

- Sind unterschiedliche Prozessor-Typen im Netz vorhanden, und müssen speziel-le Aufträge durch bestimmte Prozessor-Typen ausgeführt werden, könnten z.B. meherer Gradientenoberflächen eingeführt werden, oder Gruppen-Adressierungen berücksichtigt werden.
- Ist die Prozessor-Leistung von Knoten zu Knoten unterschiedlich, so kann dies durch Normalisierung der Werte "geringe Last", "mittelschwere Last" bzw. "ausgelastet" ausgeglichen werden.
- Unterschiedliche Kosten der Verwendung von Kommunikations-Verbindungen kann durch Veränderung der Druckanpassungs-Funktion gelöst werden:
  Anstelle von:

  $$p_i = \begin{cases} 0 & \text{falls } w_i = 0 \text{ (Knoten geringer Last)} \\ p_j + 1 & \text{sonst (mit j Nachbar mit kleinstem Druckwert)} \end{cases}$$

  jetzt (mit c_{ij} = Kommunikations-Kosten zwischen Knoten i und j)

  $$p_i = \begin{cases} 0 & \text{falls } w_i = 0 \text{ (Knoten geringer Last)} \\ c_{ij} + p_j & \text{sonst (mit j Nachbar mit kleinstem Druckwert)} \end{cases}$$

- Fehler-Toleranz:
  Es darf angemerkt werden, daß das Gradienten-Verfahren dann sehr fehlerkritisch bzgl. dem Ausfall von Knoten ist, wenn nicht z.B. mittels eines Polling-Verfahrens der Nachbarknoten vor Absendung von Jobs die Funktions-Fähigkeit des Zielknotens überprüft wird

- Einordnung in den Beschreibungsrahmen:

Algorithmus: Gradienten-Verfahren nach Lin und Keller	
Zielfunktion	globaler Lastausgleich **Verfahren: Lastausgleich (load balancing)**
Randbedingungen	Keine Angaben zu weiteren Randbedingungen
Taktik/Strategie	
information policy	Information über Nachbarknoten, deren Lastzustand sowie ihren Druckwert **Verfahren: dynamisch (1)**
negotiation policy	Kooperation der Komponenten **Verfahren: zielknoteninitiiert, quellknotengesteuert**
decision policy	**Verfahren: dynamisch (2)**
transfer	"transfer-policy" implizit durch die Bestimmung von $w_i = 0$
location	"location-policy" verteilt mittels der Druckoberfläche
selection	"selection-policy" nicht vorhanden
status	"status-policy" nicht vorhanden
policy issues	keine
Struktur-Merkmale	logische Anordnung der Knoten als Netz, Hülle, Quader oder mehrdimensionaler Quader begrenzter Wirkungsbereich **Verfahren: dezentral, asynchron**
Ann. über Systemumgeb.	homogene und heterogene Umgebungen möglich Auch in Multiprozessor-Systemen mit einer großen Anzahl an Rechnerknoten gut einsetzbar Berücksichtigung weiterer System-Aspekte gut möglich

3.1.2.4 Der Up-Down Algorithmus von Mutka und Livny

In [MuLi87] beschreiben Mutka und Livny einen Algorithmus, der mit dem Ziel entwickelt wurde, den Zugriff auf freie Ressourcen entfernter Rechnerknoten nach bestimmten Fairness-Kriterien zu gestalten. Das bedeutet, daß Benutzer, die Ressourcen nur wenig belasten, nicht zu lange auf deren Zugriff warten müssen sollten, wohingegen Benutzer, die Ressourcen sehr intensiv beanspruchen, nicht alle verfügbaren Ressourcen über lange Zeit besetzt halten sollten. Dies wird erreicht durch die Verwendung des Up-Down Algorithmus. Dieser Algorithmus stellt sicher, daß Dauer und Umfang der Allokation entfernter Ressourcen darüber entscheiden, wieviele weitere entfernte Ressourcen alloziiert werden können und wie wahrscheinlich es ist, daß alloziierte Ressourcen zugunsten anderer Knoten von einem Rechner abgezogen werden.

- Struktur des Mechanismus:
  Die Allokation von entfernten Ressourcen wird über einen zentralen Koordinator abgewickelt, der mittels des Algorithmus jedem Rechner über eine Allokations-Tabelle eine gewisse Priorität zuweist. Diese Allokations-Tabelle wird "schedule index"-Tabelle genannt; die Einträge heißen "schedule indices" SI[i] für alle Rechner i und repräsentieren die zugewiesenen Prioritäten. Diese Prioritäten geben an, welchen Vorrang ein Rechner bei der Zuweisung entfernter Rechenleistung hat. Eine hohe Priorität der Zuweisung entspricht einer kleinen Zahl (evtl. kleiner Null) in der Allokations-Tabelle; eine niedrige Priorität entspricht einer großen Zahl für SI[i]. Gilt SI[i]<SI[j], so hat der Rechner i Priorität über den Rechner j. Bei SI[i]=SI[j] wird eine zufällige Auswahl getroffen.

- Taktik/Strategie des Algorithmus:
  Der Algorithmus wird in jedem Intervall *SchedInterval*, bei jeder Status-Änderung eines Rechners von "nicht verfügbar" nach "verfügbar" und nach jeder Ausführungs-Beendigung entfernter Jobs angestoßen. Dann durchläuft er zwei Schritte: In einem erste Schritt wird die eigentliche Allokation von Rechenleistung an wartende Rechnerknoten vorgenommen. Im zweiten Schritt findet die Modifikation der "schedule index"-Tabelle statt.
  Bei der Allokation entfernter Rechenleistung wird zunächst die frei verfügbare Kapazität an Rechner mit niedrigster Priorität zugewiesen. Die verbleibenden Nachfrager nach Rechenleistung konkurrieren mit Rechnern, die bereits Rechenkapazität alloziiert haben. Ist SI[i] eines nachfragenden Rechners i kleiner als SI[j] eines Rechners j mit alloziierter Kapazität, so wird dessen Kapazität abgezogen (evtl. laufende Jobs angehalten und gecheckpointed) und dem Rechner i höherer Priorität zugewiesen.
  Nach dem Allokations-Schritt erfolgt die Modifikation der "schedule indices" gemäß den Veränderungen in Allokation und Lastverteilung. Rechnerknoten, die wenig entfernte Ressourcen über kurze Zeit beanspruchen und jetzt die Allokation erhöhen bzw. verlängern, werden in ihrer Priorität herabgesetzt (der "schedule index" wird erhöht); Rechner, die entfernte Kapazitäten lange bzw. in hohem Maße verwenden, und jetzt ihre Last herabsetzen, werden in ihrer Priorität angehoben (SI[i] wird verringert). Die Modifikation der "schedule indices" im zeitlichen Verlauf gibt Abbildung 3.3 wieder.

  Man kann erkennen, daß mittels einer Funktion g(SI[i]) der Index immer kleiner wird, je länger ein Rechner auf Allokation warten muß, bzw. daß der Index durch die Funktion f(SI[i]) immer größer wird, je länger Speicher in Anspruch genommen wird. Je

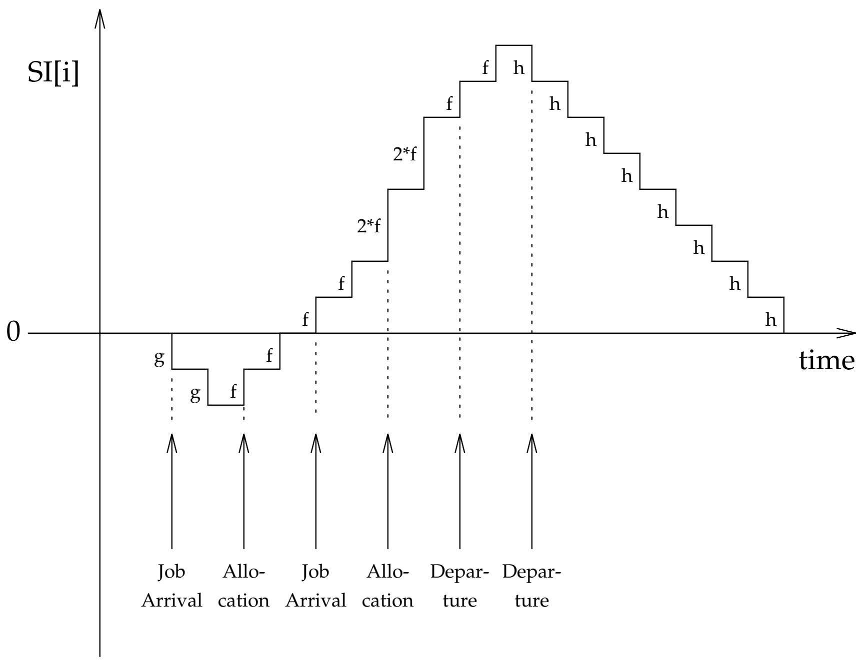

Abbildung 3.3: Modifikation des "schedule index" im zeitlichen Ablauf (nach [MuLi87])

mehr Prozessoren alloziiert sind, umso stärker steigt der "schedule index" innerhalb eines Intervalls an (2*f(SI[i])). Die Funktionen h und l steuern die Normalisierung des "schedule index" über die Zeit, solange keine entfernte Kapazität nachgefragt wird.

- Der Algorithmus:

```
for each ( scheduling Interval: SchedInterval
           station state change: not avail.→avail.
           remote job completed and leaving system )
   do  Allocation of remote capacity:
       S1 ← [bag of WS's that have remote capac. allocated]
       S2 ← [bag of WS's that want remote capac. allocated]
       S3 ← [WS's with free capacity]
       while (S3 <> EMPTY)
         do  if S2 <> EMPTY
               then s= WS in S2 with smallest SI entry
                    allocate capac. of x ∈ S3 to s
                    S2 ← S2 - [s]; S3 ← S3 - [x];
               else break;
             fi
         od
       while (S2 <> EMPTY)
         do  s= WS in S2 with smallest SI entry
             t= WS in S1 with largest SI entry
             if SI[s] < SI[t]
               then preempt capac. of x ∈ S3 from t;
                    allocate capac. of x to s
                    S2 ← S2 - [s]; S1 ← S1 - [t]; S3 ← S3 - [x]
               else break;
             fi
         od
       Modification of schedule index-table:
       for each WS i:
           do  if i wants remote proc. capacity
                 then if i has remote capac.
                        then SI[i]:=SI[i]+NumProcessors*f(SI[i]);
                        else SI[i]:=SI[i]-g(SI[i]);
                      fi
                 elif SI[i] > 0 then SI[i]:=SI[i]-h(SI[i]);
                 elif SI[i] < 0 then SI[i]:=SI[i]+l(SI[i]);
               fi
           od
   od
```

- Einordnung innerhalb des Beschreibungsrahmens:

Algorithmus: Up-Down Algorithmus von Mutka und Livny	
Zielfunktion	Faire Allokation entfernter Kapazitäten von Benutzern mit geringer Ressourcenbelastung gegenüber Benutzern mit hoher Ressourcenbelastung in Umfang und Zeit
Randbedingungen	keine Angaben zu weiteren Randbedingungen
Taktik/Strategie	
information policy	Information über freie Kapazitäten, allociierte Kapazitäten, benötigte Kapazitäten und Rechner-Verfügbarkeit Einholen der Information durch Anpollen der Rechner durch den Koordinator Erneuerung der Information nach jedem Intervall *SchedInterval* **Verfahren: dynamisch (1)**
negotiation policy	Kooperation der Rechner mit einem ausgezeichneten Koordinator **Verfahren: überwacherinitiiert, überwachergesteuert**
decision policy	**Verfahren: dynamisch (2), deterministisch, preemptiv**
transfer	keine explizite Angabe einer transfer-policy
location	die location-policy entspricht dem Abschnitt "Allocation of remote capacity" im Algorithmus
selection	keine explizite Angabe einer selection-policy
status	die status-policy entspricht der zweiten while-Schleife des Abschnittes "Allocation of remote capacity" im Algorithmus
policy issues	Fairness
Struktur-Merkmale	Struktur des Koordinators: Halten einer "schedule index"-Tabelle **Verfahren: zentral**
Ann. über Systemumgeb.	Berücksichtigung der unterschiedlichen Granularität der Prozesse: langlaufende Prozesse werden zugunsten von kürzeren Prozessen angehalten und gecheckpointed

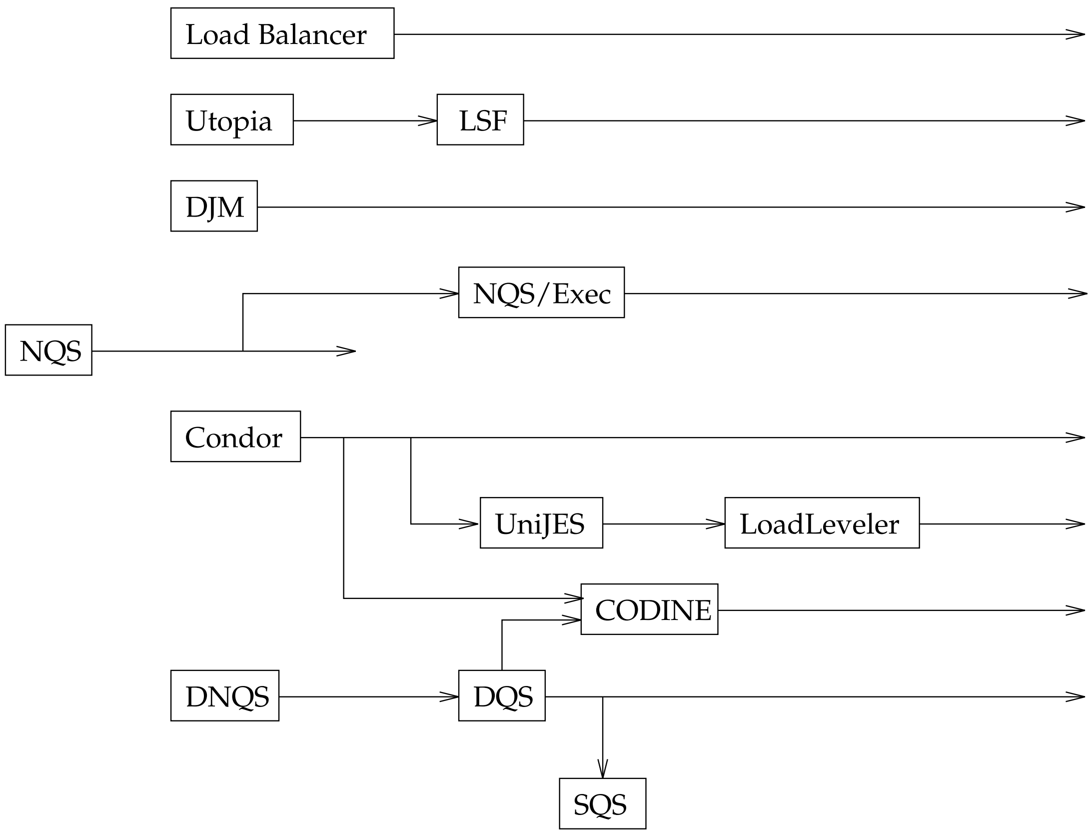

Abbildung 3.4: Batch-Queuing Systeme und ihre Entstehungsgeschichte ([KaNe93])

3.2 Überblick und Analyse verschiedener Produkte und Forschungsprojekte

3.2.1 Überblick über existierende Produkte

Die Verwendung vernetzter Workstations als Workstation-Clusters insbesondere auch im betrieblichen oder wissenschaftlichen Umfeld führt zur Notwendigkeit des Einsatzes entsprechender Cluster-Management- und Batch-Queueing-Software. Verschiedene Systeme sind heute auf dem Markt käuflich oder als Public-Domain-Software erhältlich.

Mit zu den wichtigen Produkten zählen sicherlich DQS (Distributed Queuing Systems), ein an der Florida State Universität entwickeltes System, Condor, ein Produkt, das an der Universität Wisconsin entwickelt worden ist, Load Leveler von IBM, CODINE (Computing in Distributed Computing Environments) der Firma Genias Software sowie NQS/Exec (Network Queueing System) der Firma Sterling Software. Weiterhin wichtige Produkte sind Load Balancer von Freedman Sharp an Associates Inc., Distributed Job Manager (DJM) und SQS (Scalable Queueing System), zwei Public Domain Produkte sowie Load Sharing Facility (LSF) der Firma Platform Computing Corporation in Canada.
Zu Forschungsarbeiten im Bereich Lastbalancierung und Datenmigration zählt unter anderem das an der Universität Stuttgart entwickelte Projekt "HiCon" (Hierarchical Controlled Network Computing).

Zunächst soll eine ausführlichere Analyse der Systeme Condor und Load Leveler vorgenommen werden. Dann soll im Hinblick auf die Bereiche Lastbalancierung und Datenmigration das Forschungsprojekt HiCon näher untersucht werden.

3.2.2 Condor

Condor ist ein in Workstation-Umgebungen operierendes Lastverwaltungs-System, das auf die möglichst optimale Verwendung der Rechner des verteilten Systems hin konzipiert worden ist, wobei die Minimierung der Interferenz zwischen Job-Scheduling und der lokalen Beanspruchung durch Workstation-Benutzer besonders berücksichtigt wird. Aspekte, die bei der Entwicklung dieses Systems besonders berücksichtigt wurden, sind darüberhinaus ([Gosc91] S476):

- Minimierung der Durchlaufzeit jedes Hintergrund-Jobs durch faire Behandlung von kurzen gegenüber langlaufenden Jobs nach dem Up/Down-Algorithmus
- Minimierung der Scheduling-Kosten (Rechenzeit und Speicher-Kapazität)
- Automatischer Neustart von Jobs, die auf ausgefallene Rechner gescheduled wurden
- Transparenz von Ort und Ausführung des Jobs gegenüber dem Benutzer.

Die Struktur des Condor-Lastverwaltungs-Systems
Die Struktur von Condor ist so organisiert, daß die Vorteile der zentralen Verfahren mit denen der dezentralen Verfahren kombiniert werden. Jede Workstation des Pools enthält einen "local scheduler" sowie eine "remote unix facility" neben der "background job queue". Auf einer der Workstations läuft zusätzlich ein Coordinator. Die Funktionsweise der Komponenten kann dabei wie folgt beschrieben werden:

- Die vom Benutzer abgesetzten Prozesse werden in der background job queue gehalten. Jede Workstation hält entsprechende Status-Information über alle lokal gehaltenen Jobs.
- Ein "local scheduler" beobachtet die Station:
  Falls ein Prozess zur Zeit abgearbeitet wird, beobachtet der "local scheduler" alle 30 Sekunden, ob ein Benutzer die Aktivität an der lokalen Station wieder aufnehmen möchte; gegebenenfalls wird der Prozess angehalten ("preempted"). Sind momentan keine Prozesse zu bearbeiten, so kann diese Workstation als möglicher Kandidat für die Bearbeitung von Jobs anderer Rechner dienen. Warten mehr als ein Hintergrund-Job auf Ausführung, so entscheidet der "local scheduler" selbst, welcher Prozess als nächstes ausgeführt werden soll.
- Kompatibilität zwischen lokaler und entfernter Job-Ausführung wird durch die "remote unix facility" erzielt.
  Wird beim Aufruf dieser Komponente ein Prozess auf einen anderen Rechner verschoben, so läuft lokal ein entsprechender Shadow-Prozess. Dieser führt z.B. bei System-Aufrufen die notwendige Arbeit und Kommunikation mit dem entfernten Prozess durch. Bei einer Verschiebungsentscheidung bzgl. eines entfernten Prozesses wird der Job durch die "remote unix facility" gecheckpointed.

- Auf einer Workstation läuft zusätzlich ein Coordinator.
  Der Coordinator prüft in regelmäßigen Abständen (2 Minuten), welche der Workstations freie Ressourcen haben und wie groß die verfügbare Speicher-Kapazität ist bzw. ob auf Rechnern Hintergrundjobs auf Ausführung warten. Entsprechend dieser Information wird anhand des Up-Down-Algorithmus Kapazität von Workstations mit überschüssigen Ressourcen an solche mit wartenden Prozessen zugewiesen. Dadurch, daß der Coordinator keine großen Mengen an Information über jede Workstation hält, kann die Verantwortung des Coordinators einfach gehalten werden und die Fehlertoleranz des Gesamtsystems erhöht werden.

Strategie der Lastbalancierung bei Condor
Allgemein kann die Strategie von Condor wie folgt formuliert werden:
Der zentrale Coordinator beobachtet alle 2 Minuten die Workstations des Pools und assoziiert Rechner mit freien Ressourcen mit solchen, deren Prozess-Warteschlange mehr als einen Prozess enthalten (dies entspricht zentralen Verfahren). Die Stationen entscheiden lokal, welche Jobs mit Hilfe der zugewiesenen Kapazitäten auf einem entfernten Rechner zur Ausführung gebracht werden sollen. Dies entspricht dem Vorgehen in dezentralen Verfahren. Durch die Beobachtung der Aktivität der lokalen Benutzer wird durch entsprechendes Anhalten und Migrieren des Prozesses sichergestellt, daß diese im Zweifelsfall ihre Konsole lokal nutzen können. Da Condor speziell auch eine unterschiedliche Granularität der Prozesse hinsichtlich ihrer Ausführungsdauer berücksichtigt, ist sichergestellt, daß die Ausführung kurzer Jobs nicht durch langlaufende Prozesse zu lange verzögert wird. Der Up-Down-Algorithmus legt mit Hilfe des Scheduling Index (vgl. voriges Kapitel) die Priorität der Prozesse fest. Können nun einem Prozess mit hoher Priorität keine freien Ressourcen zugewiesen werden, wird ein auf einem entfernten Rechner laufender Auftrag mit niedrigerer Priorität zugunsten des höher priorisierten Prozesses angehalten und gecheckpointed, so daß die freiwerdenden Kapazitäten für den Job mit höherer Priorität zur Verfügung stehen.

Wesentliche Dämonen und Prozesse von Condor:
Die Dämonen von Load Leveler haben folgende Aufgaben:

- master Dämon:
  managed alle übrigen Dämonen der entsprechenden Maschine
- schedd Dämon:
  managed das Weiterleiten von Jobs an Condor
- shadow Prozess:
  für lokale Systemaufrufe des verschobenen Jobs zuständig
- startd Dämon:
  empfängt ankommende Jobs von anderen Maschinen und kreiert einen starter Prozess
- starter Prozess:
  durch den startd Dämon kreiert, managed der starter Prozess einen laufenden Job
- kbdd Prozess:
  beobachtet Tastatur- und Maus-Aktivität zur Überwachung der Aktivität lokaler Benutzer

- collector Dämon:
  auf dem "Central Manager" ist der zentrale Empfänger des Maschinen-Status von allen Rechnern des Condor Pools

- negotiatior Dämon:
  des "Central Managers" ist zentraler Empfänger des Job-Status aller Rechner und zentrale Verteilungs-Stelle (und beherbergt daher den eigentlichen Algorithmus zur Lastbalancierung)

Benutzer- und Systemadministrator-Kommandos:

1. Kommandos zum Absetzen von Jobs, Job-Command-Files:
   - condor_submit
   - job-description-file: notwendiges File für condor_submit
2. Kommandos zur Überwachung:
   - condor_q: Information über alle Jobs in der lokalen Queue
   - condor_globalq: Information über alle Jobs auf allen Hosts
   - condor_status: Information über alle Maschinen im Pool
   - condor_summary: Information über vollendete Jobs (Accounting-Information)
3. Komandos und Files zur Konfiguration:
   - condor_config: Anpassung von Condor an einen bestimmten Host
4. Kommandos zur Steuerung und zum Eingriff:
   - condor_prio: Änderung von Job-Prioritäten in Queues
   - condor_rm: Job aus Queue entfernen

3.2.3 Load Leveler

Das Produkt Load Leveler von IBM baut auf der Funktionalität auf Condor auf, mit dem Ziel, die in einem heterogenen Workstation-Cluster vorhandenen Ressourcen durch Verteilung von Aufträgen optimal zu nutzen. Leistungsmerkmale von Load Leveler sind u.a.: Unterstützung heterogener Cluster, Checkpointing und Migration von Jobs, Job-Accounting sowie zu einem bestimmten Grad auch Parallel-Job-Verarbeitung.

Load Leveler Struktur sowie wesentliche Dämonen und Prozesse:
Bei Load Leveler wird in Anlehnung an die Struktur von Condor eine Workstation als "Central Manager" konfiguriert (mit dem negotiatior und collector Dämon), die übrigen Maschinen des Load Leveler Workstation-Pools als mögliche Kandidaten für entfernte Ausführung von Prozessen (mit den Dämonen master, schedd, startd, starter und kbdd). Darüberhinaus können Rechner als "submit-only"-Maschinen konfiguriert werden, was es auch für Rechner außerhalb des Pools möglich macht, Jobs von Load Leveler zu ausführen zu lassen.

Job-Scheduling bei Load Leveler:
Da Load Leveler auf Condor aufbaut, ist zu vermuten, daß auch die Lastbalancierungs-Strategie der von Condor zumindest ähnlich ist. Alle Jobs werden zu dem die Anforderungen des Job Command-Files am besten erfüllenden Rechner des Load Leveler Pools geschickt - unter Berücksichtigung der durch die Messung der Auslastung der Ressourcen und der Verfügbarkeit der Maschinen (User-Aktivität) ermittelten Größen.
Automatisch können Entscheidungen zu Umverteilung, Starten und Stoppen sowie zu Ausführung an einem bestimmten Zeitpunkt getroffen werden.
Die Umsetzung der Job-Verteilungs-Entscheidung wird durch Checkpointing-Funktionalität und Unterstützung von Prozeß-Migration erreicht.

Lasterfassung bei Load Leveler:
Der zentrale Prozess, der im Load Leveler System den Status des Rechners beobachtet, heißt collector Dämon. Für die Erkennung der Beanspruchung des Rechners durch den lokalen Besitzer wird die Tastatur- und Maus-Aktivität durch den keyboard Dämon überwacht. Information wird insbesondere eingeholt über User-Aktivität, Systemlast, Plattenplatz und Swap-Größe ([KaNe93b] S34). Für die Messung der durch Prozesse verursachten Last verwendet Load Leveler das "Berkeley One Minute Load Average" ([IBM93a]). Information über den Status gescheduleter Jobs liefert der negotiator Dämon.

Ressourcen-Konfiguration bei Load Leveler:
Load Leveler unterstützt verschiedene Hardware-Plattformen, wie z.B. Sun-Workstations, SGI Workstations, IBM RISC System/6000 oder IBM 9076 Scalable POWERparallel Systems (SP1).
Genaue Spezifikationen der Maschinenkonfigurationen werden über ein globales und jeweils lokales Konfigurations-File an das System mitgeteilt.
Im Machine-Configuration-File angegebene Größen sind:

1. für das Scheduling notwendige Angaben:

- Maschinen-Name
- Architektur
- Betriebssystem
- auf dieser Maschine ablauffähige Job-Klassen
- maximale Anzahl gleichzeitig lauffähiger Jobs
- spezifische "Features" dieser Maschine
- System-Priorität
- Größe des für lokale Jobs reservierten Swap-Bereichs

2. für die Verwaltung notwendige Angaben:
   - Definition der Directory-Pfade für Administration-File, lokales Configuration-File, File für Checkpoints der Jobs, Log-Files, File für History von lokalen Load Leveler-Jobs, Load Leveler Software, u.a.
   - Spezifikationen für alle Load Leveler Dämonen:
     - Directory-Angaben
     - Log-File-Angaben
     - Debug-Spezifikationen
     - weitere Dämonen-spezifische Angaben

Im Administration-File angegebene Größen sind:

1. "User-Stanzas" (Charakteristik der Benutzer):
   Angaben zu Benutzer-Priorität, Default-Klasse, Maximale Anzahl Jobs, die der Benutzer ablaufen lassen kann
2. "Class-Stanzas" (Charakteristik einer Klasse):
   Angaben zu Klassen-System-Priorität, Liste der Benutzer, die Jobs dieser Klasse absetzen dürfen sowie Liste der Benutzer, die dies nicht dürfen; Angaben zu Hardlimit von CPU, Data, Core-Größe, File-Größe, Stack-Größe und RSS
3. "Machine-Stanzas" (Charakteristik der Maschine):
   Festlegung, ob Rechner "Central Manager" oder eine "Submit-only"-Maschine ist

Konfiguration und Rekonfiguration von Workstations ist auch zur Laufzeit möglich.
Die Verfügbarkeit der Maschinen für Lastverteilung wird implizit in den Konfiguration-Files entschieden. Ein Rechner kann immer, zu bestimmten Zeiten oder nie verfügbar sein bzw. genau dann, wenn über den kbdd-Dämon keine Benutzer-Aktivität registriert wird.

Angaben im Job-Command-File:
Spezifizierung der Anforderungen und Charakteristiken eines an die Lastverwaltung weitergegebenen Jobs wird über das Job-Command-File geregelt. Wesentliche Parameter sind:

- Name und Argumente des Jobs, Filename für Standard-Input, Output, Error, Environment-Variablen, Directory und Shell, von der aus der Job gestartet werden soll,

- Datum und Uhrzeit des Start-Zeitpunktes
- Job-Priorität zwischen 0 und 100
- Erwartetes Anwachsen des Jobs während der Laufzeit in kByte
- Klassentyp: Spezifikation der Gruppe von Maschinen, die diese Klasse von Jobs unterstützen
- Halte-Status: User-, System- oder User and System-Hold
- Parallele Jobs: Anzahl benötigter Knoten für die Bearbeitung
- Account Number: identifiziert den Job für Accounting-Daten
- Checkpointing: entscheidet, ob der Job gecheckpointed werden kann
- Restarting: Möglichkeit des Restartens eines Jobs, der vor seiner Beendigung abbricht
- Anforderungen und Preferenzen des Jobs:
  - Architektur
  - Betriebssystem
  - Plattenplatz
  - Speicherplatz
  - Spezifizierung von bestimmten Maschinen nach Namen
  - Spezifizierung von Maschinen nach bestimmten Merkmalen (Features)
- Angabe von Limits für den Job (Hardlimit und Softlimit)
  - CPU-Limit (max. mögl. CPU-Nutzung)
  - Data-Limit (max. mögl. Größe an Daten-Segmenten, die ein verschickter Job verwenden kann)
  - Core-Limit (max. Größe des Core-Files)
  - RSS-Limit (max. mögl. Größe an physischem Speicherplatz, den der Prozeß alloziieren kann)
  - File-Limit (max. Größe eines kreierten Files)
  - Stack-Limit (max. Größe des Stacks)

Job-Accounting:
Load Leveler erstellt für Jobs, deren Abarbeitung beendet ist in einem File Abrechnungs-Information zur Verfügung, unter anderem über Job-Parameter oder Umfang an verbrauchten Ressourcen. Diese Funktionalität dient u.a. dazu, über die verschiedenen Job-Typen auf dem Laufenden zu bleiben oder für verschiedene Typen von Jobs die benötigten Ressourcen festzuhalten. ([IBM93b])

Benutzer- und Systemadministrator-Kommandos:

Load Leveler unterteilt Kommandos in User-Kommandos und in solche, die nur von Systemadminsitratoren abgesetzt werden können.

1. Kommandos zur Überwachung:
   - llq: Status-Abfrage abgesetzter Jobs
     - Standard-Ausgabe:
       Die Standard-Ausgabe liefert Angaben zu Job-Identifier, Job-Eigentümer, Zeitpunkt der Absendung, momentaner Job-Status, Job-Priorität, Job-Größe in Megabytes, Job-Klasse, ausführender Rechner
     - Erweiterte Ausgabe:
       Die Erweiterte Ausgabe liefert zusätzlich Angaben zu Zeitpunkt der Ankunft bei Load Leveler, Zeitpunkt der Job-Beendigung, vom Eigentümer festgelegte Job-Priorität, Benutzer-System-Priorität des Jobs, Klassen-System-Priorität des Jobs, System-Priorität des Jobs, Job-Priorität zum Zeitpunkt der Absendung, Mitteilungs-Status des Jobs, Virtual-Image-Größe, Checkpoint-Status, Name des Programms, Programm-Aufruf-Argumente, gesetzte Environment-Variablen, Files für Eingabe, Ausgabe und Errors, Verzeichnis, in dem Job ausgeführt wird, Job-Anforderungen und Job-Präferenzen des Job-Command-Files, lokale und entfernte Benutzer und System CPU-Zeit auf der Client- und Server-Maschine nach Job-Beendigung, ausführender Rechner, absendender Rechner, über Job-Status zu benachrichtigender Benutzer, Job-Shell, Job-Klasse, Limits, wie im Job-Command-File angegeben
   - llstatus: Status-Abfrage der Rechner
     - Standard-Ausgabe:
       Die Standard-Ausgabe der Maschinen-Überwachung liefert Angaben zu Rechner-Name, Anz. laufender Load Leveler Jobs, totale Anz. an Load Leveler Jobs auf diesem Rechner, Maschinen-Status (Busy, Down, Drain, Flush, Idle, None, Running, Suspend), Berkeley "Ein-Minuten-Last-Durchschnitt" dieser Maschine, Hardware-Architektur und Betriebssystem der Maschine, Sekunden seit letzter Maus- od. Tastatur-Aktivität
     - Erweiterte Ausgabe:
       Zusätzliche Angaben können abgerufen werden zu qualifiziertem Rechner-Namen, mögliche Zustände von Jobs auf diesem Rechner (start, suspend, continue, vacate, kill), System-Priorität von Jobs auf dieser Maschine, Anzahl Benutzer mit Jobs auf diesem Rechner, Anzahl nicht laufender Jobs auf dieser Maschine, Aktivität von schedd und startd Dämons auf diesem Host, Zeitpunkt des Setzens des Maschinen-Status, verfügbarer Swap-Größe und Disk-Größe sowie physischer Speicherplatz, Anzahl CPU's auf diesem Rechner, Menge möglicher Klassen (z.Zt. nicht verwendete und alle mgl. Klassen), Menge von verfügbaren "Features", TCP/IP-Teilnetz dieser Maschine, max. Anzahl Jobs, die einzelner Benutzer gleichzeitig laufen lassen kann bzw. die überhaupt zur gleichen Zeit laufen können, Zeitpunkt der letzten Rekonfiguration, Zeitpunkt des letzten Status-Updates an den "Central Manager"

- Graphisches Benutzer-Interface (GUI):
  Die graphische Benutzer-Oberfläche liefert in drei Fenstern Angaben zum Status abgesetzter Jobs, Status von Rechnern und Load Leveler-Nachrichten.

2. Kommandos zum Eingriff und zur Steuerung;

   - llinit: Neuen Rechner des Pools initialisieren
   - llctl (recycle): alle Load Leveler Dämonen stoppen und neustarten
   - llctl (reconfig): alle Load Leveler Dämonen zum erneuten Einlesen der Configuration-Files veranlassen
   - llctl (stop): Load Leveler Dämonen auf speziellem Host stoppen
   - llctl (start): Load Leveler auf spezifischem Host starten
   - llctl (drain): Keinen weiteren Job auf angeg. Host akzeptieren
   - llctl (suspend/resume): alle Jobs auf angeg. Host suspendieren bzw. fortsetzen
   - llctl (flush): laufende Jobs terminieren und zu System-Queue zur erneuten Verteilung zurücksenden
   - llfavorjob, llfavoruser: System-Prioritäten verändern
   - llprio: Job-Prioritäten verändern
   - llreschedule: schedd-Dämon veranlassen, Job-Queues der angegebenen Maschine zu scannen und u.U. Status auf den neuesten Stand zu bringen
   - llcancel: Jobs der Load Leveler Queue löschen
   - llhold: Jobs anhalten bzw. weiterlaufen lassen

3. weitere Kommandos:

   - llpreen: nicht erwartete Files der Load Leveler-Directories entdecken, anzeigen bzw. entfernen

Mechanismen zu Fehlertoleranz und Fragen der Sicherheit:
Der Load Leveler Master Scheduler erkennt Maschinen, deren Dämonen nicht mehr aktiv sind, sendet an solche Maschinen keine Jobs mehr, und versucht, den abgestürzten Dämonen zu reaktivieren. Bei Absturz des Master Scheduler laufen die aktiven Jobs zu Ende, es werden keine weiteren Jobs zur Ausführung gebracht. In jedem Fall ist aufgrund der Checkpointing-Unterstützung der Verlust an Daten gering.
Load Leveler hat außer der Abstützung auf das Sicherheitssystem von UNIX keine zusätzlichen Sicherheitsmechanismen implementiert.

Schnittstellen zum System:
Load Leveler unterstützt verschiedene Hardware-Plattformen, wie z.B. Sun-Workstations, SGI Workstations, IBM RISC System/6000 oder IBM 9076 Scalable POWERparallel Systems (SP1).
Für die Funktionsfähigkeit des Systems erforderlich ist NFS oder AFS (Network File System,

Andrew File System). Message-Passing zwischen Jobs wird nicht unterstützt. Bezüglich der Kooperation mit anderen Batch-Queueing-Systemen besteht bei Load Leveler die Schnittstelle zu NQS (Network Queueing System): Load Leveler akzeptiert größtenteils NQS-Skripts und kann Jobs an Maschinen außerhalb des eigenen Workstation-Pools routen, wenn diese NQS unterstützen)

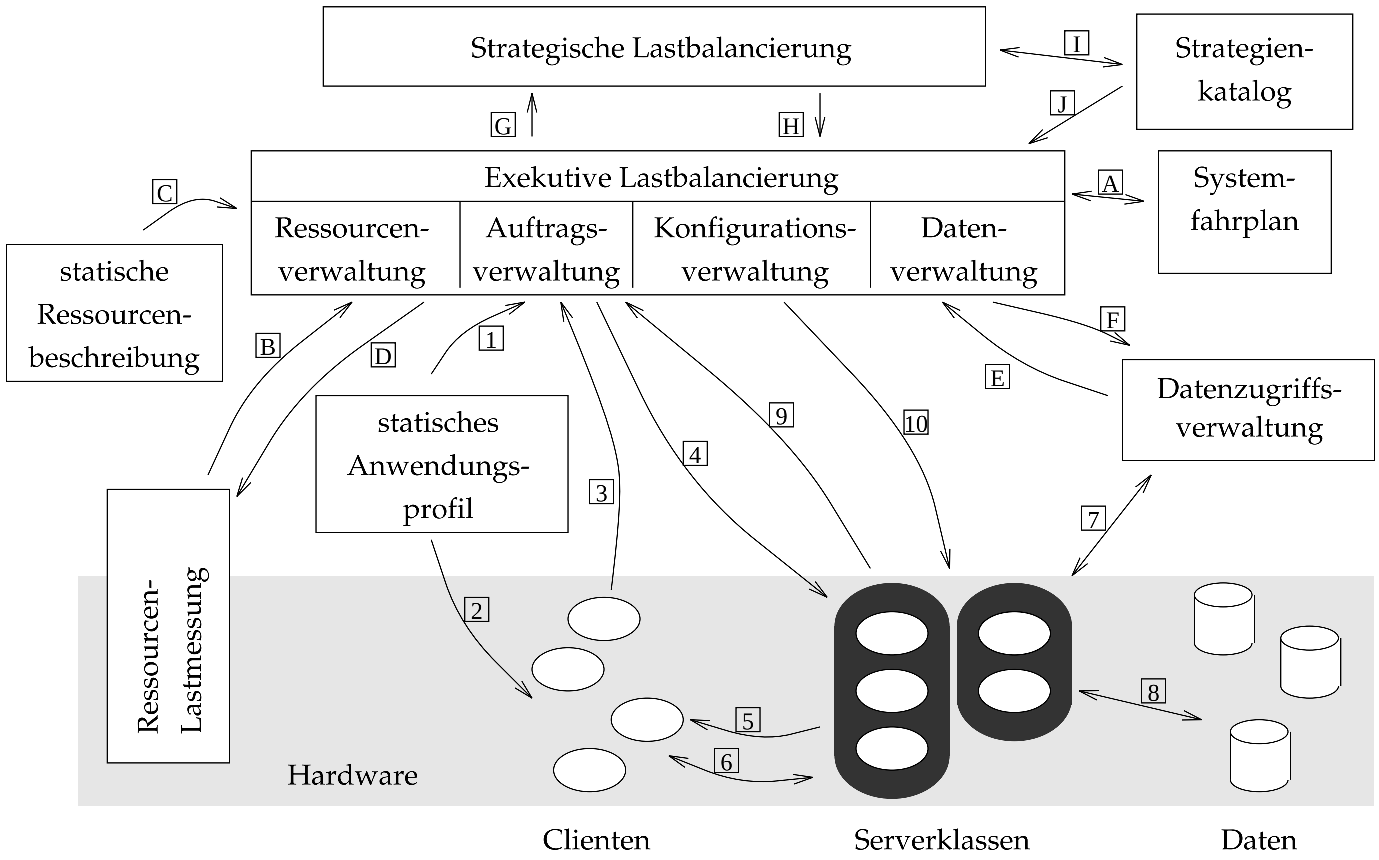

Abbildung 3.5: Logische Struktur des HiCon-Ansatzes (nach [Beck93])

3.2.4 HiCon

Im Gegensatz zu den vorangegangenen Beispielen konzentriert sich das an der Universität Stuttgart entwickelte Lastbalancierungs-System HiCon ([Beck93], [Beck94]) nicht nur auf die Verteilung unabhängiger Aufträge, sondern speziell auf die Balancierung von datenintensiven, verteilten Anwendungen in heterogenen Client-Server-Architekturen.
Ziel der Lastbalancierung nach HiCon ist die Minimierung der Gesamtausführungszeit der Anwendung; nach [Beck93] wird diese Art der Balancierung auch soziale Lastbalancierung bezeichnet.
Das Zusammenspiel der Komponenten gibt die Abbildung 3.5 (nach [Beck93]) wieder.

Die zentrale Komponente von HiCon ist die Exekutive Lastbalancierung. Aufträge oder Auftragsgruppen werden von der Auftragsverwaltung von Clienten entgegengenommen $\boxed{3}$ und an eine Instanz einer Serverklasse zugewiesen $\boxed{4}$. Die Konfiguration der Serverklassen erfolgt über die Konfigurationsverwaltung $\boxed{10}$. Information über den aktuellen Zustand der Ressourcen wird über die Ressourcen-Lastmessung $\boxed{B}$ der Ressourcenverwaltung mitgeteilt; die statische Ressourcenbeschreibung $\boxed{C}$ liefert Information über die Charakteristika der Ressourcen. Die Datenzugriffsverwaltung informiert die Datenverwaltung über Ort und Zustand der Datensätze $\boxed{E}$. Die Synchronisation des Zugriffs auf Daten durch Serverinstanzen $\boxed{8}$ wird über die Datenzugriffsverwaltung geregelt $\boxed{7}$, die wiederum durch die Datenverwal-

tung Hinweise über die günstige Verteilung der Datensätze erhält $\boxed{F}$. Durch den System-Fahrplan $\boxed{A}$ kann die Exekutive Lastbalancierung die Abwicklung der Lastbalancierungs-Aufgabe im voraus planen und anhand aktueller Parameter korrigieren. Die in der Exekutiven Lastbalancierung angewandte Lastbalancierungs-Strategie kann über die Strategische Lastbalancierungskomponente ausgewechselt werden $\boxed{G}$, $\boxed{H}$ und $\boxed{J}$, wozu die strategische Lastbalancierung einen Katalog verschiedener Lastbalancierungs-Strategieen hält $\boxed{I}$. Je nach ausgewählter Strategie werden Exekutive Lastbalancierung $\boxed{H}$ und Ressorucen-Lastmessung $\boxed{D}$ geeignet konfiguriert.
Die Funktionalität der einzelnen Komponenten soll im folgenden näher untersucht werden.

Verwaltung von Aufträgen und Auftragsgruppen durch die Auftragsverwaltung der Exekutiven Lastbalancierung:

Die Aufgaben der Komponente Auftragsverwaltung der Exekutiven Lastbalancierung umfassen die transparente Zuordnung von Einzelaufträgen und Auftragsgruppen an Instanzen. Allgemein erfolgt die Balancierung aufgrund der Faktoren "Rechenaufwand", "Datenlokalität und "Kommunikationsbedarf". Im Einzelnen umfaßt das zur Entscheidungsfindung benötigte Wissen:

- Hardware, Zustand der Serverinstanzen - vgl. Ressourcenverwaltung, Beschreibung
- Verhalten der Hardware, momentane und zukünftige Ressourcenbelastung (vgl. [Beck93] S17) - vgl. Ressourcenlast-Messung
- Verteilung von Daten, Replikationen, Aufwand für Kontextverwaltung und Antwortzeitverhalten der Aufträge - vgl. Systembeobachtung
- Statische und dynamische Charakterisierung der Anwendung, Reihenfolgebezieungen etc. - vgl. Anwendungsbeschreibung
- Angaben aus dem System-Fahrplan - vgl. Protokollierungs-Funktionalität

Die Zuweisung der Aufträge geht dabei wie folgt vor sich:
Aufträge oder Auftragsgruppen werden von der Auftragsverwaltung von Clienten entgegengenommen $\boxed{3}$ und an eine Instanz einer Serverklasse zugewiesen $\boxed{4}$. Die Zuweisung erflogt dabei so spät wie möglich, um für die Bearbeitung des Auftrages die bestgeeignetste Serverinstanz auswählen zu können. Wird also eine Serverklasse über die Auftragsverwaltung durch einen Client aufgerufen, so wird innerhalb der Exekutiven Lastbalancierung zunächst der Systemfahrplan angepaßt $\boxed{A}$, danach mittels der Strategischen Lastbalancierung eine geeignete Lastbalancierungs-Methode aktiviert $\boxed{10}$ und mittels einer Bewertungsfunktion festgestellt, welche Instanz die für den Auftrag bestgeeignetste ist. Der Auftrag verbleibt dabei so lange in der Klassenwarteschlange der Auftragsverwaltung, bis auch wirklich sichergestellt ist, welche Serverinstanz tatsächlich am besten geeignte ist; dann wird der Auftrag zugewiesen und der Instanzen-Warteschlange hinzugefügt. Die Serverklasse unterstützt die Lastbalancierung während der Bearbeitung $\boxed{9}$ bzw. nach Rückgabe der Resultate durch Weitergabe von Information über ihren Zustand, was insbesondere bei kontextsensitiven Aufrufen wesentlich ist, d.h. wenn Client und Serverinstanz direkt miteinander kommunizieren müssen $\boxed{6}$. Nach

Beendigung des Auftrages durch die Serverinstanz sendet diese eventuell Resultate an den Client zurück $\boxed{5}$.

Verwaltung von Datenbeständen durch Datenverwaltung und Datenzugriffsverwaltung

Die Verwaltung von Datensätzen gliedert sich bei HiCon in die Teile Datenverwaltung (transparente Verteilung und Replikation globaler Daten) und Datenzugriffsverwaltung (Synchronisation des Zugriffs auf globale Daten; [Beck93] S4). Datenverwaltung und Datenzugriffsverwaltung sorgen dafür, daß die Serverinstanzen einer Klasse einen virtuellen, gemeinsamen Datenspeicher sehen ([Beck93] S17). Bei HiCon sind die Komponenten Datenverwaltung und Lastverteilung unabhängig voneinander angelegt:
Die Verteilung der Daten wird bei der Zuweisungsentscheidung von Aufträgen mitberücksichtigt (vgl. Auftragsverwaltung), steht aber nicht direkt unter dem unmittelbaren Zugriff der Lastbalancierung. Die Verschiebung von Daten wird i.d. Regel implizit durch Auftragszuweisung durchgeführt ([Beck93] S10), d.h. Daten werden erst dann verschoben, wenn Aufträge diese bei der Datenzugriffsverwaltung anfordern $\boxed{7}$.

- Beschreibung der Datenverwaltung:
  Aufgaben der Datenverwaltung betreffen Verteilung von Datenbeständen, Placieren von Original-Datensätzen, Verteilen von Kopien, Ungültigmachen von Kopien und Konsistenthaltung von Datenpartitionen - allgemein die dynamische Migration und Replikation von Daten.
  Entscheidungsgrundlagen liefern u.a. Profile von Auftragsgruppen (vgl. Anwendungs- und Auftragsbeschreibung), Informationen aus der Systembeobachtung oder längerfristige Beobachtung aggregierter Daten. Informationen dazu werden der Datenzugriffsverwaltung über die Datenverwaltung der Exekutiven Lastbalancierung mitgeteilt $\boxed{F}$.
  Es wird beispielsweise eine Kopie an eine Serverinstanz gesendet, wenn diese die Datenpartition lesen will, ein Original gesendet, wenn die Wahrscheinlichkeit für einen Schreibzugriff hoch ist und ein Datenobjekt sofort entzogen, falls die Instanz den Satz aller Wahrscheinlichkeit nach nicht mehr verwendet.

- Beschreibung der Datenzugriffsverwaltung:
  Die Datenzugriffsverwaltung sorgt für die Synchronisation des Zugriffs auf gemeinsame Daten. Nach Anmeldung des Zugriffs auf den gewünschten Datensatz werden die Daten nach geeigneter Sperrung der entsprechenden Instanz zugänglich gemacht und zum frühestmöglichen Zeitpunkt wieder freigegeben $\boxed{7}$.

Beobachtung des Workstation-Clusters durch Datenzugriffsverwaltung, Ressourcen-Lastmessung und statische Ressourcenbeschreibung

- Ressourcen-Lastmessung
  Beobachtete Größen der Ressourcen-Lastmessungen umfassen allgemein Prozessoren, Platten, Verbindungsnetze, Busse, Warteschlangen und den Zustand der Serverinstanzen. Dabei sind vor allem wichtig: Run Queue Length, also die Serverwarteschlange;

weniger relevant sind meist: CPU-Busy-Time, Auslastung von Plattencontroller und Belastung des Netzwerks für die dynamische Lastbalancierung $\boxed{\text{B}}$.

- Systembeobachtung
  Die Datenzugriffsverwaltung liefert der Exekutiven Lastbalancierung Informationen über Ort und Zustand der Datensätze $\boxed{\text{E}}$. Weitere durch das Lastbalancierungs-System beobachtete Größen sind Antwortzeitverhalten von Prozessen und Aufträgen, Verteilung von Aufträgen und erreichte Laufzeiten.

- Statische Ressourcenbeschreibung
  Unveränderliche Charakteristiken der Ressourcen werden über die statische Ressourcenbeschreibung an die Exekutive Lastbalancierung mitgeteilt $\boxed{\text{C}}$.

- Meß-Intervalle
  Prozesse zur Messung der relevanten Größen senden periodisch nur dann den Lastvektor an das System, wenn eine signifikante Laständerung beobachtet werden kann. (Minimierung des Nachrichtenaufkommens im Netz)

- Variation der beobachteten Größen, je nach ausgewählter Strategie
  Das Lastbalancierungs-System kann zur Laufzeit Elemente des Lastvektors sowie Meßperioden variieren; dabei werden je nach ausgewählter Strategie die entsprechenden Größen und Intervalle eingestellt $\boxed{\text{D}}$.

Ressourcenkonfiguration durch die Konfigurationsverwaltung der exekutiven Lastbalancierung

Die Ressourcenverwaltung der Exekutiven Lastbalancierung verwaltet geeignete Serverkonfigurationen und kreiert bzw. terminiert Serverinstanzen. Dazu notwendige Entscheidungen werden u.a. aufgrund von Profilen von Auftragsgruppen und längerfristigen Beobachtungen aggregierter Daten getroffen.
So werden z.B. bei ständig sehr langen Klassen-Auftragswarteschlangen neue Instanzen kreiert oder bei großen mittleren Wartezeiten von Instanzen auf neue Aufträge existierende Instanzen gelöscht.

Das statische und dynamische Anwendungsprofil von verteilten Anwendungen

Bei HiCon gliedert sich eine Anwendung in Serverklassen und Serverinstanzen. Der Ablauf einer Anwendung besteht aus der Funktionsausführung durch Serverinstanzen. Zur Entscheidung wird auch Information aus einem statischen und dynamischen Anwendungsprofil gewonnen, das redundant ist, aber eine flexiblere Anpassung an verschiedene Anwendungen erlaubt, als dies ein festes Standardprofil für alle Aufträge gestatten würde.

1. Statische Beschreibung:
   Die Statische Beschreibung gibt Abschätzungen, die für den gesamten Verlauf der Anwendung gültig sind. Aufträge lassen sich Auftragstypen einteilen; die Zuordnung zu bestimmten Auftragstypen erfolgt aufgrund von Parametern und Zuständen der Serverinstanzen.

- Angaben pro Auftragstyp:
  - mittlerer Aufwand an Rechenzeit
  - externe Datenzugriffe pro Ausführung
  - Anzahl von Unteraufrufen an verschiedene Serverklassen
  - Schreib- und Leseverhältnis bei Zugriff auf globale Daten
- Globale Angaben:
  - Häufigkeit von Serveraufrufen bestimmter Auftragstypen
  - Parallelität von Serveraufrufen bestimmter Auftragstypen
  - aus Parallelität entstehende Kommunikations-Last

2. Dynamische Beschreibung:
   - Einzelaufträge:
     - Beschreibung erst möglich zum Aufrufzeitpunkt
     - Angabe von Maßen für benötigte Prozessor-Rechenzeit und Anzahl notwendiger Plattenzugriffe
     - Auflistung der voraussichtlich zur Auftragsbearbeitung benötigten globalen Daten
     - Abschätzung der während der Auftragsbearbeitung entstehenden Unteraufrufe
   - Auftragsgruppen:
     - Reihenfolgebeziehungen und Schleifen, allg. Beziehung zwischen Aufträgen
     - Abschätzen der
       * Intensität der Kooperation der Teilaufträge
         (betrifft die Instanzenauswahl und deren Kommunikations- Verbindung)
       * entstehende Parallelität
         (betrifft die Möglichkeit der Konfiguration von Servern)
       * Reihenfolge-Abhängigkeiten
         (betrifft Wartezeiten auf synchrone Aufrufe(
       * Anzahl und Verhältnis von Lese- und Änderungszugriffen auf Kontext-Partitionen
         (betrifft den Kompromiß von Parallelität und Kontextverwaltungsaufwand)

Einsatz unterschiedlicher Strategien durch die Komponenten Strategien-Katalog und strategische Lastbalancierung

HiCon verwaltet im Strategien-Katalog einen Katalog möglicher Lastbalancierungs-Strategien, die bei Bedarf durch die strategische Lastbalancierung zum Einsatz gebracht werden. Entscheidungsgrundlagen für die Auswahl einer Strategie liefert die Beobachtung des Systems (statisches Anwendungsprofil, Ressourcenlast-Messung und erreichte Laufzeiten) $\boxed{\text{B}}$, $\boxed{\text{G}}$ und Kriterien, die von der Strategischen Lastbalancierung verwaltet werden und angeben, unter welchen Bedingungen eine Strategie gut verwendet werden kann. Die

Entscheidung für einen Strategien-Wechsel wird durch die strategische Lastbalancierung getroffen, die eine geeignete Strategie aus dem Strategien-Katalog auswählt $\boxed{\text{I}}$, die Strategie in der exekutiven Lastbalancierung einsetzt $\boxed{\text{H}}$ und die benötigten Lastvektor-Größen und Meßintervalle für die Ressourcenlast-Messung einstellt $\boxed{\text{D}}$. Die Kriterien, die den Einsatz unterschiedlicher Lastbalancierungs-Strategien entscheiden, werden anhand verschiedener Größen durch die Strategische Lastbalancierungs-Komponente regelmäßig verbessert.

System-Fahrplan

Der Systemfahrplan verwaltet die erwartete Auftragsverteilung und Systembelastung (Zukunft) und protokolliert bzw. führt eine Anpassung des Fahrplanes anhand gemessener Größen durch (Vergangenheit). dabei wird der Systemfahrplan primär über den Achsen Server, Ressourcen und Aufträgen aufgespannt. ([Beck93] S18)
Beispiele für Aufzeichnungen im System-Fahrplan sind weiterhin:

- Server- und Datenkonfiguration
- momentane Auslastung der Ressourcen
- Charakteristika der Ressourcen
- erwartete Auslastung und Laufzeit eines Auftrags an verschiedenen Stellen im System

Beispiele für die Anpassung des Systemfahrplanes:

- Serverinstanzen können das Ende einer Bearbeitung mitteilen, was im Systemfahrplan protokolliert wird $\boxed{9}$, $\boxed{\text{A}}$.
- Anpassung des Systemfahrplans bei Aufruf einer Serverklasse durch einen Client

Benutzer- und Systemadministrator-Schnittstelle:

In [Beck93] (S25ff) werden für die Simulation des HiCon-Modells folgende Angaben zur graphischen Darstellung relevanter Information gemacht:

- Graphische Darstellung der Anwendungsbearbeitung:
  gemittelte prozentuale Nutzungszeit (busy time) für jeden Prozessor, jede Platte und jeden Kanal,
- Phasen der Bearbeitung aller Prozesse über die Zeit:
  - Rechenzeit
  - Wartezeit aufgrund von Synchronisation
  - Wartezeit aufgrund von Plattenzugriffsverzögerungen
  - Wartezeit aufgrund von Unteraufrufen
- Zustand der Serverinstanzen (gegliedert nach Serverklassen):

– bereits erledigter Anteil des momentan bearbeiteten Auftrags

– derzeit lokal vorhandene Datenpartitionen bzw. Kopien

– aktuelle Länge der Klassenauftragswarteschlangen

Zur Unterstützung der Fehlersuche werden folgende Möglichkeiten angeboten:

- Timeouts:
  Jeder Bibliotheks-Aufruf ist mit Timeouts versehbar, so daß Protokollfehler und Ausfälle der Anwendung erkennbar sind

- Protokolldateien:
  Es sind für alle Serverinstanzen Protokolldateien vorhanden, in die alle Anwendungsprozesse schreiben, so daß Fehler durch Vergleich der Protokolldateien feststellbar sind.

Kapitel 4

Einsatz-Szenarien von Lastbalancierungs-Systemen

Wie durch die Analyse verschiedener Algorithmen, Produkte und Forschungsprojete schon erkennbar wurde, sind Lastverwaltungs-Systeme für durchaus sehr unterschiedliche Szenarien einsetzbar: Während in Lastbalancierungs-Algorithmen keine oder nur wenige Angaben über Netzstrukturen oder Rechner-Voraussetzungen gemacht wurden, wurde z.B. der Unterschied der Konzeption z.B. von Load Leveler einerseits und HiCon andererseits durch die Entwicklung bezüglich des beabsichtigten Einsatzes doch recht deutlich.
Einer der wesentlichen Aspekte von Einsatz-Szenarien für Lastverwaltungs-Systeme ist sicherlich die Art von Objekten (Prozesse und Daten), die verteilt werden sollen. Sollen in erster Linie einzelne, unabhängige Aufträge innerhalb der vorhandenen Kapazitäten verteilt werden, so stehen andere Aspekte der Lastverteilung im Vordergrund als bei der Verwaltung z.B. von datenintensiven verteilten Anwendungen. Dies kommt unter anderem in der Zielfunktion zum Ausdruck, der der Lastbalancierungs- und evtl. auch Datenmigrations-Mechanismus zugrunde liegt. Entsprechend der Konzeption der Lastverwaltung bezüglich diese Aspektes muß auch das Management geeignet angepaßt werden. Insbesondere sind z.B. Management-Strategien bzgl. des Performance-Managmenets je nach der Zielfunktion des Lastverwaltungs-Algorithmus unterschiedlich zu formulieren. Dies läßt sich für unterschiedliche Einsatz-Szenarien generell feststellen.
Neben dem Aspekt der zu schedulenden Objekte ist sicherlich ein weiterer Aspekt das System, auf dem Prozesse und Daten verteilt werden sollen. Lastverwaltungs-Systeme, die ausschließlich in homogenen Umgebungen arbeiten, müssen weniger Aspekte bei der Verteilung berücksichtigen als solche, die heterogene Strukturen berücksichtigen müssen. Der Wahl effektiver Algorithmen sind in heterogenen Umgebungen engere Grenzen gesetzt als in homogenen Umgebungen.
Innerhalb der Betrachtung des Aspektes des System, auf dem die Lastverwaltung arbeitet, ist eine weitere Differenzierung von Interesse. Lastbalancierung von Aufträgen wird zum einen in verteilten Systemen betrieben, in denen Worstations durch Kommunikations-Verbindungen miteinander kommunizieren. Andererseits ist Lastbalancierung auch für Parallel-Rechner wesentlich, in denen keine "langsamen" Verbindungen existieren, sondern die Anordnung der Rechnerknoten und ihre Entfernung voneinander so gestaltet ist, daß die Kommunikation geschwindigkeits-optimiert ist. Und wesentlich ist ebenfalls, ob ein gemeinsamer virtueller Speicher vorhanden ist, oder ob die Lastverwaltung von jeweils lokalen Speicherbereichen ausgehen muß.

Zu unterscheiden ist weiterhin, ob Lastverwaltung als eigenständige Software installiert wird, wie dies bei gängigen Batch-Queuing-Produkten wie Load Leveler oder DQS der Fall ist, oder ob ein Lastverwaltungs-Mechanismus fest in das verteilte Betriebssystem integriert ist, wie dies z.B. für den Mach-Mikrokernel existiert.
Zwei weitere Aspekte von Szenarien des Einsatzes von Lastbalancierung und Datenmigration sind sicherlich das Vorhandensein von interaktiven Benutzern und damit verbunden die Konzeption der Ressourcen als eine gemeinsame, virtuelle Ressource. Soll Lastverwaltung in einer Umgebung durchgeführt werden, in der interaktive Benutzer lokal ihre Workstations verwalten, so kann es nötig sein, Ressourcen in private und der Lastverwaltung zugängliche Bereiche aufzuteilen. Weiterhin wird das Konzept der Verfügbarkeit von Rechnern für die Lastbalancierung und Datenmigration von Bedeutung. Lastverwaltung kann bei Vorhandensein interaktiver Benutzer aber auch so ausgelegt werden, daß diese keine lokalen, privaten Ressourcen auf ihren Rechnern zur Verfügung haben, sondern innerhalb eines vollständig transparenten, gemeinsamen Ressourcen-Raumes arbeiten. Durch mehrere Lastverwaltungs-Systeme schließlich können auch mehrere Domänen in diesem Sinne gebildet werden. Dies gilt auch dann, wenn Workstation-Cluster ohne interaktiven Benutzer-Betrieb arbeiten. Hier dienen Cluster dann vor allem als alternative Parallel-Rechner insbesondere der Bearbeitung von größeren Problemstellungen.
Als letzter Aspekt sei hier noch die Entfernung von Rechnerknoten genannt. Der Einsatz von Lastverwaltungs-Systemen ist grundsätzlich auf der Verwaltung von Prozessen und Daten in LAN's ausgelegt. Es sei hier allerdings die Möglichkeit angesprochen, Lastverwaltung auch über größere Entfernungen hinweg zu betreiben. Wenn für Benutzer nicht Transparenz-Gesichtpunkte im vordergrund stehen, sondern in erster Linie die möglichst flexible Nutzung möglichst großer Ressourcen-Kapazitäten, könnte ein entsprechend angepaßtes Lastverwaltungs-System durchaus interessante Möglichkeiten bieten.

Kapitel 5

Entwicklung eines Lastbalancierungs-Modells als Grundlage für das Management

Es soll nun ausgehend von vorhandenen Lastbalancierungsprodukten, Algorithmen und Systemen ein Modell der Lastbalancierung und Datenmigration entworfen werden, das die Grundlage bildet für die Konzeption des eigentlichen Managements.

Die Analyse von Lastbalancierungs-Systemen und Algorithmen in Kapitel 3 sollte den Funktionalitätsumfang von Produkten verdeutlichen. Dabei stellt sich die Frage nach der konzeptionellen Einordnung der Funktionen und Fähigkeiten in die Bereiche Management und Lastbalancierung bzw. Datenmigration. Welche der oben analysierten Eigenschaften eines Produktes im Bereich Lastverwaltung ist tatsächlich unter den Begriff Lastbalancierung bzw. Lastverwaltung einzuordnen - und welche der Eigenschaften gehören in den Bereich des Managements einer Lastverwaltungs- und Datenmigrations-Komponente?
Durch die Definitionen von "Lastverwaltung" und "Management einer Komponente eines verteilten Systems" wird die Trennung der Funktionalitäten und die Zuordnung zu den Bereichen leicht möglich: Unter "Lastverwaltung und Datenmigration" werden nur jene Funktionen zusammengefaßt, die in direktem Zusammenhang mit der Zuweisung von Aufträgen und Datensätzen zu Funktionseinheiten stehen. Dabei soll das in [Ludw92] vorgestellte Regelkreis-Schema der Lastverwaltung zugrunde gelegt werden.
Das "Management von Lastverwaltung und Datenmigration" dagegen soll jene Aufgaben umfassen, die mit der Verwaltung, Planung, Konfiguration, Steuerung, Überwachung und Fehlerbehebung der Lastverwaltungs- und Datenmigrations-Komponente befaßt sind.
Das nun vorgestellte Lastbalancierungsmodell gliedert sich in die Abschnitte "Funktionaltät der Lastbalancierung" und "Management-Aspekte in Lastbalancierungs-Systemen". Die Funktionaltität der Lastbalancierung kann in Anlehnung an das Regelkreismodell in die Bereiche "Lasterfassung", "Lastbewertung" und "Lastverschiebung" aufgeteilt werden. Nach einem Überblick über das Modell mit seinen Funktionsbereichen soll eine nähere Beschreibung der einzelnen Funktionsbereiche gegeben werden.

5.1 Überblick

1. Funktionalität von Lastbalancierung und Datenmigration:

    (a) Lasterfassung

    - Systembeobachtung und Ressourcenlastmessung
        - Beschaffung von Information über die Auslastung der Ressourcen und Maschinen-Verfügbarkeit (insbes. User-Aktivität)
        - Beschaffung von Information über die Verteilung von Prozessen und Daten
    - Beschreibung der System-Konfiguration
        - Beschaffung von Information über die Ressourcen-Konfiguration
    - Anwendungs- und Auftragsbeschreibung (Profile)
        - Beschaffung von Information über die zu balancierende Anwendung bzw. Aufträge
    - Protokollierungs-Funktionalität: "Vergangenheit"
        - Protokollierung des Systems anhand des Ist-Zustandes bzw. der Vergangenheit

    (b) Lastbewertung

    - Auftragsverwaltung/Prozeßverwaltung
        - Treffen von Lastbalancierungsentscheidungen bzgl. Prozessen / Aufträgen (Verteilung und Umverteilung)
        - Treffen von Lastbalancierungsentscheidungen bzgl. starten, stoppen und neustarten von Prozessen
    - Datenverwaltung
        - Treffen von Entscheidungen bzgl. der Verteilung, Umverteilung Replikation und Partitionierung von Daten
        - Treffen von Entscheidungen bzgl. des Löschens/Ungültigmachens von Datensätzen
    - Protokollierungs-Funktionalität: "Zukunft"
        - Erstellung eines (vorausschauenden) Planes zur Abwicklung der Lastbalancierungs-Aufgabe (vgl. auch "statische Lastbalancierung")

    (c) Lastverschiebung

    - Auftragsverwaltung/Prozeßverwaltung
        - Durchführung der Prozeßverlagerung (Unterstützung von Prozeßmigration) bzw. des Startens bzw. Stoppens von Prozessen, Checkpointing-Funktionalität
    - Datenverwaltung
        - Durchführung der Verteilung, Umverteilung, Replikation und Partitionierung der Datensätze
        - Durchführung des Löschens/Ungültigmachens von Datensätzen

2. weitere Funktionsbereiche von Lastbalancierung und Datenmigration:

(a) Parallel-Job-Unterstützung und Datenzugriffsverwalung
    - Auftragsverwaltung/Prozeßverwaltung
        - Mechanismen zur Unterstützung von Parallel-Job-Verarbeitung und Message-Passing
    - Datenverwalung
        - Konsistenthaltung von Daten, Replikationen und Partitionen
        - Synchronisation des Zugriffs auf Daten

(b) System-spezifische Gesichtspunkte
    - System-Schnittstelle
        - Hardware- und Software-Voraussetzungen
        - Schnittstelle zum verteilten Betriebssystem
        - Kooperation mit div. verteilter Software und anderen Lastbalancierungs-Systemen
        - Schnittstelle zu anderen Komponenten des vert. Ablaufsystems
        - Schnittstelle zur verteilten Anwendung
    - Weitere struktur-spezifische Aspekte
        - Dämonen der Lastbalanicierung
        - weitere Struktur-Merkmale

3. Management-Aspekte von Lastbalancierungs- und Datenmigrations-Systemen

(a) Konfiguration von Lastbalancierung und Datenmigration
    - Konfiguration der vorhandenen Ressourcen für die Lastbalancierung
    - Konfiguration der Benutzer des Lastbalancierungs- und Datenmigrations-Systems

(b) Mechanismen zu Fehlertoleranz, Fragen der Sicherheit
    - Mechanismen zur Fehlertoleranz
    - Mechanismen zu Fragen der Sicherheit

(c) Mechanismen zum Einsatz unterschiedlicher Strategien
    - Treffen von Entscheidungen über den Einsatz verschiedener Balancierungs-Strategien
    - Modifizierung des Lastbalancierungs-Systems entsprechend gewählter Strategie

(d) Protokollierung der Arbeitsweise des Lastbalancierungs-Systems
    - Protokollierung des Systems anhand des Ist-Zustandes bzw. der Vergangenheit

(e) Benutzer- und Systemadministrator-Schnittstelle
    - Kommandos zur Überwachung von Funktionen und Status des Lastbalancierungs-Systems
    - Kommandos zur Konfiguration des Systems
    - Kommandos zum Eingriff in das System
    - Unterscheidung Benutzer-Kommandos - Systemadministrator-Kommandos
    - Graphische Benutzer-Oberfläche (GUI)

5.2 Funktionalität der Lastbalancierungs- und Datenmigrations-Komponente

5.2.1 Lasterfassung

Die Lasterfassung stellt dieerste Komponente des Regelkreis-Modells dar. Sie beinhaltet die "information-policy" des Lastbalancierungs-Beschreibungsrahmens , dient der Entscheidungsvorbereitung und kann in die Bereiche

- Systembeobachtung und Ressourcenlast-Messung
- Beschreibung der System-Konfiguration
- Anwendungs- und Auftragsbeschreibung
- Protokollierung der Vergangenheit

unterteilt werden, auf die im Folgenden näher eingegangen werden soll. Die Systembeobachtung und Ressourcenlast-Messung betrifft die Auslastung der Ressourcen, die Aktivität von Benutzern und damit die Verfügbarkeit von Rechnern sowie die aktuelle Verteilung von Prozessen und Daten im System. Die Beschreibung der Systemkonfiguration betrifft alle Informationen über aktuelle System-Konfigurationen, insbesondere zur verteilung von Jobs und Daten in heterogenen Umgebungen. Die Anwendungs- und Auftragsbeschreibung umfaßt alle Requirements und Informationen über vermutliches Verhalten von Anwendungen und Jobs. Die Protokollierungs-Funktionalität umfaßt alle drei genannten Bereiche und repräsentiert die Information über die Vergangenheit des System-Verhaltens.

5.2.1.1 Systembeobachtung und Ressourcenlastmessung

1. Auslastung der Ressourcen und Benutzer-Aktivität:
   für alle Workstations im Pool:
   - Prozessor-Auslastung (Anzahl momentan aktiver Prozesse, Zeitscheiben)
   - Prozessor-Warteschlange (Anzahl Jobs)
   - Memory-Auslastung
   - Disk-Space-Auslastung
   - Kommunikationsverbindungs-Belastung (= busy time)
   - Lastbalancierungs-Warteschlange an diesem Rechner
   - diverse Größen (z.B. Swapping-Rate, Auslastung und Warteschlange von E/A-Geräten, Anzahl Jobs, die an der Workstation angekommen sind / die von der Workstation abgesendet worden sind ([StSi84] S52))
   - Benutzeraktivität und Tageszeit-Sensitivität (Uhr)
     (Verfügbarkeit bzw. Nichtverfügbarkeit von Ressourcen aufgrund definierter Ereignisse)

   für alle E/A-Geräte des Pools:

- Geräte-Auslastung
- Anzahl wartende Jobs

dabei sind folgende Aspekte zu berücksichtigen:

(a) Wie groß ist der Anteil der Auslastung bzgl. der für die Lastbalancierung zur Verfügung gestellten Größe? (vgl. Konfiguration der Ressourcen)

(b) Beobachtung aktueller Werte und Sammlung aggregierter Information (vgl. hohe bzw. geringe Änderungsdynamiken)

(c) Auswahl der Meßgrößen und Bestimmung der Meßperioden wird abhängig von der verwendeten Strategie durch die Mechanismen zum Einsatz unterschiedlicher Lastbalancierungs-Strategien

(d) Frage der Normalisierung der Meßgrößen aufgrund vorhandener Heterogenität (z.B. [FeZh86] S587; vgl. auch Konfigurations-Beschreibung)

2. aktuelle Verteilung von Prozessen und Daten im System:
   - Prozesse:
     - Welche Prozesse wo?
     - Welcher Vaterprozeß wo?
     - Welche Sohnprozesse wo?
     - Welche erzeugten Daten? Wo? Datengröße?
     - Eigenschaften und Zustand des Prozesses
       * Speicherplatzbedarf
       * bisherige Prozessor-Belastung (bisher verbrauchte CPU-Zeit)
       * voraussichtliche verbleibende Laufzeit
       * bisherige Anzahl an Migrationen auf andere Hosts (vgl. [StSi84] S51)
       * bisherige Datenzugriffe - Anz., Art (rw,...), Ort,
       * z.Zt. gehaltene Locks ([StSi84] S51)
       * bisherige Kommunikation mit anderen Prozessen
       * bisherige Unteraufrufe
       * Zustand: running / queued / suspended / completed / killed / checkpointed
   - Daten:
     - Original? Wo? Welche und wieviele Replikate? Wo?
     - Replikat? Wo? Wo zugehöriges Original?
     - Partition? Wo? Übrige Partitionen? Wo? Partition: Original oder Kopie?
     - Eigenschaften und Zustand des Datensatzes:
       * Speicherplatzbedarf
       * evtl. Zugriffsverfahren und Pfade
       * evtl. Konsistenzbedingungen
       * evtl. geforderte Verfügbarkeit
       * bisherige Zugriffshäufigkeit und -art (r/w)

5.2.1.2 Beschreibung der System-Konfiguration

Information über System-Konfiguration (mit Berücksichtigung heterogener Strukturen)

- für alle Workstations und E/A-Geräte: im Pool oder nicht?
- Heterogene Aspekte:
  für alle Workstations:
  - Prozessor-Typ, Multitasking/Queuing?, maximale Anzahl Prozesse, Geschwindigkeit
  - Betriebssystem-Plattform (Typ, Release-Version, div. Aspekte)
  - Art der Kommunikationsverbindungen (Physikalische Eigenschaften, Länge der Verbindung, Güte der Verbindung)
  - Liste der (physischen / logischen im Sinne des Lastbalancierungs-Algorithmus) Nachbarn der Workstation (vgl. [StSi84] S52)
  - Maschine als Submit-Host konfiguriert? (d.h. nur Absetzen von Jobs, keine Bearbeitung von Jobs auf dieser Maschine möglich)

  für alle E/A-Geräte:
  - Gerätetyp, Geschwindigkeit, E/A-Prozessorlast pro Zugriff
- Maschinen-Verfügbarkeit (Endbenutzer-Konfiguration bzw. Systemadministrator-Konfiguration):
  für alle Workstations und E/A-Geräte im Pool:
  - immer - zu bestimmten Zeiten - falls keine lokale Benutzer-Aktivität - nie (vgl. [KaNe93a] 5.2 bzw. [IBM93a] S1)
  - Konfiguration einer exklusiven CPU bzw. E/A-Gerät (für Spezialprogramme wie Benchmarking oder Real-Time-Applications) (vgl. [KaNe93a] 4.4.)
- Maximal vorhandene Kapazitäten / maximal für die Lastbalancierung vorhandene Kapazitäten :
  für alle Workstations:
  - vorhandener Plattenplatz
  - vorhandener Speicherplatz
  - vorhandene Swap-Größe
  - Prozessor-Leistung (in MHz, Zeitscheiben, CPU-Zeit, %, Runtime-Begrenzungen, maximale Anzahl parallel bearbeitbarer Jobs)
  - Warteschlangenlänge von Prozessor und Lastbalancierung

  für alle E/A-Geräte:
  - vorhandener Speicherplatz
  - Prozessor-Leistung

Diese Information, wie auch jede Art von Leistungskenngrößen von Prozessoren und Kommunikationsverbindungen sindin hegerogener Umgebung homogenisierbar, d.h. es ist möglich, diese Größen durch Gewichtung vergleichbar zu machen.

Div. Konfigurations-Information:

- Konfiguration von Servern

5.2.1.3 Anwendungs- und Auftragsbeschreibung

1. Anwendungsbeschreibung: Anwendungsrequirements und Informationen über vermutliches Verhalten
   (statisches und dynamisches Anwendungsprofil einer datenintensiven verteilten Anwendung nach [Beck93])
   - statische Beschreibung
     (Abschätzung gültig für den gesamten Verlauf der Anwendung)
     - Gliederung der Aufträge in Typen; Angaben pro Auftragstyp
       * mittlerer Aufwand an Rechenzeit
       * externe Datenzugriffe pro Ausführung
       * Anzahl an Unteraufrufen an verschiedene Serverklassen
       * Schreib- und Lese-Verhältnisse bei Zugriff auf globale Daten

       Globale Angaben:
       * Häufigkeit von Server-Aufrufen bestimmter Auftrags-Typen
       * Parallelität von Server-Aufrufen bestimmter Auftrags-Typen
       * aus Parallelität entstehende Kommunikationslast
       * Anwendungsname
   - dynamische Beschreibung
     - Einzelaufträge (Beschreibung erst möglich zum Aufrufzeitpunkt; vgl. Auftragsbeschreibung)
     - Auftragsgruppen
       * Reihenfolge-Beziehungen und Schleifen: Beziehungen zwischen Aufträgen
       * Abschätzen der Intensi-tät der Kooperation der Teilaufträge (Instanzenwahl und deren Kommunikationsverbindungen)
       * Abschätzen von entstehender Parallelität (Möglichkeit der Konfiguration von Servern)
       * Abschätzen von Reihenfolge-Abhängigkeiten (Wartezeiten auf synchrone Aufrufe)
       * Abschätzen von Anzahl und Verhältnis von Lese-/ Änderungs-Zugriffen auf Kontext-Partitionen (Kompromiß von Parallelität und Kontextverwaltungsaufwand)
2. Auftragsbeschreibung: Job-Requirements, Information über vermutliches Verhalten und maximal mögliches Verhalten

- Job-Name, Angaben zu Submitter, Input/Output/Error-Files, Initial-Directory, Shell, etc.
- Startzeitpunkt (Datum, Uhrzeit), vgl. [IBM93b] S20, [KaNe93a] 4.5, vorausschauende Planung von Lastbalancierung und Datenmigration
- Job-Priorität (= Priorität der Bearbeitung), vgl. [IBM93b] S20, [KaNe93a] 4.3.
- Job-Art (Batch-Job, paralleler Job, interactiver Job, redirectiver Job) [KaNe93a] S3f; Auftrags-Typ (siehe statische Anwendungsbeschreibung)
- Angaben zum Job-Verlauf: Halte-Status ([IBM93b] S20) Checkpointable ([IBM93b] S51), Restart yes/no ([IBM93b] S57)
- Abschätzungen der während der Auftragsbearbeitung entstehende Unteraufträge
- Ressourcen-Anforderungen (vgl. [KaNe93a] 4.8)
  - Ressourcen-Art
    - Architektur
    - Betriebs-System
    - Prozessor-Typ (vgl. auch [LiKe87]
    - spez. Maschinen oder Klasse von Maschinen gefordert
    - spez. Features gefordert
  - Minimal-Anforderungen
    - Rechenzeit-Bedarf
    - Disk-Space
    - Memory-Space
    - Swap-Space
    - spez. Angaben für parallele Jobs (vgl. [IBM93b] S20)
  - Limits
    - CPU-Limit (vgl. [IBM93b] S22)
    - Data-Limit (vgl. [IBM93b] S22)
    - max. Parallelität (vgl. [Beck92] S15)
    - max. Größe des Anwachsens des Programms (vgl. [IBM93b] S20)
    - Time-Limits=max. zulässige Laufzeit (vgl. [Beck92] S15, [KaNe93a] 4.2)
  - div. quantitative Angaben
    - mittlerer Rechenzeitbedarf
    - exklusive CPU notwendig? (vgl. [KaNe93a] 4.3.)
    - spezielle Ressourcen notwendig (vgl. [StSi84] S51)
    - Anzahl notwendiger Plattenzugriffe und r/w-Verhältnis
    - Anzahl notwendiger E/A-Zugriffe und r/w-Verhältnis
    - Auflistung aller voraussichtl. benötigten Datensätze
    - Auflistung aller voraussichtl. benötigten E/A-Geräte

3. Datenobjekt-Beschreibung

Die Anwendungs- und Auftrags-Profile haben dabei für die Lastbalancierung und Datenmigration folgende Bedeutung:

- finden der korrekten Zielarchitektur
- finden der geeigneten Ressourcen-Kapazitäten
- Einfluß auf max. mögliches Prozeßverhalten, z.B. bezüglich möglicher Prozeß-Zustände
- Unterstützung der güstigsten Verteilung - statisch in der Protokollierungs-Funktionalität und dynamisch zum Aufruf-Zeitpunkt

5.2.1.4 Protokollierung der Vergangenheit

Protokollierung von Ereignissen auf dem Netz sind für Lastbalancierung und Management von Interesse. Lastbalancierungs-seitig wird die Vergangenheit mitprotokolliert, um daraus eine verbesserte Entscheidungsfähigkeit des Algorithmus gewinnen zu können. Die Gewinnung von Statistiken und die Einbeziehung wiederkehrender Ereignisse können die Entscheidung über die Verteilung von Jobs günstig beeinflussen.

5.2.2 Lastbewertung

Die Lastbewertungs-Komponente als Kernstück des Algorithmus trifft mit Hilfe der Information aus der Lasterfassungs-Komponente Entscheidungen bzgl. der Aufträge und Daten im System und plant evtl. deren Placierung und Ausführung im voraus. Dabei verwendet sie die Teilstrategien "location-policy", "transfer-policy", "selection-policy" und "status-policy" der "decision-policy", sowie die "negotiation-policy", wie in dem Lastbalancierungs-Beschreibungs-Rahmen definiert.

5.2.2.1 Auftrags- und Prozeß-Verwaltung

1. Entscheidungsgrundlagen:
   (a) Auslastung der Ressourcen und User-Aktivität / Maschinen-Verfügbarkeit
   (b) System-Konfiguration (mit Berücksichtigung heterogener Strukturen)
   (c) Aktuelle Verteilung von Prozessen und Daten im System
   (d) Profile der Anwendung (Aufträge, Daten und deren Querbeziehungen)
2. Einflußfaktoren in die Entscheidung:
   - Datenverwaltung: z.B. Aufwand für Kontextverwaltung
   - Geeignet konfigurierte Teilmenge an Information der Lasterfassungs-Komponente
   - Mechanismen zum Einsatz unterschiedlicher Strategien: indirekt durch gewählte Strategie, d.h. durch die Wahl des Algorithmus und der Parameter der Entscheidungsfunktion
   - Einflußfaktoren auch durch die vorausschauende Planung von Lastbalancierung und Datenmigration aus dem Fahrplan der Lastbalancierung und durch System-Administrator-Spezifikation
   - Fehlertoleranz-Mechanismen
   - Benutzer-Schnittstelle: Entscheidung per Eingriff über geeignete Kommandos

- div. Einflußfaktoren, wie z.B.: Infos von kooperierenden Lastbalancierungs-Systemen oder Fairness-Kriterien

3. Entscheidung über:

   - Verteilung von Jobs ("transfer-policy", "location-policy", "selection-policy")
   - Umverteilung von Jobs (vgl. Checkpointing, "status-policy")
   - Starten, stoppen und neustarten von Jobs ("status-policy")
   - Ausführung wann? (Verzögerung auf bestimmte günstige Tages- bzw. Nachtzeit; "status-policy")

   Das vorzeitige Löschen von laufenden Jobs ist keine Entscheidung der Lastbalancierung, da im Normalfall Aufträge vollendet werden sollen. Neben Einzelentscheidungen über bestimmte Jobs können auch Entscheidungsbündel erforderlich sein, z.B. wenn das Placieren eines Jobs die Verlagerung anderer Jobs nötig macht. Das Duplizieren von Aufträgen unter dem Gesichtspunkt des Wettlaufs zwischen konkurrierenden Prozessoren soll hier nicht berücksichtigt werden.

5.2.2.2 Datenverwaltung

1. Entscheidungsgrundlagen:

   (a) Auslastung der Ressourcen und User-Aktivität (Speicherkapazitäten von Platten und Speicher)

   (b) System-Konfiguration

   (c) Aktuelle Verteilung von Prozessen und Daten im System

   (d) Anwendung (Jobs, Daten und Querbeziehungen) (Profile von Aufträgen und Auftragsgruppen, insbes. Schreib- und Lese-Zugriffe auf Daten; Aktuelle Anfragen von Aufträgen)

2. Einflußfaktoren in die Entscheidung:

   - Geeignet konfigurierte Teilmenge an Information der Lasterfassungs-Komponente
   - Einfluß-Faktoren durch die vorausschauende Planung von Lastbalancierung und Datenmigration.

3. Entscheidung über:

   - Placierung von (neu entstandenen) Originaldatensätzen ("transfer-policy", "location-policy", "selection-policy")
   - Umverteilung von Original-Datenbeständen ("transfer -policy", "location-policy", "selection-policy")
   - Anlegen einer Kopie/Partition; Wahl des geeigneten Datengranulats ("status-policy", "selection-policy")
   - Placierung von Kopie bzw. Partition ("transfer -policy", "location-policy", "selection-policy")

- Umverteilung von Kopie bzw. Partition (”transfer -policy”, ”location-policy”, ”selection-policy”)
- Ungültigmachen/Löschen der Kopie, Partition bzw. des Originals (”status-policy”, ”selection-policy”)

5.2.2.3 Vorausschauende Planung von Lastbalancierung und Datenmigration

Die vorausschauende Planung von Lastbalancierung und Datenmigration kann zum einen die Erstellung eines (vorausschauenden) Planes zur Abwicklung der Lastbalancierungs-Aufgabe (nach [Ludw92] S39 ”Mapping”, d.h. Rechnerknotenzuteilung genannt) bedeuten. HiCon beispielsweise besitzt diese Möglichkeit durch einen System-Fahrplan, der auch automatische Anpassungen zur Laufzeit erlaubt (vgl. [Beck93] S18f, [Beck92] S23). Andererseits besteht in gängigen Lastverwaltungs-Systemen die Möglichkeit, die Ausführung eines Jobs auf ein bestimmtes Datum und eine bestimmte Uhrzeit zu verzögern. (vgl. [KaNe93a] 4.5.)

5.2.3 Lastverschiebung

Die Entscheidungsumsetzung der in der Lastbewertungs-Komponente getroffenen Entscheidungen wird durch die Lastverschiebe-Komponente des Regelkreismodells durchgeführt.

5.2.3.1 Auftrags- und Prozeß-Verwaltung

Unterstützung der Entscheidungsumsetzung:

- Unterstützung von Prozeßmigration
- Checkpointing-Funktionalität
- Durchführung des Startens und Stoppens von Prozessen

5.2.3.2 Datenverwaltung

Unterstützung der Entscheidungsumsetzung:

- Placierung von (neu entstandenen) Originaldatensätzen
- Umverteilung von Original-Datenbeständen
- Anlegen einer Kopie/Partition; Einstellung des geforderten Datengranulats
- Placierung von Kopie bzw. Partition
- Umverteilung von Kopie bzw. Partition
- Ungültigmachen/Löschen der Kopie, Partition bzw. des Originals

5.2.4 Parallel-Job-Unterstützung und Datenzugriffsverwaltung

Neben dieser Funktionalität des eigentlichen Regelkreis-Modells müssen reale Lastverwaltungs-Systeme die Ergebnisse ihrer Entscheidung weiter unterstützen. Dies betrifft sowohl Prozesse und Prozess-Gruppen wie auch Daten-Sätze.

5.2.4.1 Auftrags- und Prozeß-Verwaltung

Parallel-Job-Unterstützung

- Unterstützung paralleler Umgebungen
- Unterstützung von Systemaufrufen entfernt ausgeführter Prozesse (Kommunikation mit dem lokalen Rechner; vgl. z.B. "shadow-prozesse bei Condor oder Load Leveler)

5.2.4.2 Datenverwaltung

1. Synchronisation des Zugriffs auf Daten
   (a) Anmeldung des Zugriffs auf statische Datensätze
   (b) Sperrung des Datensatzes
   (c) Zugänglichmachen des Datensatzes an anfragenden Auftrag
   (d) Freigabe nach Jobende oder auf Signal durch Job o.ä.
2. Konsistenthaltung von Datenpartitionen und Datenkopien

5.2.5 Struktur-spezifische Gesichtspunkte

5.2.5.1 System-Schnittstelle

Ein Lastverwaltungs-System arbeitet nicht unabhängig von anderen Teilen des verteilten Systems sondern benötigt zur korrekten Arbeitsweise die Interaktion zu verschiedenen Komponenten. Es müssen also verschiedene Schnittstellen notwendigerweise oder optional vorhanden sein:

1. Schnittstelle zur verteilten Anwendung
2. Schnittstelle zum verteilten Betriebs-System
3. Schnittstelle zum Kommunikations-System
4. weitere notwendige Hardware-Voraussetzungen sind z.B. File-Systeme (z.B. AFS und NFS) oder Name-Server (z.B. NIS; vgl. [KaNe93a] 1.3)
5. Unterstützung von parallelen Umgebungen durch Schnittstellen zu Interprozess-Communication-Software, wie zum Beispiel PVM, Linda, Express oder p4 (vgl [KaNe93a] 2.4)
6. Schnittstellen zur Kooperation mit anderen Lastbalancierungs-Systemen (Load Leveler bietet z.B. die Möglichkeit NQS-Scripts zu verarbeiten und abzuschicken; [IBM93b])

5.2.5.2 Weitere struktur-spezifische Aspekte

Ein Lastbalancierungs-System besteht aus Dämonen, die über das Netz miteinander kommunizieren. Je nach der Struktur des Algorithmus ist das System dabei zentral oder dezentral organisiert. Bei der Verwendung von zentralen Verfahren existieren Dämonen, die einen zentralen Master bilden, der über Aufträge Entscheidungen trifft oder zumindest, wie bei Condor, Ressourcen an anfragende Rechner zuweist. Durch weitere Dämonen auf den Rechnern wird die Lastbalancierung und das Management der Lastbalancierung möglich. Im dezentralen Fall wird der Lastbalancierungs-Algorithmus und das System durch gleichberechtigte Dämonen realisiert.

5.3 Management-Aspekte in Lastbalancierungs- und Datenmigrations-Systemen

5.3.1 Konfiguration von Lastbalancierung und Datenmigration

1. Konfiguration der vorhandenen Ressourcen für die Lastbalancierung:

   - Maschine an Workstation-Pool hinzufügen / Maschine vom Pool entfernen / Maschine als Submit-Host konfigurieren (d.h. nur das Absetzen von Jobs, jedoch kein Bearbeiten von gescheduleten Jobs auf diesem Rechner möglich)
   - Konfiguration der Maschinen-Verfügbarkeit
   - Konfiguration der maximal für die Lastbalancierung zur Verfügung stehenden Kapazitäten
   - Konfiguration zur Anpassung des Lastbalancierungs-Systems an heterogene Umgebungen (z.B. Aliase für Home-Verzeichnisse die in verschiedenen Rechnern in unterschiedlichen Pfaden liegen; vgl. CODINE, [GEN94a])
   - weitere Konfigurationen: z.B.
Konfiguration von Serverinstanzen, Erzeugung und Terminierung von Serverinstanzen (vgl. HiCon)

   Konfigurationsmöglichkeiten

   - automatisch oder über Benutzer- und Systemadministrator-Schnittstelle
   - zu Beginn oder zur Laufzeit (vgl. [KaNe93a] 5.1)
   - durch Systemadministrator oder vom Benutzer (vgl. [KaNe93a] 5.3)

2. Konfiguration der Zugriffsmöglichkeiten der Benutzer auf vorhandene Ressourcen (vgl. [KaNe93a] 4.1)
Begrenzung von:

   - Zugriff auf best. Maschinen des Pools, auf best. Architekturen oder Warteschlangen;
   - Zugriff zu bestimmten Tageszeiten
   - CPU-Zeit, Memory-Größe, Disk-Space, Swap-Space, Job-Run-Time

- Ausführung bestimmter Jobs bzw. best. Anzahl von Jobs
- Priorität der Bearbeitung von Jobs: Prioritäten-Hierarchie von Benutzern

5.3.2 Mechanismen zu Fehlertoleranz und Fragen der Sicherheit

1. Fehlertoleranz

   - Erkennung fehlerhafter Maschinen
     - keine Kommunikation mit Master-Scheduler
     - Dämonen laufen nicht / Maschine läuft nicht
   - Rerouting von Jobs, die an diese Maschine gesendet werden sollten
   - Checkpointing und Migration von Jobs von der fehlerhaften Maschine auf andere bzw. Neustart des Jobs
   - Versuch des Recoverys
     - Dämon recovern
     - Master-Scheduler Recovery
   - Vermeidung von "Single-Point-Failure-Problems"
     Abhängigkeiten von einzelnen Master-Schedulern vermeiden (z.B. durch optionale, passive Master-Scheduler, vgl. [GrSn93])
   - weitere Fehlertoleranz-Mechanismen

2. Fragen der Sicherheit

   - Sicherheits-System des Unix-Operating-System
   - Authentifizierungs-Mechanismen (z.B. Kerberos von MIT)
   - Authorisierungs-Mechanismen(z.B. durch OSF/1 DCE Zugangs-Kontroll-Listen - ACLS)
   - Validation-Services
   - Sicherstellung der Nachrichten-Integrität (z.B. durch verschlüsselte Checksummen) und Datengeheimhaltung (durch Verschlüsselungs-Verfahren)
   - weitere Sicherheits-Mechanismen

5.3.3 Mechanismen zum Einsatz unterschiedlicher Strategien

1. Verwaltung eines Katalogs möglicher Strategien
2. Verwaltung eines Katalogs von Kriterien, unter denen eine Strategie besonders gut angewendet werden kann
3. Beobachtung des Systems für den Einsatz verschiedener Strategien:

   - Protokollierung der Vergangenheit, statische Beschreibung der Anwendung (Soll-/Ist-Vergleich, z.B. erreichte Laufzeiten, Zukunft)
   - Auslastung der Ressourcen sowie

- Ressourcenverwaltung: Beschreibung der Konfiguration der Ressourcen

4. Verbindung zur Benutzerschnittstelle zur Kommando-gesteuerten Auswahl von Strategien

5. evtl. Verbesserung des Kriterien-Katalogs anhand Beobachtungen des Systems

6. Entscheidung:

   - Strategie-Wechsel?
   - Auswahl geeigneter Strategie anhand Systembeobachtung, Kriterienkatalog und Benutzer-Schnittstelle
   - Auswahl geeigneter Meßintervalle der Lasterfassungs-Komponente

7. Implementierung der Strategie: Anpassung von

   - Auftragsverwaltung, Prozeßverwaltung
   - evtl Datenverwaltung
   - beobachtete Größen der Ressourcenlast-Messung
   - Größe des Meßintervalls der Ressourcenlast-Messung
   - evtl. weitere Informationsbeschaffungs-Komponenten

   Methode der Strategie-Implementierung:

   - automatisch oder per Eingriff in das System
   - Anpassung zum Zeitpunkt der Compilierung des Systems (wie z.B. bei DQS, vgl. [KaNe93a]) oder zur Laufzeit (vgl. Strategische Lastbalancierungs-Komponente von HiCon, [Beck93])

5.3.4 Protokollierung der Arbeitsweise des Lastbalancierungs-Systems

Eine Aufzeichnung der Aktivitäten der Lastbalancierung ist für Lastbalancierung und Management gleichermaßen wichtig. Management-seitig gibt es verschiedene Gründe, die Arbeitsweise von Lastverwaltung und Datenmigration zu protokollieren. In Lastverwaltungs-Produkten wird die Vergangenheit in erster Linie bezüglich des Ressourcen-Verbrauchs beendeter Aufträge und Prozesse für das Abrechnungs-Management der Jobs (Aufgeschlüsselt nach Jobs und Workstations) aufgezeichnet. (vgl [KaNe93a] Kap. 4.10. und [IBM93b])

5.3.5 Benutzer- und Systemadministrator-Schnittstelle

An der Benutzer- und Systemadministrator-Schnittstelle können verschiedene Aktivitäten zur Interaktion und Konfiguration des Lastverwaltungs-Systems durchgeführt werden. Die über diese Schnittstelle angebotene Funktionalität kann als Management-Funktionalität betrachtet werden, die Benutzern und Systemadministratoren in unterschiedlichem Umfang zur Verfügung gestellt wird. Die Benutzer- und Systemadministrator-Schnittstelle kann allgemein in die Bereiche Überwachung, Konfiguration und Eingriff/Steuerung eingeteilt werden.

Das Absetzen von Aufträgen wird gesondert als lastbalancierungs-spezifische Funktionalität behandelt.

- Absetzen von Aufträgen/Jobs:
  Das Absetzen von Aufträgen und Jobs wird in den meisten Batch-Queuing Systemen über einen entsprechenden Befehl und ein zugehöriges Job-Script-File abgewickelt, das angibt, welche Anforderungen und Charakteristiken der abgesetzte Job hat.

- Überwachung:
  Information über den momentanen und vergangenen Status des Systems wird über eine graphische Benutzeroberfläche angezeigt oder durch interaktive Kommandos abgefragt. Dabei werden verschiedene Aspekte berücksichtigt:

  - Information kann angezeigt werden über den Status laufender Jobs, über Warteschlangen des Lastverwaltungs-Systems, über Rechner und deren Benutzer, sowie über Nachrichten (Events) des Lastverwaltungs-Systems.
  - Accounting-Information gibt Auskunft über die Vergangenheit der oben genannten Elemente; Statistiken geben Information über aggregierte Größen der oben genannten Elemente
  - Information kann angezeigt werden über alle Instanzen oben angegebener Elemente oder nur über bestimmte Instanzen (z.B. Information über alle Jobs oder nur über die Jobs auf einem bestimmten Rechner)
  - Informaiton kann aufgeschlüsselt werden nach verschiedenen Kriterien wie z.B. Rechnerarchitektur, Gruppen etc.
  - es können unterschiedlich differenzierte Informations-Ausgaben bezüglich der oben genannten Elemente ausgegeben werden. (z.B. Standard Listing oder langes Listing über Rechner)

- Konfiguration:
  Die Konfiguration des Lastverwaltungs-Systems wird über geeignete Kommandos sowie über Konfigurations-Files durchgeführt. Konfiguration betrifft Rechner, Lastverwaltungs-Warteschlangen (Queues) sowie die Einrichtung von Benutzern. Jedes dieser Objekte kann zu Gruppen mit unterschiedlichen Merkmalen zusammengefaßt werden. Wesentliche Merkmale sind u.a. Prioritäten, Rechte oder Beschränkungen.

- Eingriff/ zur Steuerung:
  Die Steuerung des Lastverwaltungs-Systems betrifft vor allem Aufträge/Jobs, Dämonen des Lastbalancierungs-Systems sowie die Rechner des Lastverwaltungs-Pools. Jobs können neu gestartet, gelöscht oder verschoben werden; Dämonen können gestartet, angehalten oder zu bestimmten Aktionen veranlaßt werden; Rechner können z.B. bzgl. ihrer Verfügbarkeit oder ihrer Priorität modifiziert werden.

Kapitel 6

Management von Lastbalancierung und Datenmigration

6.1 Überblick über das vorgestellte Konzept

Es soll nun ein Konzept des Managements von Lastbalancierung und Datenmigration vorgestellt werden. Wie in Abbildung 6.1 angedeutet, kann das Management dabei in drei Bereiche aufgeteilt werden:

1. Auf der Ebene der Benutzer umfaßt das Management Bereiche wie die Administration und Konfiguration der Benutzer genauso wie z.B. die Überwachung von Sicherheits-Aspekten oder die Verwaltung von Abrechnungs-Konten.

2. Auf der Ebene des verteilten Ablauf-Systems umfaßt das Management die Verwaltung der eigentlichen Lastverwaltungs-Software. Dabei läßt sich eine Aufteilung der Lastbalancierung und Datenmigration in die logischen Komponenten Informations-Erfassung, Entscheidungs-Findung sowie Entscheidungs-Umsetzung (bzw. Lasterfassung, Lastbewertung und Lastverschiebung nach dem Regelkreis-Modell nach [Ludw92]) sowie einer zusätzlichen Komponente Ergebnis-Unterstützung (die sich wie im Lastverwaltungs-Modell des vorigen Kapitels angedeutet mit der Unterstützung von Parallelen Umgebungen bzw. lokalen Zugriffen entfernter Prozesse einerseits und mit der Konsistent-Haltung und Zugriffs-Synchronität der Daten andererseits befaßt) durchführen. Innerhalb dieser Unterteilung können die Strategie-Komponenten des einheitlichen Beschreibungsrahmens eine weitere Modularisierung unterstützen.

3. Auf der Ebene des verteilten Rechen-Systems befaßt sich das Management von Lastbalancierung und Datenmigration mit der Verwaltung und Überwachung der dieLastverwaltung betreffenden Ressourcen und Objekte; hier sind auch Zusammenhänge zwischen Netzmanagement und dem Management von Lastbalancierung und Datenmigration zu untersuchen.

Nach [Hege93] stellt eine Netzmanagement-Architektur ein allgemeines Rahmenwerk dar, das den Anforderungen eines integrierten Managementkonzepts mit dessen Voraussetzungen:

- Interpretierbarkeit der Information der zu managenden Komponenten in herstellerunabhängiger Weise

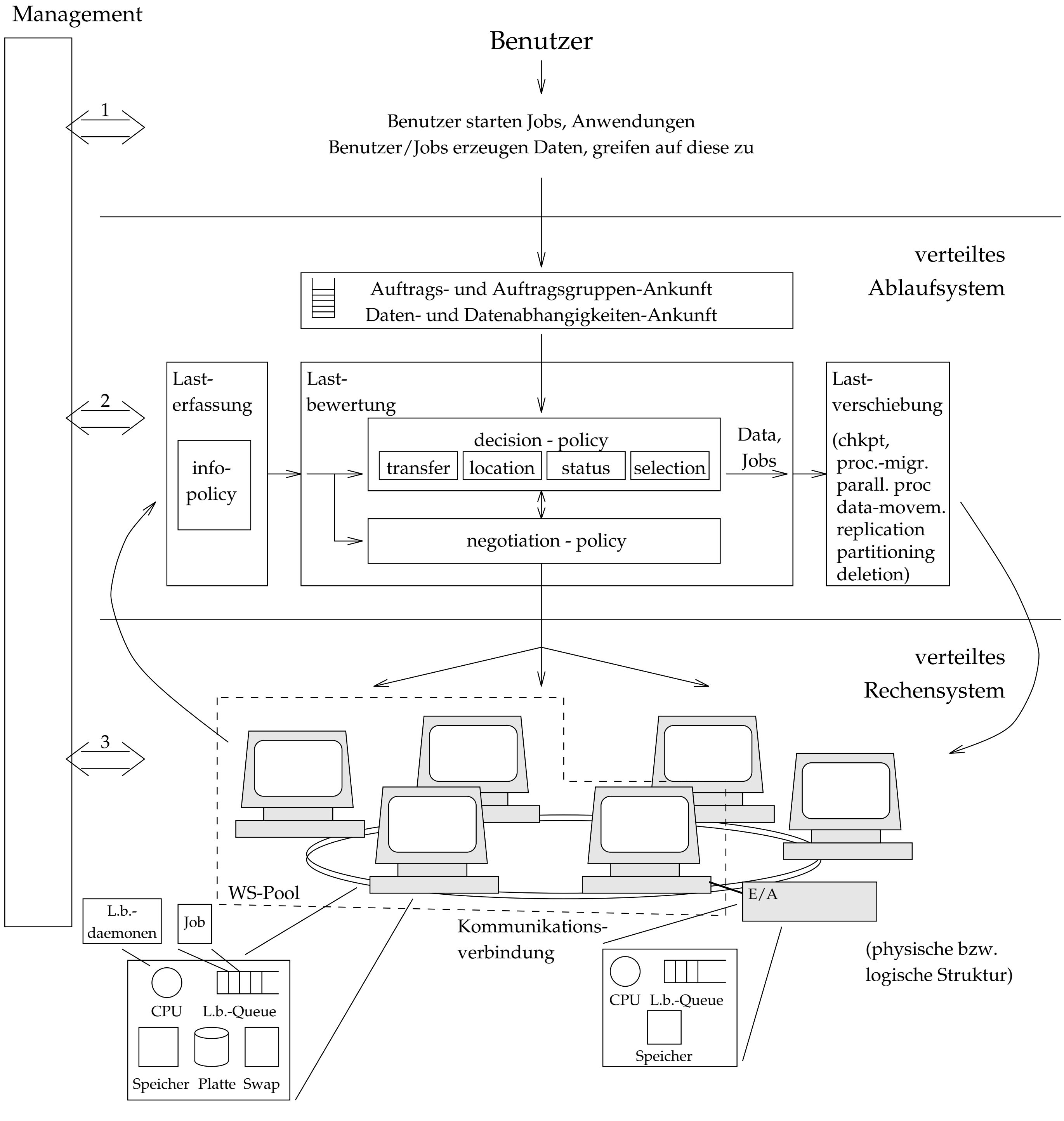

Abbildung 6.1: Überblick über das Management-Konzept: Struktur und Objekte

- Zugangsmöglichkeit zu dieser Information über wohldefinierte Schnittstellen und Protokolle

möglichst gerecht werden soll. Es soll diese Gesichtspunkte konzeptionell in einer solchen Weise umfassen, daß daraus bei gegebenem realem Netz mit entsprechenden Betreiberanforderungen ein konkretes Managementsystem entwickelt werden kann. Eine Netz-Management-Architektur, die für ein integriertes Netz-Management in heterogener Umgebung geeignet sein soll, muß nach [Hege93] zu folgenden Teilaspekten entsprechende Modelle bereitstellen:

- Beschreibung von Management-Objekten (Informations-Modell)
- Behandlung und Unterstützung von Organisationsaspekten (Organisations-Modell)
- Beschreibung von Kommunikationsvorgängen zu Management-Zwecken (Kommunikations-Modell)
- Strukturierung von Management-Aufgaben (Funktions-Modell)

Es soll daher zunächst im Informations-Modell des Lastbalancierungs-Managements die Management-Objekte beschrieben werden, die für das Management von Lastbalancierung und Datenmigration relevant sind. Im Anschluß daran sollen wesentliche Organisationsaspekte behandelt werden, die für die daran anschließende Beschreibung des Funktions-Modells wichtige Rahmenbedingungen liefern. Innerhalb des Funktions-Modells soll ausführlicher eingegangen werden auf die wesentlichen Funktionen Konfigurations-Management, Fehler-Managmenet, Leistungs-Management, Abrechnungs-Management sowie Sicherheits-Management. Abschließend soll noch kurz eine Beschreibung von Besonderheiten bei einer Schnittstelle zwischen Lastverwaltung und Datenmigration und dem Management-System gegeben werden.

6.2 Entwurf eines Informations-Modells

Wesentliches Element des Informations-Modells ist die Beschreibung der Management-Objekte (MO's). Nach [Hege93] versteht man unter einem Management-Objekt die Managmenet-Information, die aus Management-Sicht ein zu verwaltendes, zu überwachendes oder zu steuerndes Betriebsmittel beschreibt. MO's sind also für Management-Zwecke geeignete Abstraktionen von Ressourcen.
Wesentliche Management-Objekte innerhalb des Managements von Lastbalancierung und Datenmigration sind:

- Rechner-Knoten
- Ressourcen, die auf diesen Rechner-Knoten innerhalb der Lastverwaltung angesprochen werden, d.h.
  - CPU
  - Speicher
  - Platte
  - Swap-Speicher

- Ein-/Ausgabe-Bausteine
- Kommunikations-Verbindungen
- logische Komponenten der Lastbalancierung und Datenmigration
  - Lasterfassungskomponente mit "information-policy"
  - Lastbewertungskomponente mit
    - "negotiation-policy" und
    - "decision-policy" (und deren Teil-Komponenten "transfer-policy", "location-policy", "selection-policy" sowie "status-policy")
  - Lastverschiebekomponente
  - Ergebnis-Unterstützungs-Komponente mit Funktionalität wie im Lastverwaltungs-Modell beschrieben
  - Eingangs-Warteschlange
- physische Komponenten der Lastbalancierung und Datenmigration
  - System-spezifische Dämonen des Lastverwaltungs-Systems
  - evtl. Master-Dämon(en) des Lastverwaltungs-Systems
  - evtl. Lastbalancierungs-eigene Warteschlagen, weiterhin kurz Queues genannt ([GEN94b])
  - Schnittstellen zu übriger Software des verteilten Ablaufsystems
- Benutzer und Verwalter des Lastbalancierungs- und Datenmigrations-Systems
- verteilte Anwendungen, die mittels der Lastverwaltung gescheduled werden
- Jobs, aus denen eine verteilte Anwendung bestehen kann, und die mittels der Lastverwaltung verteilt werden
- Daten-Objekte, die mittels der Datenmigrationkomponente verteilt und verwaltet werden

Es soll möglich sein, Objekte zu Gruppen und Klassen zusammenzufassen:

- Eine Gruppe sei dabei eine Menge mit Namen, die Objekte einer Sorte enthalten kann
- Eine Klasse sei dabei eine durch Eigenschaften spezifizierte Menge mit Namen, die Objekte einer Sorte enthalten kann

In den nun folgenden Abschnitten werden die zu managenden Objekte näher umschrieben. Dies soll in der von OSI vetretenen objekt-orientierten Beschreibung von Objekte geschehen. Dabei sollen je nach Objekt Angaben gemacht werden zu:

- Objekten, die in diesem Objekt enthalten sind
- Eigenschaften dieses Objekts anhand eines geeigneten Profils

- mögliche Zustände dieses Objektes
- mögliche Typen, die von diesem Objekt existieren
- aktuelle Zustände
- mögliche Aktivatoren, die auf dieses Objekt angewendet werden können
- Informationen, die die Vergangenheit des Objektes betreffen
- mögliche Events, die dieses Objekt absenden kann

Rechner-Knoten

- Enthaltene Objekte:
  CPU, Speicher, Platte, Swap, Kommunikations-Verbindungen
  physische Lastverwaltungs-Objekte: Dämonen, Queues, Schnittstellen
  "flüchtige" Objekte: Jobs, Daten
- Rechner-Profil:
  konstantes Profil
  - Rechner-Id, Rechner-Name
  - Besitzer
  - Architektur, Betriebssystem-Plattform
  - Id und Namen enthaltener Objekte (s.o.)

  variables Profil
  - Rechner-Typ (s.u.)
  - Rechner-Priorität (bzgl. der Auswahl durch die Lastverwaltung)
  - Lastverwaltungs-Domäne (WS-Pool): Im Pool/nicht im Pool
  - Administrations-Domäne
  - Sicherheits-Domäne
  - physische und logische (nach dem Lastbalancierungs-Algorithmus) Nachbar-Rechner-Knoten
  - Benutzer-Beschränkungen:
    * Beschränkungen für bestimmte Benutzer, Benutzer-Typen Benutzer-Gruppen
    * Prioritäten bestimmter Benutzer, Typen von Benutzern oder Benutzergruppen
    * Rechte auf diesem Rechner
    * Maximale Kapazitäten für Benutzer, Typen oder Gruppen von Benutzern
    * Beschrnkungen von Jobs oder Queues für bestimmte Benutzer, Benutzer-Typen oder -Gruppen

    * Verfügbarkeit mit Zeitangaben
    * Authentifizierungs- und Authorisierungs-Mechanismen
  - Beschränkungen für Jobs, die von der Lastverwaltung verschoben werden
    * Beschränkungen für bestimmte Job-Typen, Job-Gruppen oder Job-Klassen
    * Prioritäten der Bearbeitung auf dieser Workstation
    * Maximale Kapazitäten für Jobs
  - Beschränkungen für Datensätze auf dieser Maschine
    * Beschränkungen für bestimmte Daten-Klassen oder Daten-Gruppen
    * Prioritäten der Datenhaltung für bestimmte Daten-Klassen oder Gruppen
    * Maximale Kapazitäten für bestimmte Daten-Klassen oder Gruppen

- Rechner-Status:
  momentane Konfiguration der Verfügbarkeit
  momentaner Status der Verfügbarkeit
  momentaner Zustand der Nutzung

- Informationen über die Vergangenheit des Rechners:
  - Anzahl Jobs, die von diesem Rechner abgesendet wurden
  - Anzahl Jobs, die an diesem Rechner angekommen sind

- Definition von möglichen Zuständen:
  1. Zustände der Verfügbarkeit:
     - verfügbar
     - nicht verfügbar
  2. mögliche Verfügbarkeits-Konfigurationen:
     - immer
     - zu bestimmten Zeiten (mit Zeitangaben)
     - falls keine Benutzer-Aktivität
     - nie
  3. mlgiche Zustände der Nutzung:
     - running
     - idle
     - down

- Definition von Rechner-Typen:
  - Master-Host
  - Execution-Host (führt Jobs aus bzw. speichert Datensätze, die von der Lastverwaltung gescheduled wurden)
  - Submit-Host (kann Jobs oder Daten an die Lastverwaltung senden)
  - beliebige Kombination oben genannter Typen

Ein-/Ausgabe-Baustein

- Enthaltene Objekte:
  CPU, Speicher
  physische Lastverwaltungs-Objekte: Dämonen, Queues, Schnittstellen
  "flüchtige" Objekte: Jobs oder Daten

- Ein/Ausgabe-Baustein-Profil:
  konstantes Profil

  - Ein/Ausgabe-Baustein-Id, Name, Typ
  - Besitzer
  - Geschwindigkeit
  - Id und Namen enthaltener Objekte (s.o.)

  variables Profil

  - Priorität (bzgl. der Auswahl durch die Lastverwaltung)
  - Lastverwaltungs-Domäne (WS-Pool): Im Pool/nicht im Pool
  - Administrations-Domäne
  - Sicherheits-Domäne
  - Benutzer-Beschränkungen:
    * Beschränkungen für bestimmte Benutzer, Benutzer-Typen Benutzer-Gruppen
    * Prioritäten bestimmter Benutzer, Typen von Benutzern oder Benutzergruppen
    * Rechte
    * Maximale Kapazitäten für Benutzer, Typen oder Gruppen von Benutzern
    * Beschrnkungen von Jobs oder Queues für bestimmte Benutzer, Benutzer-Typen oder -Gruppen
    * Verfügbarkeit mit Zeitangaben
    * Authentifizierungs- und Authorisierungs-Mechanismen
  - Beschränkungen für Jobs, die von der Lastverwaltung verschoben werden
    * Beschränkungen für bestimmte Job-Typen, Job-Gruppen oder Job-Klassen
    * Prioritäten der Bearbeitung
    * Maximale Kapazitäten für Jobs
  - Beschränkungen für Datensätze auf dieser Maschine
    * Beschränkungen für bestimmte Daten-Klassen oder Daten-Gruppen
    * Prioritäten der Datenhaltung für bestimmte Daten-Klassen oder Gruppen
    * Maximale Kapazitäten für bestimmte Daten-Klassen oder Gruppen

- E/A-Status:
  momentane Konfiguration der Verfügbarkeit
  momentaner Status der Verfügbarkeit
  momentaner Zustand der Nutzung

- Informationen über die Vergangenheit des E/A-Gerätes:

  – Anzahl Jobs, die von diesem Rechner abgesendet wurden
  – Anzahl Jobs, die an diesem Rechner angekommen sind

- Definition von möglichen Zuständen:

  1. Zustände der Verfügbarkeit:
     - verfügbar
     - nicht verfügbar
  2. mögliche Verfügbarkeits-Konfigurationen:
     - immer
     - zu bestimmten Zeiten (mit Zeitangaben)
     - falls keine Benutzer-Aktivität
     - nie
  3. mlgiche Zustände der Nutzung:
     - running
     - idle
     - down

CPU

- Eigenschaften der CPU:

  – Prozessor-ID, Typ
  – Maximale Anzahl parallel ausführbarer Prozesse
  – Geschwindigkeit
  – Maximale Länge der Prozessor-Warteschlange

- variables Profil der CPU:

  – maximal für die Lastverwaltung verfügbare CPU-Leistung
  – exklusive Konfiguration einer CPU

- CPU-Status:

  – Anzahl Jobs in der Prozessor-Warteschlange
  – Anzahl momentan aktiver Prozesse

Speicher

- Speicher-Größe
- maximal für die Lastverwaltung und Datenmigration verfügbare Speicher-Größe (hard-limit und soft-limit)
- momentan belegter Speicher-Platz

Platte

- Platten-Platz
- maximal für die Lastverwaltung und Datenmigration verfügbare Platten-Größe (hard-limit und soft-limit)
- momentan belegter Plattenplatz

Swap-Speicher

- Swap-Größe
- maximal für die Lastverwaltung und Datenmigration verfügbare Swap-Größe (hard-limit und soft-limit)
- Swapping-Rate

Kommunikations-Verbindung

- Eigenschaften der Kommunikations-Verbindung:
  - maximale Übertragungskapazitäten
  - Länge
  - Güte der Verbindung
  - TCP/IP Subnetz, in dem die Kommunikations-Verbindung enthalten ist
  - evtl. weitere Physikalische Eigenschaften
- Status der Kommunikations-Verbindung:
  - momentane Auslastung der Kommunikations-Verbindung (busy-time)
  - Kommunikations-Verbindung up/down
- Angaben über die Vergangenheit:
  - Auslastung der Kommunikations-Verbindung über die Zeit

Lastbalancierung und Datenmigration

- Folgende Objekte sind in dem Objekt "Lastbalancierung und Datenmigration" enthalten:

- physische Objekte der Lastverwaltung (Dämonen, Queues, Schnittstellen)
  - logische Objekte der Lastverwaltung (Lasterfassung, Lastbewertung, Lastverschiebung, System-Warteschlange, Ergebnis-Unterstützung)

- Eigenschaften des Objekts:
  - Betreiber-Name (zuständiger Verwalter)
  - System-Name
  - Zielfunktion
  - Leistungsgrenzen

- Zustand des Objektes:
  active - down - idle

- Aktivatoren: shutdown, reboot, fasthalt

physische Objekte der Lastbalancierung und Datenmigration

- Dämonen und Master-Dämon
  - Dämonen-Typ (hersteller-abhängig)
  - Dämonen-Zustand (idle, running, down)
  - Aktivatoren: create, kill, shutdown, reboot

- Queues
  - Queues enhalten Job-Objekte
  - Eigenschaften von Queues:
    - Queue-ID, Name, Typ
    - Länge der Queue
    - evtl. Besitzer (falls anders als Rechner-Besitzer) - Queue-Priorität - Benutzer-Beschränkungen (falls anders als bei Rechner)
  - Zustand von Queues:
    - momentane Anzahl Jobs in der Queue - Verfügbarkeit, Konfiguration der Verfügbarkeit - momentaner Status der Nutzung
  - weitere Aktivatoren:
    create, delete

logische Objekte der Lastbalancierung und Datenmigration

- Lasterfassung:
  - eingestellte Größen der Informations-Beschaffung

- Frequenz des Information-Updates bzw. Zeitpunkte der Informations-Erneuerung
- Zustand der Nutzung der Lasterfassungs-Komponente
- Inhalt des Log-Files bzgl. der Protokollierung der Vergangenheit

- Lastbewertung: Informationen von "negotiation-policy", "transfer-policy", "location-policy", "selection-policy", "status-policy":
  - momentan eingesetzter Algorithmus der Teil-Komponente
  - Algorithmen-abhängige Parameter der Teil-Komponente
  - aktuelle Entscheidung
  - Zustand: up, down, unknown

  Dabei können die Teilkomponenten der "decision-policy" folgende Zustände aktueller Entscheidungen annehmen:

  - "transfer-policy": transfer - keep_locally - no_decision
  - "location-policy": location=*Rechner_ID* - no_decision
  - "selection-policy": selection=*Prozess_ID*, selection=*Data_ID* - no_decision
  - "status-policy":
    * für Jobs: queue, start, hold, continue, restart, checkpoint, Zeitpunkt der Ausführung
    * für Daten: move, delete, replicate, partition(Anfang, Ende) replicate_and_partition(Anfang,Ende)

- Lastverschiebung:
  - Zustand der Nutzung (up/down)
  - momentan eingesetzter Verschlüsselungs-Mechanismus
  - Events:
    * unsuccessful Process-Migration, unsuccessful Ceckpointing, unsuccessful Process-Start
    * unsuccessful Data-Placement, unsuccessful Replication, unsuccessful Data-Partitioning, unsuccessful Deletion

- System-Warteschlange:
  - Eigenschaften: Maximale Länge der System-Warteschlange für Daten und Jobs
  - System-Warteschlangen-Zustand:
    * Status (up/down)
    * Momentane Anzahl Jobs, die zur Verteilung anstehen
    * Momentane Anzahl Datensätze, die zur Verteilung anstehen
    * Job- und Datensatz-Ankunfts-Frequenz
  - Angaben zur Vergangenheit:

        * Aufzeichnung der Ankunfts-Rate von Jobs und Daten über die Zeit

Benutzer:

- Angaben zu Eigenschaften von Benutzern
  - Benutzer-Name, Benutzer-ID, Benutzer-Typ
  - Priorität der Benutzer bzgl. der Bearbeitung von Jobs gegenüber anderen Benutzern
  - Zugehörigkeit zu Benutzer-Gruppen (vgl. auch administrative Domänen und Sicherheits-Domänen
  - Benutzer-Rechte:
    * Unterscheidung globaler und lokaler Rechte
    * Rechte bzgl. Zugriff auf Rechner, Rechner-Typen, Rechner-Klassen, Rechner-Gruppen
    * Rechte bzgl. Zugriff auf Queues
    * Rechte bzgl. mögliche absetzbare Job-Typen oder Job-Klassen
    * Rechte bzgl. maximaler Anzahl Jobs, die der Benutzer (in bestimmter Zeiteinheit oder generell) absetzen kann
    * Rechte bzgl. maximal verfügbare Ressourcen-Kapazitäten (hard- und soft-limit)
- Weitere Aktivatoren bzgl. Benutzern:
  - create user
  - delete user
- Aufzeichnungen bzgl. Vergangenheit:
  - Abrechnungs-Konten
  - Anzahl Jobs, die von einem Benutzer abgesendet wurden
- Definition von Benutzer-Typen:
  - Manager
  - System-Administrator
  - Workstation-Besitzer
  - Workstation-Benutzer
  - beliebige Kombinationen der Typen

Verteilte Anwendungen

- In verteilten Anwendungs-Objekten enthaltene Objekte:
  - Job-Objekte
  - Datensätz-Objekte

- Profil des Verteilten-Anwendungs-Objektes:
  - Anwendungs-ID, Name, Typ
  - Anwendung gestartet von (Benutzer-ID)
  - Start-Zeitpunkt (Datum, Uhrzeit)
  - statische und dynamische Anwendungs-Beschreibung nach dem Lastverwaltungs-Modell
  - Prioritäten gegenüber anderen Anwendungen
  - maximale Zugriffs- und Kapazitäts-Rechte
- Status der verteilten Anwendung
- mögliche Aktivatoren: start - hold - continue - restart - kill
- mögliche Events:
  - Änderungen des Ausführungs-Status
- Definition von möglichen Zuständen:
  - Status der Anwendung:
    idle, active, error, completed, waiting, unknown
  - Änderungen des Ausführungs-Status:
    start - hold - continue - restart - kill - completed - error

Aufträge - Jobs

- *konstantes Profil*
  - Job-ID, Job-Typ, Job-Klasse
  - Absender (Benutzer), Absendender Rechner, Vater-Prozeß
  - Start-Zeitpunkt (Datum, Uhrzeit)
  - Angaben aus dem Auftrags-Profil (vgl. Lastverwaltungs-Modell)

  variables Profil
  - momentaner Ort der Ausführung
  - Ort des Vaterprozesses
  - Job-Priorität der Bearbeitung
  - mögliche Zustands-Änderungen
  - Zugriffs-Priorität auf Datensätze
  - maximale Zugriffs-Rechte (falls anders als die des Absenders)
- aktueller Job-Status
  - Zustand der Ausführung wie definiert (s.u.)

  - zur Zeit gehaltene Locks
  - Speicherplatz-Bedarf
  - aktuelle Sohn-Prozesse (ID's, Orte)

- Weitere Aktivatoren:
  - vgl. definierte, mögliche Änderungen des Job-Zustands

- Aufzeichnungen über die Vergangenheit des Job-Ablaufs:
  - bisherige Daten-Zugriffe (Anzahl, Art, Daten-ID)
  - bisher erzeugte Daten (Anzahl, Daten-ID)
  - bisherige Laufzeit
  - bisherige Kommunikation mit anderen Prozessen (Anzahl, Prozess-ID)
  - bisherige Unteraufrufe
  - frühere Rechner-Orte der Prozess-Ausführung (von denen aus der Job gecheckpointet und weiter migriert wurde)

- mögliche Job-Events
  vgl. Zustands-Ausführungs-Änderungen

- Definition von Job-Status und Ausführungs-Status-Änderungen:
  - mögliche Ausführungs-Zustände von Jobs:
    queued, running, migrating, resting, completed, unknown
  - mögliche Zustands-Änderungen:
    queue, start, hold, continue, restart, checkpoint, kill, completed, error

- Definition von Job-Typen:
  - redirective
  - interactive
  - batch
  - checkpointing
  - parallel
  - Kombinationen dieser Typen

Daten-Objekte

- *konstantes Profil*
  - Daten-ID, Daten-Typ
  - Entstehungs-Zeitpunkt (Datum, Uhrzeit)

  variables Profil
  - momentaner Ort der Lagerung, Rechner-Id, Platte oder Speicher
  - Speicherplatz-Bedarf

  - Zugriffs-Möglichkeiten (ro, rw, wo), Zugriffs-Pfad
  - falls Original: ID und Orte von Kopien und kopierten Partitionen
  - falls Replikation: ID und Orte des Originals
  - falls Partition: ID und Orte übriger Partitionen

- aktueller Daten-Status
  - Zustand der Ausführung wie definiert (s.u.)

- Weitere Aktivatoren:
  - vgl. definierte, mögliche Änderungen des Job-Zustands

- Aufzeichnungen über die Vergangenheit des Job-Ablaufs:
  - bisherige Zugriffs-Häufigkeiten und Zugriffs-Arten
  - bisherige Verweildauer auf diesem Rechner
  - frühere Orte incl. dortige Vergangenheit der Datenobjekte

- mögliche Daten-Events
  vgl. Zustands-Ausführungs-Änderungen

- Definition von Daten-Status und Ausführungs-Status-Änderungen:
  - mögliche Ausführungs-Zustände von Daten:
    resting, locked, accessible, migrating, unknown
  - mögliche Zustands-Änderungen:
    move, lock, unlock, delete

- Definition von Daten-Typen:
  - Original
  - Replikation
  - Partition
  - Kombinationen dieser Typen

6.3 Entwicklung eines Organisations-Modells

Zentrales Konzept des Organisations-Modells für das Management von Lastbalancierung und Datenmigration ist die Bildung und Berücksichtigung verschiedener Domänen.
Nach [Slom94] stellt die Domänenbildung ein Rahmenwerk zur Aufteilung von Management-Verantwortlichkeiten dar, indem Management-Objekte zum Vorteil der Manager zu Gruppen zusammengefaßt werden. [Slom94] definiert dabei eine Domäne als ein Objekt, das eine Ansammlung von Objekten repräsentiert, die explizit zu einer Gruppe zusammengefaßt wurden, um eine gemeinsame Management-Policy (Management-Verfahrensklasse) anzuwenden.
Wesentliche Domänen für das Management der Lastverwaltung stellt zum einen die Lastverwaltungs-Domäne (Workstation-Pool eines Lastverwaltungs-Systems), zum anderen die Lastverwaltungsbereichs-Domäne dar. Dabei umfaßt die Lastverwaltungs-Domäne

alle jene Objekte des verteilten Rechner-Netzes, die an der Lastbalancierung und Datenmigration *eines* Lastverwaltungssystems tatsächlich beteiligt sind, während die Lastverwaltungsbereichs-Domäne alle jene Objekte beinhaltet, die mögliche oder tatsächliche Kandidaten für den Lastverwaltungs-Vorgang sind. So sind beispielsweise alle Rechner eines Workstation-Clusters, die mögliche Kandidaten für den Einsatz im Lastverwaltungssystem Elemente der Lastverwaltungsbereichs-Domäne, und alle Rechner, die momentan am Lastverwaltungsvorgang beteiligt sind, Elemente der Lastverwaltungs-Domäne. Es ist leicht einzusehen, daß die Lastverwaltungs-Domäne eine echte Teilmenge der Lastverwaltungsbereichs-Domäne darstellt. Weiterhin kann leicht eingesehen werden, daß verschiedene Lastverwaltungs-Systeme, die miteinander kooperieren, in verschiedenen Lastverwaltungs-Domänen operieren, die auch von verschiedenen Managern verwaltet werden können, sowie, daß die Schnittmenge der Domänen die Kooperations-Punkte beider Lastbalancierungs-Systeme darstellen. Rechner dieser Kooperations-Punkte müssen von den Managern verschiedener Domänen geeignet verwaltet werden, was u.U. eine Kooperation der Manager erforderlich macht.
Die Lastverwaltungs-Domäne und Lastverwaltungsbereichs-Domäne kann hinsichtlich der Maschinen-Typen weiter unterteilt werden in eine Execution-Domäne sowie eine Submit-Domäne, wobei die Schnittmenge beider Bereiche diejenige Gruppe von Rechnern darstellt, von denen aus sowohl Jobs abgesetzt werden können, als auch die entfernte Abarbeitung von Jobs unterstützt wird.
Daneben müssen innerhalb des Managements von Lastbalancierung und Datenmigration bereits vorhandene Domänen berücksichtigt werden. Hier sind vor allem Abrechnungs-Domänen und Sicherheits-Domänen von besonderer Bedeutung.
Zu beachten ist ferner, daß die Lastverwaltung als Teilbereich des Ressourcen-Managements (vgl. z.B. [Gosc91]) auch bestimmte Objekte zu Gruppen zusammenfaßt und entsprechend verwaltet. So können z.B. in heterogener Umgebung Jobs an eine Gruppe von Rechnern gesendet werden, die alle von einer bestimmten Architektur sind. Derartige Domänen betreffen allerdings die Lastverwaltung an sich und sind von Domänen des Managements der Lastverwaltung zu trennen.

6.4 Konzeption des Funktions-Modells für das Mangement

Das Funktionsmodell stellt neben dem Informationsmodell einen zentralen Aspekt des Management-Konzepts dar. "Die funktionale Dimension betrifft die Zuordnung von Management-Aufgaben zu Funktionsbereichen." ([Hege93] S88) Die in diesem Konzept berücksichtigten Bereiche sind die im Management-Framework der ISO angegebenen Bereiche Konfiguration, Fehler, Leistung, Abrechnung und Sicherheit. Dabei sollen bereichsspezifisch Funktionen, Dienste und Management-Objekte, die in diesem Bereich von Interesse sind, festgelegt werden. Zu berücksichtigen sind insbesondere auch Wechselwirkungen zwischen verschiedenen Funktionsbereichen ([Hege93] S126ff).

6.4.1 Konfigurations-Management

Das ordnungsbemäße Funktionieren eines Lastverwaltungs-Systems im verteilten Rechensystem hängt wesentlich von einer geeigneten Konfiguration der Ressourcen, der

Lastverwaltungs-Software und der Benutzer, die auf diesem System operieren, ab.
Als Beispiele für wesentliche Aspekte des Konfigurations-Managements können folgende Möglichkeiten andeutungsweise skizziert werden:

- Welche Objekte müssen konfiguriert werden?
  Hier können folgende Punkte voneinander abgegrenzt werden:

  1. Konfiguration der Maschinen und Ressourcen
  2. Konfiguration der E/A-Server
  3. Konfiguration der physischen und logischen Komponenten der Lastverwaltung selbst
  4. Konfiguration der Benutzer
  5. Konfiguration der Schnittstellen des Lastbalancierungs-Systems zu übriger Software des verteilten Ablauf-Systems
  6. Konfiguration des Netzes: Netz-Management

- Wann muß Konfiguration möglich sein?
  Die Ressourcen und Objekte der Lastverwaltung sind ständigen Änderungen in Verfügbarkeit und Zustand unterworfen. Konfiguration darf nicht ein einmaliger Vorgang bei der Installation des Lastverwaltungs-Systems sein, sondern Umkonfigurationen müssen auch dann möglich sein, wenn die Lastverwaltung z.B. mit der Bearbeitung einer verteilten Anwendung beschäftigt ist. Umkonfigurationen sind beispielsweise wichtig im Fehlerfall oder zur Leistungs-Steigerung (vgl. nächster Punkt). Dabei soll weiterhin gefordert werden, daß Konfigurationen möglichst keine Unterbrechung des laufenden Betriebes der Lastverwaltung notwendig machen. [Slom94] stellt fest, daß Änderungen so durchgeführt werden müssen, daß das resultierende, veränderte System in einem konsistenten Zustand weiterläuft. Da es unwahrscheinlich ist, daß diese Konsistenz alleine durch das Hinzufügen von Management-Funktionalität erbracht werden kann, müssen geeignete Möglichkeiten fest in das Lastverwaltungs-System eingebaut werden. ([Slom94] S493ff)

- Aus welchen Gründen müssen (Um-)Konfigurationen vorgenommen werden?
  Außer der Konfiguration bei der Installation eines Lastverwaltungs-Systems können Umkonfigurationen z.B. aufgrund ausgefallener Rechner (Fehler-Management) oder zu Zwecken der Leistungs-Verbesserung (Performance-Management) notwendig werden (vgl. Kapitel "Fehler-Management", "Leistungs-Management")

- Welche Aktionen betreffen das Konfigurations-Management?
  Wesentliche Aktionen sind:

  – Einstellen der notwendigen Konfigurations-Parameter der vorhandenen Ressourcen und Objekte
  – Initialisierung oder Ungültigmachung von Konfigurationen
  – Überwachung der vorhandenen Objekte und ihrer Konfiguration
  – Überwachung von Ereignissen, die eine Umkonfiguration notwendig machen (z.B. Kapazitätsgrenzen; vgl. auch vorigen Punkt)

- Von welchen Akteuren sollen welche Aktionen des Konfigurations-Managements ausgeführt werden dürfen?
  Wesentliche Aspekte sind:

  1. Manager müssen möglichst umfassend operieren dürfen, müssen aber lokale Bedürfnisse von Besitzern von Rechnern berücksichtigen
  2. Besitzer von Rechnern müssen gemäß einer Management-Policy möglichst weitgehend ihren Rechner für die Lastverwaltung konfigurieren können, oder sollten zumindest Konfigurations-Entscheidungen mit beeinflussen können, haben aber darüberhinaus keinen Zugang zu Konfigurations-Aktionen auf anderen Ressourcen
  3. Benutzer des Lastverwaltungs-Systems haben grundsätzlich keine Konfigurations-Möglichkeiten.

- Auf welche Art und Weise muß die Konfiguration (aus der Ferne) durchgeführt werden?
  Konfigurationen von Ressourcen dürfen die Grenzen der Zuständigkeit bezüglich der Lastverwaltung nicht überschreiten, d.h. daß Konfiguration aus der Ferne nicht lokale Rechte verletzen darf, bzw. daß Konfigurationen ausführenden Einheiten nur innerhalb der Grenzen der Lastverwaltung Aktionen ausführen dürfen, sofern nicht zusätzliche Zuständigkeiten den Einheiten weitere Eingriffsmöglichkeiten gewähren (vgl. Sicherheits-Managment)

- Welchen Wirkungsumfang sollen Konfigurationen haben?
  Gewisse Konfigurationen könne global für die gesamte Domäne gelten, während andere Konfigurationen nur lokale Bedeutung für eine Ressourcen bzw. ein Objekt haben.

Im folgenden soll nun näher auf notwendige Konfigurations-Möglichkeiten der Objekte eingegangen werden:

- Rechner und deren Ressourcen:
  - Konfiguration der Lastverwaltungs-Software auf dieser Maschine
  - Konfiguration der maximalen Kapazitäten (hardlimit und softlimit) für die Lastverwaltung
  - Konfiguration von Benutzer-Zugriffs-Beschränkungen
  - Konfiguration von Beschränkungen bzgl. der Akzeptanz von Jobs
  - Konfiguration von Beschränkungen bzgl. der Akzeptanz von Datensätzen

- Ein/Ausgabe-Server:
  - Konfiguration der Lastverwaltungs-Software auf dem Server
  - Konfiguration der maximalen Kapazitäten (hard- und softlimit) für die Lastverwaltung
  - Konfiguration von Benutzer-Zugriffs-Beschränkungen

  - je nach Server-Typ Konfiguration der Beschränkungen bzgl. der Akzeptanz bestimmter Jobs oder Datensätze

- Physische Komponenten der Lastverwaltung:

  1. Dämonen des Lastverwaltungs-Systems
     - Konfiguration des Zustandes des Dämons (up/down/not avail.)
     - Konfiguration des Dämons für die vorhandene Architektur
     - Initialisierung der Dämonen

     (Die von dem Dämon repräsentierte Funktionalität wird über die logischen Komponenten der Lastverwaltung konfiguriert)
  2. Queues des Lastverwaltungs-Systems
     - Konfiguration der Länge der Queue

     (bei mehreren Queues können weitere Konfigurationen ähnlich der Rechner-Konfiguration durchgeführt werden, um so verschiedenartige Queues zu erhalten)
  3. Schnittstellen zu Software des verteilten Ablaufsystems
     - Anpassung an die Erfordernisse des angeschlossenen Systems gemäß den Spezifikationen

- Logische Komponenten des Lastverwaltungs-Systems:

  1. Lasterfassungskomponente
     - Konfiguration der von der Lasterfassungskomponente beobachteten Größen, die zugleich die Grundlage der Entscheidungs-Findung der Lastbewertungskomponente bildet
  2. Lastbewertungskomponente
     - Konfiguration der Teil-Komponenten "negotiation-policy" und "decision-policy" (sowie deren Teilkomponenten) anhand Lastbalancierungs-spezifischer Parameter
     - Auswahl austauschbarer Lastbewertungs-Komponenten soweit möglich und von der Lastverwaltung unterstützt

- Benutzer und Verwalter des Lastverwaltungs-Systems:

  - Festlegung von Name, Typ, Gruppen-Zugehörigkeiten
  - Konfiguration von Beschränkungen des Zugriffs auf best. Rechner oder Architekturen
  - Konfiguration von Beschränkungen des Zugriffs zu bestimmten Zeiten
  - Konfiguration der Beschränkung der Ausführung von Jobs
  - Konfiguration von Beschränkungen der Verwendung von Ressourcen-Kapazitäten
  - Konfiguration von Prioritäten gegenüber anderen Benutzern

  Abgleich mit schon vorhandenen Daten von Benutzern notwendig

6.4.2 Fehler-Management

Ziel des Fehler-Managements von Lastbalancierung und Datenmigration ist die Verfügbarkeit der durch die Lastbalancierung und Datenmigration erbrachten Leistung möglichst hoch zu halten.
Die Bedeutung des Fehler-Managements für die Lastverwaltung kann insbesondere deutlich gemacht werden, wenn z.B. langlaufende Anwendungen betrachtet werden, deren Ergebnis für ein Unternehmen von Bedeutung für Wettbewerbsvorteile ist.

Es soll für das Fehler-Management ein Konzept der möglichst unterbrechungsfreien und verlustfreien (im Sinne des für den Benutzer richtigen Ergebnisses des Auftrags) Fehler-Behandlung und -Behebung entworfen werden.

Sammlung von möglichen Fehlern in Lastverwaltungs-Systemen

- Absturz des Master-Dämons ("Single-Point-Failure")
- Absturz eines Dämons der Lastverwaltung
- Ausfall eines Rechner-Knotens ohne Verlust der daruaf laufenden Prozesse und Daten
- Ausfall eines Rechner-Knotens: Verlust der darauf laufenden Prozesse und Daten, mit Restart
- Ausfall eines Rechner-Knotens ohne Restart
- zeitweiser oder dauerhafter Zusammenbruch der Kommunikation zwischen zwei Rechnerknoten
- Absturz entfernt laufender Prozesse
- Falsche Placierung von Prozessen oder Daten (falsche Architekturen oder ungenügende Kapazitäten)
- Nichtvorhandensein von Dämonen auf Rechnern, die von Prozessen kontaktiert werden
- Zu häufiges automatisches Checkpointen von Prozessen
- Prozeß, der Architektur verlangt, die zur Zeit nicht verfügbar ist, wird vergessen
- Nichtbeachtung der Nicht-Verfügbarkeit von Benutzer-beanspruchten Rechnern
- Prozeß, der Locks auf Datensätzen hält, gibt Locks nicht wieder frei (z.B. weil er abgestürzt ist)
- Konsistenz der Datenhaltung ist gestört: Kopie eines Datensatzes stimmt mit dem Original nicht überein

Sammlung von Fehler-Ursachen

- Lastverwaltungs-System arbeitet nicht korrekt, weil es über den Grenzen der Leistungs-Fähigkeit arbeitet (z.B. Erweiterbarkeit des Rechner-Pools ist begrenzt, zu hohe Änderungsraten in Maschinen-Verfügbarkeiten oder Kommunikation, zu hohe Last-Situationen auf dem gesamten Cluster)
- Andere Objekte wie Queues oder lokale Ressourcen werden überbeansprucht
- Konfiguration der Ressourcen ist nicht korrekt
- Job-Submission-Files sind nicht korrekt
- Kommunikations-Verbindungen sind unterbrochen
- Andere Software-Einheiten belegen Dienste oder Ressourcen: Verletzung der Lastbalancierungs-"Grenzen"
- unterliegende Services von z.B. Prozeß-Migration oder Parallel-Job-Support sind fehlerhaft
- Teile des Systems oder Lastverwaltungs-Systems sind ausgefallen
- physische Einwirkungen führen zu Fehlern

Konzeption des Fehler-Managements

- Entwicklung eines möglichst umfassenden Fehler-Toleranz-Konzepts
  - Vermeidung von Single-Point-Failure-Problems: Abhängigkeiten von einzelnen Master-Schedulern vermeiden: z.B. optionale, passive Master-Scheduler im Hintergrund
  - Weitere Verwendung des Group-Masking-Konzepts: Fehler-Toleranz durch parallele Bearbeitung von Aufträgen auf mehreren Rechnern (vgl [CDK94] S466)
  - Rerouting von Jobs, die auf fehlerhafte Maschine gescheduled werden sollten
  - Checkpointing und Migration von Prozessen bzw. konsistente Verschiebung von Datensätzen von fehlerhaften Maschinen (soweit möglich) auf andere Rechner
  - Neustart von Prozessen auf anderen Rechnern, die nicht gecheckpointet werden können
  - Einführung von Sicherheits-Prioritäten für Jobs, so daß wesentliche/wichtige Jobs mit höherer Fehler-Toleranz bearbeitet werden können (z.B. für Group-Masking-Konzept)
  - Durchführung von Selbst-Tests der Fehler-Toleranz-Mechanismen zur Überprüfung der korrekten Arbeitsweise
- Geeignete Mitprotokollierung relevanter Daten: Präventiv-Maßnahmen
  - Protokollierung von Daten (-Inhalten) und Jobs (incl. Ergebnissen) mit Zeitstempel: Erstellung eines Recovery-Files
  - Einführung eines Fehler-Dokumentations-Systems (Trouble-Ticket-System)

  - Einführung eines Dokumentations-Systems für Konfigurations- Änderungen (nach [Hege93] führen Umkonfigurationen und Hinzufügen von Komponenten leicht zum Auftreten von Fehlern)

- Geeignete Überwachungs-Funktionalität und Fehler-Frühwarn-System
  - allg. Konfiguration geeigneter Alarme und Error-Events
  - Ermittlung und Einstellung von Lastverwaltungs-Leistungs-Grenzen; Konfiguration als Events
  - Beobachtung von Umkonfigurationen und Neukonfigurationen (vgl. Protokollierung von Konfigurations-Änderungen)
  - Überwachung "lebenswichtiger" Funktionen des Systems;
  - Überwachung der Orte und Zustände von Jobs und Daten im System (vgl. System-Beobachtung der Lasterfassungskomponente des Lastverwaltungs-Modells)
  - Überwachung von Konfigurationen von Ressourcen

- Auf Frühwarnsystem aufbauendes Fehler-Vermeidungs-Konzept
  - Geeignete Funktionalität der Reaktion auf Alarme, abgestuft nach Dringlichkeit
  - Retten momentaner Zustände eines gerade abstürzenden Rechneres soweit möglich
  - Einleiten von Maßnahmen beim Erreichen von Leistungs-Grenzen
  - Berücksichtigung von Fehler-Fortpflanzungen

- Möglichst verlustfreies Fehler-Behebungs-Konzept
  - Versuch der Recoverys abgestürzter Dämonen / Master-Dämonen
  - Zuschalten neuer Rechner in den Pool bei Ausfall anderer Rechner
  - Ursachen-Feststellung von Fehler-Ereignissen
  - Recovery verlorener Jobs und Daten mit Hilfe des recovery-Files
  - Sicherstellung der Konsistenz, insbesondere beim Scheduling von verteilten Anwendungen

6.4.3 Leistungs-Management

Das Ziel des Leistungs-Managements ist allgemein, daß das zu managende System, hier also das Lastverwaltungs-System, nicht nur (fehlerfrei) funktionieren soll, sondern darüberhinaus auch "gut", d.h. "gut" bzgl. einer bestimmten Dienst-Güte bzw. optimal unter den gegebenen Umständen, laufen soll ([Hege93]).
Das Konzept des Leistungs-Managements von Lastbalancierung und Datenmigration soll daher u.a. Definitionen zu Leistungs-Aspekten, Bestimmungen zu Überwachungs-Parametern und Eingriffsmöglichkeiten sowie Hinweise zu weiteren Möglichkeiten der Leistungs-Verbesserung umfassen.

1. Definitionen zu Leistungs-Aspekten:
   Wesentliches Kriterium des Leistungs-Managements sind Definitionen darüber, wann ein System "gut" funktioniert. Wesentliche Kriterien, die eine Grundlage für die Messung der Güte des Services der Lastverwaltung schaffen können, sind die in Kapitel 3.1.1 angegebenen möglichen Zielfunktionen.
   Definitionen von Leistungs-Grenzen, wie beim Fehler-Management angegeben, haben auch Bedeutung für das Leistungs-Management, da Eingriffe zur Leistungs-Verbesserung u.a. gerade auch an Leistungs-Grenzen erforderlich sind.
   Wesentlich ist darüberhinaus die Entwicklung von Erfahrung bzgl. Wirkungs-Zusammenhängen bei Leistungs-Verbesserungen, die u.U. durchaus komplex und interdependent sein können.

2. Bestimmung geeigneter Last-Indices für die Messung der Maschinen-Auslastung:
   Eine wesentliche Größe der Leistungsmessung stellt der "Load-Index" dar, also die Größe, die geeignt die Auslastung eines Rechners repräsentiert. Beispiele sind:

   - CPU-Verwendung
   - Länge der Prozessor-Warteschlange ("ready queue") ("load average" in UNIX-Terminologie)
   - sog. "Stretch-Faktor", d.h. das Verhältnis zwischen Ausführungszeit eines Prozesses auf einer beladenen Maschine und seiner Ausführungszeit auf der selben Maschine, wenn diese sonst keine Prozesse ablaufen lassen würde

   Load Leveler und Condor z.B. verwenden den "Berkeley One Minute Load Average", der die durchschnittliche Anzahl von Prozessen auf der "ready to run"-Warteschlange des Betriebs-Systems angibt.
   [FeZh86] gibt einen Last-Index für die Messung von Last auf einem Rechnerknoten an, unter der Annahme, daß das Ziel der Lastbalancierungs-Strategie die Minimierung der Antwortzeit des Benutzer-Kommandos ist, das für eine entfernte Ausführung in Frage kommt. (vgl. [FeZh86])

3. Bestimmung von Überwachungs-Parameter für die Leistungs-Messung:
   Folgende Beispiele von Leistungs-Parameter sollen für die zu managenden Objekte angegeben werden:

   - "wait-ratio"$=\frac{time\ job\ waits\ for\ service}{job's\ service\ time}$
     d.h. die "wait-ratio" (nach [LiLi88]) gibt den Quality of Service an, den Benutzer für die entfernte Ausführung von Jobs erhalten
   - "leverage"$=\frac{amount\ of\ remote\ capacity\ consumed\ to\ execute\ Job}{amount\ of\ local\ capacity\ consumed\ to\ support\ remote\ execution}$
     d.h. die "leverage" (nach [LiLi88]) spiegelt die Kosten wieder, die von dem lokalen Rechnerknoten (von dem aus der Job abgesendet wurde) zur Unterstützung des entfernt ausgeführten Jobs zur Unterstützung aufgewendet werden müssen.

   - Rechner:
     - durchschnittliche "wait-ratio" nach [LiLi88]
     - "leverage" nach [LiLi88]
     - einer der Last-Indices zur Messung der Maschinen-Belastung

    - momentan verfügbarer Speicher-Platz von Speicher und Platte
- Kommunikations-Verbindung:
    - Kapazitäts-Auslastung der Verbindung
- Jobs und Daten:
    - vgl. Lastverwaltungs-Modell: Lasterfassung
    - Zeitmessungen für alle Phasen der Bearbeitung (vgl. z.B. [Beck93]): Rechenzeit, Wartezeit aufgrund von Synchronisation, Plattenzugriffs-Verzögerungen, Unteraufrufen, Übertragungs-Zeit bei Prozeß-Migration und Checkpointing, Wartezeit in der Queue
- Benutzer:
    - Durchschnittliche Anzahl von Jobs, die von einem Benutzer an die Lastverwaltung abgesetzt wurden (nach [LiLi88])
- Lastverwaltungs-Komponenten:
    - Länge der generischen Eingangswarteschlange der Komponente
    - Ankunftsrate der Prozesse
    - aktuelle und durchschnittliche lokale Queue-Länge, "wait-ratio" für alle Prozesse sowie durchschnittliche "wait-ratio"
    - Info-Update-Frequenz der Lasterfassungs-Komponente und Anzahl an beobachteten Größen
    - Antwort-Zeit der Rechner-Auswahl der Lastbewertungs-Komponente
    - Anzahl an Prozeß- bzw. Daten-Verschiebungen pro Zeiteinheit
    - Kommunikations-Umfang von Dämonen
- Gesamt-Meß-Größen:
    - durchschnittliche gesamt Job-Antwort-Zeit bzw. je nach Zielfunktion geeignete Größe

4. Eingriffs-Möglichkeiten: Leistungsparameter und deren Wirkungsweise:
    - Algorithmus: Parameter und Strategien-Wechsel:
    - Parameter der Lasterfassungs-Komponente:
    - Parameter der Konfiguration des unterliegenden Systems:

5. Hinweise zu weiteren Möglichkeiten der Leistungs-Verbesserung:
   Vorausplanung der Verteilung von Jobs im Falle verteilter Anwendungen, Einführung speziell abgerechneter Leistungs-Prioritäten von Benutzern, Jobs und Daten sowie Berücksichtigung von Informationen über vermutliches Verhalten im Algorithmus aus den Anwendungs- und Auftrags-Profilen (vgl. Lasterfassungs-Komponente des Lastverwaltungs-Modells)

6.4.4 Abrechnungs-Management

Die Lastbalancierung stellt durch die Verteilung von System-Last Benutzern einen Dienst zur Verfügung, der allgemein mit der Verfügbarkeit zusätzlicher Ressourcen-Kapazitäten umschrieben werden kann. Diese Dienste führen allerdings auch zu Kosten, die auf die Kostenverursacher verteilt werden müssen. Die Aufteilung der Kosten an sich ist Aufgabe der Abrechnungs-Politik; innerhalb des Abrechnungs-Managements sollen lediglich die Mechanismen zur Verfügung gestellt werden, die die Kosten-Abrechnung nach einer geeigneten Abrechnungs-Politik erlauben. Zu den administrativen Funktionen kann nach [Gosc91] gezählt werden: Die Berechnung von Dienst-Benutzungen, die Generierung von entsprechenden Kosten, die Erstellung von Rechnungen, die Abwicklung der Bezahlung, die Einsammlung von Schulden, die Abwicklung von Vertägen, die Unterstützung des Rechnungs-Wesens sowie einige andere Funktionen ([Gosc91] S384).
Im Hinblick auf das Management von Lastverwaltung und Datenmigration soll hier allerdings lediglich die Bereitstellung von Möglichkeiten zur Abrechnung von Dienst-Benutzungen, und hier speziell zur Abrechnung des Lastverwaltungs-Dienstes eingegangen werden. Accounting betrifft Dienste, Ressourcen und Dienst-Benutzer. Der Verbrauch von Ressourcen durch die Bearbeitung von Jobs oder Anwendung mittels der Lastverwaltung kann mittels der Aufzeichnung relevanter Management-Information leicht durchgeführt werden (vgl. hierzu das Informations-Modell). Accounting betrifft hier insbesondere die Länge des Zeitintervalls sowie die Intensität der Inanspruchnahme der Ressourcen durch einen Job. Wesentliche Objekte, die hiervon betroffen sind, sind CPU-Rechenleistung, Speicher- sowie Platten-Platz und Ein/Ausgabe-Geräte. Die Abrechnung erfolgt durch Aufsummierung der Ressourcen auf allen Rechneren, die von dem Job (evtl. durch Checkpointing und Migration) beansprucht wurden. Ähnliches gilt für die Beanspruchung von Ressourcen durch die Haltung von Datensätzen.
Liegen die beanspruchten Ressourcen außerhalb einer Abrechnungs-Domäne, erfolgt die Abrechnung zwischen dem Dienst-Nehmer und dem Besitzer der beanspruchten Ressourcen-Kapazitäten über die entsprechenden Zuständigkeits-Stellen der Abrechnungs-Domänen.
Tatsächlich ergibt sich allerdings eine Schwierigkeit der Berechnung noch hinsichtlich der Tatsache, daß Benutzer des Lastverwaltungs-Dienstes aufgrund der Transparenz des Ortes der Prozeß-Ausführung daran interessiert sind, für gleiche Aufträge gleich bezahlt zu werden. Werden nun Prozesse über Abrechnungs-Domänen-Grenzen hinweg gescheduled, so kann bei unterschiedlichen Abrechnungs-Politiken für gleiche Dienste unterschiedliche Kosten berechnet werden. Ähnliches gilt für die Bearbeitung von Prozessen in heterogenen Netzen, in denen es ebenfalls zu Abweichungen kommen kann. Es wäre also generell mit der transparenten Verteilung von Prozessen und Daten eine geeignete transparente Abrechnung für die Dienst-Erbringer zu fordern. Dabei würden die Kosten für leistungsfähigere Dienst-Erbringer etwas über den tatsächlichen Kosten berechnet werden und die Kosten für weniger leistungsfähige Ressourcen-Anbieter etwas unter den realen Kosten, so daß ein systemweiter Ausgleich garantiert ist.

6.4.5 Sicherheits-Management

Das Sicherheits-Management befaßt sich allgemein mit dem Austausch sensibler Informationen in Rechner-Netzen und verteilten Systemen ([Hege93] S92). [CePi93] stellt fest, daß Maschinen, die an der Lastbalancierung teilnehmen, oft von verschiedenen Gruppen verwaltet werden, die sicherstellen wollen, daß Information nicht beliebig von Gastbenutzern abgerufen werden kann, die dazu nicht berechtigt sind. ([CePi93] S294) Es lassen sich grundsätzlich drei Ansatz-Punkte konkretisieren, an denen Sicherheits-Mechanismen und Verfahren wesentlich erscheinen:

- Die wesentlichste Sicherheitsanforderung erscheint an der Stelle des verteilten Systems zu liegen, an der die Grenze zwischen dem von der Lastbalancierung genutzten System und dem von anderen, z.B. lokal genutzten Systemen liegt. Speicherbereiche z.B., auf die sowohl von gescheduleten Prozessen als auch lokal zugegriffen werden kann, können mögliche Ansatzpunkte bilden für Sicherheits-Angriffe.
- Aus diesem Grund kann ein weiterer Ansatz-Punkt festgehalten werden, an dem Sicherheits-Mechanismen von Interesse sind: die Grenze von lokalen Workstation nach außen zum Netz hin. Bestimmte Authorisierungs- und Authentifizierungs-Verfahren, die für Jobs vor der Entscheidungs-Findung durchgeführt werden, helfen, die zu schützenden Rechner vor ungewünschtem Eindringen zu schützen
- Als dritten Punkt des Schutzes in Lastverwaltungs-Systemen ist der Kommunikations-Kanal zu nennen, über den die Verteilung von Jobs und Daten erfolgt. Geeignete Verschlüsselungs-Mechanismen die innerhalb der Lastverschiebe-Komponente vorgenommen werden, können Daten auch auf ihrem Weg zum Zielknoten vor unberechtigtem Zugriff schützen. Die Ansprechbarkeit solcher Sicherheitsverfahren kann über geeignete Parameter bei der Absendung der Jobs bzw. Daten eingestellt werden.

Daneben ist natürlich ein differenziertes Zugriffs-System für verschiedene Benutzer, Benutzertypen bzw. Benutzer-Gruppen vorzusehen, das u.a. Rechte der Job-Ausführung, Rechner-Auswahl, Informations-Beschaffung innerhalb der Lastverwaltung oder Management-Funktionalitäts-Verfügung regelt. Geeignete Sicherheits-Logging-Funktionen zur Protokollierung von Absendern von Jobs, Zielrechnern, Daten-Zugriffe mit Zeitstempel können die Grundlage für eine Überwachungs-Funktionalität für die Sicherheit des Lastverwaltungs-Mechanismus bilden.

6.5 Anmerkungen zur Bedeutung einer Schnittstelle zwischen Management und Lastverwaltung

Bei der Konzeption des Managements von Lastverwaltungs-Systemen sind auch Überlegungen zu einer expliziten Schnittstelle, die über die durch Management-Konzepte allgemein gängigen Mechanismen hinausgehen, von Bedeutung. Viele der von der Lastverwaltung benötigten Informationen sind auch für das Management allgemein im Sinne des Systems-Management von Bedeutung wie auch für das Lastverwaltungs-Management im Besonderen. Die Bereitstellung der Information von einer Komponente an die andere zu Zwecken der Vermeidung von Redundanz erschein aber dennoch nicht angebracht. Zum einen wäre eine Informations-Bereitstellung durch die Lastverwaltung nich sinnvoll, da die Management-Komponente als übergeordnete Funktion auch dann noch funtionieren muß, wenn die Lastverwaltung bereits fehlerhaft ist oder ganz ausgefallen. Andererseits ist auch die Bereitstellung von Information vom Management an die Lastverwaltung problematisch, da Informationen z.B. in dezentralen Verfahren weitaus effizienter durch lastbalancierungs-eigene Strategien bezogen werden kann, insbesondere aufgrund des hochdynamischen Informations-Flusses.

Gewisse Teile der Informations-Erfassung, wie z.B. weitgehend unveränderliche Konfigurationen (wie Charakteristika von Rechnern und Ressourcen) können aber durchaus bereitgestellt werden. Darüberhinaus gibt es verschiedene Informationen des Managements wie z.B. des Konfigurations-Managements (Prioritäten von Benutzern o.ä.) oder des Sicherheits-Managements, die wesentlich an der Entscheidungs-Findung beteiligt sind und daher notwendigerweise der Lastverwaltung über eine Schnittstelle angeboten werden müssen.
Bezüglich der Konfiguration zur Laufzeit ist das Management auf die Unterstützung durch die Lastverwaltung angewiesen ([Slom94]). Hier ist eine geegnete Konfiguration der Schnittstelle zum Management hin erforderlich, die eine Kommunikation im Falle von Umkonfigurationen ermöglicht.

Literaturverzeichnis

[Beck92] W. Becker. Lastbalancierung in heterogenen Client-Server Architekturen. Fakultätsbericht 1992/1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1992.

[Beck93] W. Becker. Globale dynamische Lastbalancierung in datenintensiven Anwendungen. Fakultätsbericht 1993/1, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1993.

[Beck94] W. Becker. Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen in Workstation-Netzen. Fakultätsbericht 1994/4, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1994.

[BrFi81] R.M. Bryant and R.A. Finkel. A stable distributed scheduling algorithm. In *Proceedings of the 2nd International Conference on Distributed Computing Systems*, 1981.

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems, Concepts and Design*. Addison Wesley, 1994.

[CePi93] D. Cerutti and D. Pierson. *Distributed Computing Environments*. Mc. Graw Hill, 1993.

[EfGr88] K. Efe and B. Groselj. Minimizing control overheads in adaptive load sharing. In *ACM SIGMETRICS, Performance Evaluation Review*, May 1988.

[ELZ86a] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. In *IEEE Transactions on Software Engineering*, vol. 12, no. 5, May 1986.

[ELZ86b] D. Eager, E. Lazowska, and J. Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. In *Performance Evaluation*, vol. 6, March 1986.

[Ezza86] A. Ezzat. Load balancing in nest: A network of workstations. In *Proceedings Fall Joint Computer Conference*, Dallas, Texas, 1986.

[FeZh86] D. Ferrari and S. Zhou. A load index for dynamic load balancing. In *Proceedings Fall Joint Computer Conference*, Dallas, Texas, 1986.

[GEN94a] GENIAS Software GmbH, Erzgebirgstr. 2, 93073 Neutraubling. *Codine Reference Manual*, version 3.3 edition, Juli 1994.

[GEN94b] GENIAS Software GmbH, Erzgebirgstr. 2, 93073 Neutraubling. *Codine User's Guide*, version 3.3 edition, Juli 1994.

[Gosc91] A. Goscinski. *Distributed Operating Systems - The Logical Design.* Addison Wesley, 1991.

[GrSn93] T.P. Green and J.S. Snyder. SQS, a scalable queuing system. Technical report, Supercomputer Computations Research Institute, Florida State University, 1993.

[Hege93] H.-G. Hegering. *Integriertes Netz- und Systemmanagement.* Addison Wesley, 1993.

[Hein90] E. Heinen. *Industriebetriebslehre.* Gabler Verlag, 1990.

[HLM+90] S. Hosseini, B. Litow, M. Malkawi, J. Mc Pherson, and K. Vairvan. Analysis of a graph coloring based distributed load balancing algorithm. In *Journal of Parallel and Distributed Computing*, vol. 10, no. 6, October 1990.

[IBM93a] IBM International Business Machines Corporation, Kingston, NY. *IBM LoadLeveler Administration and Installation*, March 1993. form number SH26-7220-00.

[IBM93b] IBM International Business Machines Corporation, Kingston, NY. *IBM LoadLeveler User's Guide*, September 1993. form number SH26-7226-01.

[Kale88] L. Kale. Comparing the performance of two dynamic load distribution methods. In *Proceedings Parallel Processing*, 1988.

[KaNe93a] J.A. Kaplan and M.L. Nelson. A Comparison of Queueing, Cluster and Distributed Computing Systems. NASA Technical Memorandum 109025, NASA National Aeronautics and Space Administration, Langley Research Center Hampton, Virgina 23681-0001, October 1993.

[KaNe93b] J.A. Kaplan and M.L. Nelson. A Comparison of Queueing, Cluster and Distributed Computing Systems. NASA Technical Memorandum 109025, NASA National Aeronautics and Space Administration, Langley Research Center Hampton, Virgina 23681-0001, October 1993.

[LiKe87] F. Lin and R. Keller. The gradient model load balancing method. In *IEEE Transactions on Software Engineering*, vol. 13, no. 1, January 1987.

[LiLi88] M. Litzkow and M. Livny. Condor - a hunter of idle workstations. In *Proc. of 8th International Conference on Distributed Computing Systems*, San Jose, USA, 1988.

[Ludw92] T. Ludwig. Lastverwaltungsverfahren für Mehrprozessorsysteme mit verteiltem Speicher. Dissertation der Technischen Universität München, Institut für Informatik, 1992.

[MuLi87] M.W. Mutka and M. Livny. Scheduling remote processing capacity in a workstation-processor bank network. In *Proc. of 7th International Conference on Distributed Computing Systems*, Berlin, West-Germany, September 1987.

[NiSt93] H. Nishikawa and P. Steenkiste. A general architecture for load balancing in a distributed-memory environment. In *Proceedings 13th Int. Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, USA, May 1993.

[Slom94] M. Sloman. *Network and Distributed Systems Management.* Addison Wesley, 1994.

[StSi84] J.A. Stankovic and I.S. Sidhu. An adaptive bidding algorithm for processes, clusters and distributed groups. In *Proceedings of the 4th International Conference on Distributed Computing Systems*, San Francisco, California, May 1984.

[ThLa89] M.M. Theimer and K.A. Lantz. Finding idle machines in a workstation-based distributed system. In *IEEE Transactions on Software Engineering*, vol. 15, no. 11, November 1989.