Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Rahmatullah Arbabzadah

Design und Aufbau einer Managment-Umgebung zur Steuerung von numerischen Berechnungen auf einem High-Performance Cluster

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Rahmatullah Arbabzadah

Design und Aufbau einer Managment-Umgebung zur Steuerung von numerischen Berechnungen auf einem High-Performance Cluster

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Hans Heinrich Heitmann
Zweitgutachter : Prof. Dr. -Ing. Axel Schumacher

Abgegeben am 4. Mai 2009

Inhaltsverzeichnis

1 Einführung **5**

1.1 Motivation und Einordnung der Arbeit 5
1.2 Themenbeschreibung und Themenabgrenzung 5
1.3 Ziel der Arbeit 6

2 Grundlagen **7**

2.1 Hard und Software 7
2.2 Scheduling-Algorithmen 9
2.2.1 Preemptives und nicht-preemptives Scheduling-Verfahren 9
2.2.2 Kategorien von Scheduling-Algorithmen 10
2.2.3 Scheduling in Stapelverarbeitungs-Systemen 11
2.2.4 Scheduling in interaktiven Systemen 13
2.3 Parallelrechner 15
2.3.1 Klassifizierung nach Flynn 15
2.3.2 Klassifikation von Parallelrechnern 16
2.3.3 Multiprozessorsysteme 17
2.3.4 Uniform Memory Acces (UMA)-busbasierte Symmetrische Multiprozessoren (SMP)-Architekturen 18
2.3.5 Vergleich zwischen Shared-Memory-Multiprozessor und Message-Passing-Multicomputer 18
2.4 Parallele Leistungsmaße 20
2.5 Finite Element Methode (FEM) 21
2.5.1 CAD und FEM 22
2.6 LS-Dyna 23
2.6.1 Anwendungsgebiete von LS-Dyna 24

3 Auswahl eines Jobmanagement-Systems **25**

3.1 Erwartungen an ein Jobmanagement-System 25
3.2 Jobmanagement-Systeme 26
3.2.1 Load Sharing Facility (LSF) 26
3.2.2 OpenPBS, Portable Batch System 29
3.3 Bewertung von PBS Professional, OpenPBS und LSF 32
3.4 Auswertung der Auswahl eines Jobmanagement-Systems 35

4 Analyse des Ressourcen-Bedarfs von LS-Dyna Berechnungen **36**
4.1 Einfluß der CPU . . . 36
4.1.1 LS-Dyna-Inputfile . . . 37
4.1.2 Szenarien . . . 41
4.1.3 Bewertung der Szenarien . . . 49
4.2 Einfluß von Shared-Memory und Message-Passing-Multicomputer Systemen 51
4.2.1 Isolierte Nutzung der Rechner . . . 51
4.2.2 Gemeinsame Nutzung der Rechner . . . 53

5 Zuweisung von Cluster-Ressourcen **56**
5.1 Aufbau des Systems . . . 56
5.2 Zuteilungssoftware . . . 57
5.2.1 Ein und Ausgangsparameter . . . 57
5.2.2 Vorhersagen von LS-Dyna . . . 57
5.2.3 Ressourcenverteilung . . . 59
5.2.4 Arbeitsweise des Algorithmus . . . 61
5.3 Implementierung der Zuteilungssoftware . . . 64
5.4 Bewertung der Zuteilungssoftware . . . 67

6 Zusammenfassung **68**
6.1 Forschungsbedarf . . . 68
6.2 Fazit . . . 69

Tabellenverzeichnis **70**

Abbildungsverzeichnis **71**

Literaturverzeichnis **73**

1 Einführung

1.1 Motivation und Einordnung der Arbeit

In der heutigen Zeit werden die Forschungen und die Algorithmen immer komplexer und es werden immer mehr Daten analysiert. Dadurch steigt auch der Berechnungsaufwand. Um die Wünsche der Fahrzeugingenieure zu realisieren, die sicherere und effizientere Autos entwickeln, der Astronomen, die versuchen hinter den Sinn des Universums zu kommen oder der Biologen, die versuchen die Geheimnisse des menschlichen Genoms zu verstehen, muss eine genaue Anzahl an CPU's zur Verfügung gestellt werden, um die optimale, dass heißt die kürzeste Durchlaufzeit (Zeit von Prozessstart bis Prozessbeendigung) eines Jobs (Berechnung) zu realisieren. Ein Überfluss an Ressourcen, die für ein Job zugeteilt werden, führt zu einer sehr geringen bzw. zu keiner Reduktion der Rechenzeit, weil der Kommunikationsaufwand zwischen den CPU's unverhältnismaßig ansteigt. Bei einer zu geringen Anzahl an Ressourcen wiederum, kommt es zu einer langen Durchlaufzeit des Jobs. Wird davon ausgegangen, dass im ersten Fall die Ressourcen von mehreren Benutzern geteilt werden und im zweiten Fall genügend Ressourcen verfügbar sind, dann führen beide Szenarien dazu, dass die Ressourcen nicht effizient genutzt werden.

1.2 Themenbeschreibung und Themenabgrenzung

An der HAW-Hamburg wird im Bereich des Fahrzeugbaus ein Cluster zur Verfügung gestellt, der den Studenten und Mitarbeitern ermöglichen soll, ihre Jobs (Berechnungen) mit Hilfe von Livermore Software-Dyna (LS-Dyna) optimal durchzuführen. Dies wird dadurch gewährleistet, in dem die optimale Durchlaufzeit (Zeit von Prozessstart bis Prozessbeendigung) eines Jobs realisiert wird. Hierbei ist die optimale Durchlaufzeit abhängig von der Anzahl der Ressourcen (CPU's), die ein Job in Anspruch nimmt. Um den Cluster bestmöglich in Betrieb zu nehmen, ist es erforderlich ein Jobmangement-System auf den Cluster zu installieren und zu konfigurieren. Die Aufgabe eines Jobmangement-Systems ist es, die Anforderungen der Anwender nach IT-Ressourcen mit den real verfügbaren Ressourcen möglichst in Einklang zu bringen, und zwar in einer Form, die dem Bedarf des Unternehmens bzw. der Abteilung auch tatsächlich gerecht wird, das heißt z.B. die Jobs je nach Dringlichkeit entsprechend zu

priorisieren. Desweiteren soll ein Jobmanagement-System, Hewllett-Packard-Message Passing Interface (HP-MPI) und LS-Dyna unterstützen. Aus diesem Grunde wird eine Recherche über Softwareanbieter bezüglich der Unterstützung von LS-Dyna, HP-MPI, und weiteren Diensten des Jobmanagement-Systems durchgeführt. Um den Cluster bestmöglichst auszulasten, ist es erforderlich das System zu testen, sowie LS-Dyna Jobs (Berechnungen) in verschiedenen Szenarien zu bewerten. Dabei werden LS-Dyna und HP-MPI als Black Box (geschlossenes System unter Vernachlässigung des inneren Aufbaus) gesehen.

1.3 Ziel der Arbeit

Im Rahmen dieser Arbeit soll eine Zuteilungssoftware entwickelt werden, die als Schnittstelle zwischen Benutzer und Jobmanagement-System agiert. Das heißt, dass die Benutzer ihre Berechnungen der Zuteilungssoftware übergeben. Ab diesem Punkt übernimmt die Zuteilungssoftware die weiteren Bearbeitungsschritte. Ziel der Arbeit ist es, eine Zuteilungssoftware zu entwickeln, die den Cluster bestmöglich auslastet. Dies wird dann möglich, wenn die Zuteilungssoftware die Ressourcen optimal zwischen den Benutzern verteilt. Durch die optimale Ressourcenverteilung wird die mittlere Durchlaufzeit minimiert. Um dies realisieren zu können, muss die Zuteilungssoftware eine Vorhersage über den Ressourcenverbrauch eines Jobs (Berechnung) machen. Desweiteren ist es notwendig, dass die Zuteilungssoftware die Informationen über die Auslastung des Clusters ermittelt, um die freien Rechner für den Job herauszufiltern.

2 Grundlagen

2.1 Hard und Software

Wie in der Einleitung beschrieben wurde, wird an der HAW-Hamburg im Bereich des Fahrzeugbaues ein Cluster in Betrieb genommen (siehe Abbildung 2.1).

Abbildung 2.1: Cluster der HAW-Hamburg

Auf dem Cluster können unter anderem Berechnungen mit der Crash-Simulations-Software LS-Dyna durchgeführt werden.

Der Cluster, besteht aus 12 Rechnern. Jeder Rechner beinhaltet einen Double-Quad-Core-Prozessor und 16 GB Random Access Memory (RAM). Die Verbindung zwischen den Rechnern wird mit dem InfiniBand realisiert. In Abbildung 2.2 werden die LS-Dyna Jobs mit ihren Ausführungszeiten unter der Verwendung einer CPU dargestellt. Einige der dargestellten Jobs werden im Verlauf der Bachelorarbeit verwendet um Theorien darzulegen und auf Probleme aufmerksam zu machen.

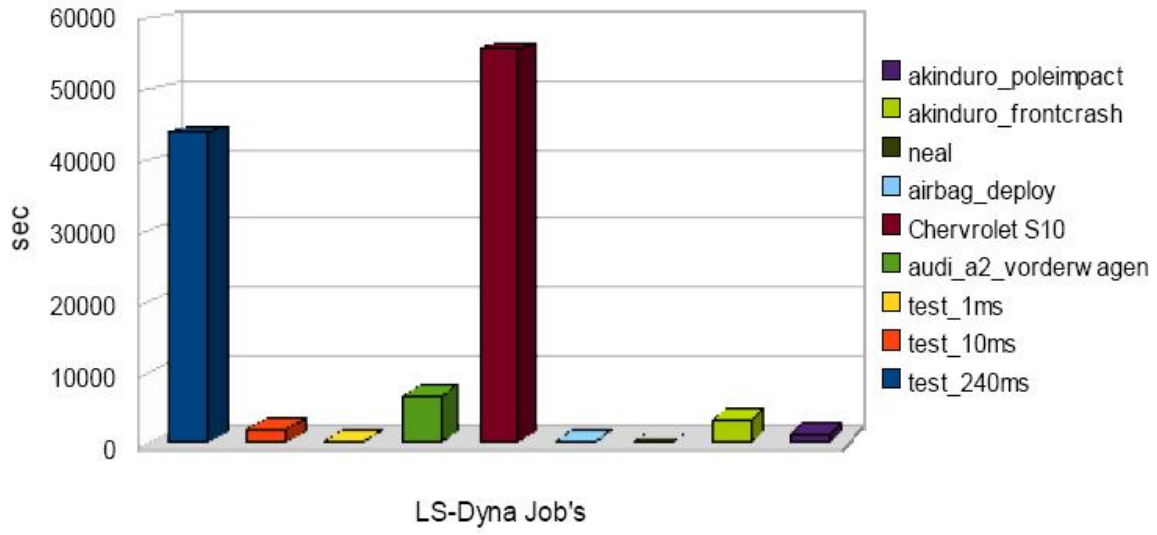

Abbildung 2.2: LS-Dyna Jobs

2.2 Scheduling-Algorithmen

Nach **Tannenbaum** wird Scheduling so beschrieben, dass mehrere Prozesse eines Rechners zur selben Zeit um die CPU konkurrieren. Diese Situation tritt dann auf, sobald zwei oder mehrere Prozesse gleichzeitig rechenbereit sind. Falls nur eine CPU verfügbar ist, muss entschieden werden, welcher Prozess als nächstes läuft. Der Teil des Betriebssystems, der diese Wahl trifft, wird Scheduler genannt. Der Algorithmus, den er verwendet, heißt **Scheduling-Algorithmus** ([2], S.148-S.149).

2.2.1 Preemptives und nicht-preemptives Scheduling-Verfahren

In einem nicht-preemptiven Scheduling-Verfahren welches in Abbildung 2.3 dargestellt wird, übergibt der Scheduler einem Prozess die angeforderten Ressourcen und wartet, bis diese vollständig abgearbeitet ist.

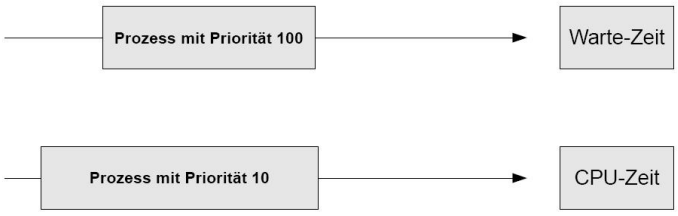

Abbildung 2.3: nicht-preemptives Scheduling-Verfahren

Hierbei entsteht der Nachteil, dass die höher priorisierten Prozesse warten müssen bis die Ressourcen freigegeben werden.

Preemption beruht auf der Annahme, dass es unterschiedliche Wichtigkeiten für unterschiedliche Jobs gibt. Diese Jobs benötigen ihre zugeteilten Ressourcen sofort, damit sie so schnell wie möglich abgeschlossen werden können. Andere Jobs sind weniger zeitkritisch und dauern in der Regel ziemlich lange. Die Dauer dieser Jobs liegt normalerweise bei einigen Tagen oder Wochen. In Abbildung 2.4 wird ein Beispiel dargestellt. Hier werden dem Prozess A mit Priorität 10 Ressourcen vor Fertigstellung seiner Berechnung wieder entzogen, um sie zwischenzeitlich einem Prozess B mit Priorität 100 zuzuteilen. Der Prozess A wird dabei in

seiner Ausführung unterbrochen und verharrt in seinem momentanen Zustand, bis ihm wieder Ressourcen zugeteilt werden. Hierbei besteht der Vorteil, dass relevante Prozesse sofort an die Reihe kommen und nicht auf CPU-Zeit warten mussen ([6], S.480 - S.483).

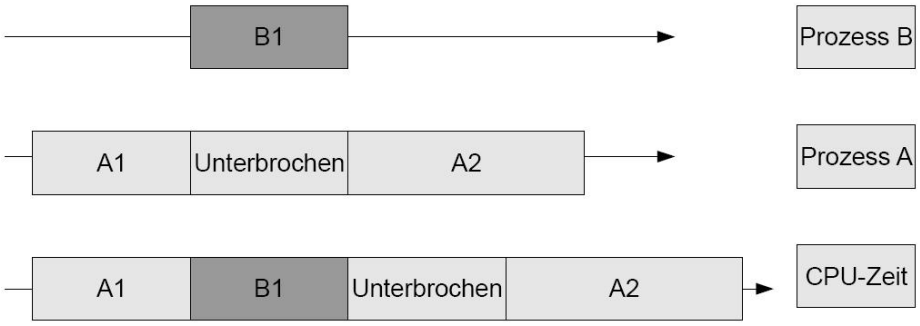

Abbildung 2.4: preemptiven Scheduling-Verfahren

2.2.2 Kategorien von Scheduling-Algorithmen

Es ist üblich, in verschiedenen Umgebungen auch verschiedene Scheduling-Algorithmen zu benutzen. Diese Situation ergibt sich daraus, dass die verschiedenen Umgebungen unterschiedliche Ziele haben, welche man nicht mit einem Scheduling-Algorithmus optimieren kann. Drei Umgebungen lassen sich unterscheiden: **1. Stapelverarbeitung, 2. Interaktion, 3. Echtzeit**. In dieser Bachelorarbeit wird im Detail auf Scheduling-Algorithmen für **Stapelverarbeitung** und **Interaktion** eingegangen ([2], S.152).

Im folgenden werden die Ziele des Scheduling-Algorithmus bei verschiedenen Umgebungen aufgelistet ([2], S.153).

Alle Systeme

- Fairness: jeder Prozess bekommt Rechenzeit der CPU
- Balance: alle Teile des Systems sind ausgelastet

Stapelverarbeitungssysteme

- Durchsatz: Maximierung der Jobs pro Stunde
- Turnaround-Zeit: Minimierung der Zeit vom Start bis zur Beendigung eines Prozesses

Interaktive Systeme

- Antwortzeit: antworte schnellstmöglich auf Anfragen
- Proportionalität: auf die Bedürfnisse des Nutzers eingehen

Echtzeitsysteme

- Meeting Deadlines: Zeitschranken einhalten
- Predictability: Vorhersagbarkeit

2.2.3 Scheduling in Stapelverarbeitungs-Systemen

Einführung

In Stapelverarbeitungssystemen gibt es keine ungeduldigen Benutzer, die auf eine sofortige Antwort an ihrem Terminal warten. Folglich sind nicht unterbrechende oder unterbrechende Algorithmen mit langen Zeiten für jeden Prozess annehmbar. Dieser Ansatz verringert die Zahl der Prozesswechsel und verbessert daher die durchschnittliche Durchlaufzeit der Prozesse ([2], S.152).

Scheduling-Algorithmus in Stapelverarbeitungs-Systemen

First-Come First-Served (FCFS) ist ein nicht präemptiver Scheduling-Algorithmus. FCFS kann man sich als eine Warteschlange mit rechenbereiten Prozessen vorstellen, die auf die CPU warten. Sobald ein Prozess neu gestartet wird, wird er am Ende der Warteschlange eingereiht. Die Prozesse bekommen die CPU in der Reihenfolge, in der sie in der Warteschlange eingereiht werden. Wenn ein Prozess die CPU bekommt, kann er beliebig lange rechnen. Wenn ein laufender Prozess blockiert wird, kommt der nächste Prozess der am Anfang der Warteschlange ist an die Reihe. Wenn ein blockierter Prozess wieder rechenbereit wird, wird er wie ein neuer Prozess am Ende der Warteschlange eingereiht. Dieser Algorithmus hat einen sehr großen Nachteil. Angenommen drei Jobs A, B und C mit den Längen 10, 4 und 3 sec, die fast gleichzeitig eintreffen (siehe Abbildung 2.5a), werden erst nach FCFS eingeordnet und bearbeitet. In diesem Fall liegt die **durchschnittliche Durchlaufzeit** bei (10+14+17)/3 = 13,67 sec und die **durchschnittliche Wartezeit** bei (0+10+14)/3 = 8 sec. Wenn die Jobs so angeordnet sind, dass die Jobs mit der kürzesten Durchlaufzeit zuerst bearbeitet werden, welches in Abbildung 2.5b dargestellt wird, dann beträgt die **durchschnittliche Durchlaufzeit** (3+7+17)/3 = 9 sec und die **durchschnittliche Wartezeit** (0+3+7)/3 = 3,3 sec. Dieser Ansatz führt zu einem neuen Scheduling-Algorithmus (Shortest Job First), der im unterem Abschnitt beschrieben wird. Desweiteren müssen eventuell kleinere Jobs lange auf ihre Abarbeitung warten ([2], S.155 - S.156).

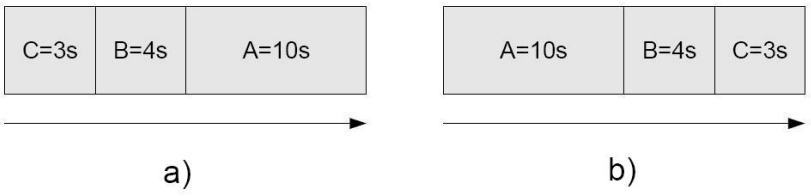

Abbildung 2.5: First-Come First-Served [2]

Shortest Job First (SJF) ist ein nicht präemptiver Stapelverarbeitungsalgorithmus. Hierbei wird davon ausgegangen, dass die Laufzeit vorher bekannt ist. Das heisst SJF ist nur dann optimal, wenn alle Aufgaben gleichzeitig verfügbar sind. Als Gegenbeispiel werden in Abbildung 2.6a fünf Prozesse, A bis E mit jeweiligen Laufzeiten von 2,4,1,1 und 1 sec dargestellt. Ihre Ankunftszeiten sind 0, 0, 3, 3 und 3 sec. Es können zur Zeitpunkt 0 A und B

bearbeitet werden, da C bis E zum Zeitpunkt 0 noch nicht angekommen sind. Wird hier SJF Verwendet, lautet die Reihenfolge A, B, C, D, E und die **durchschnittliche Durchlaufzeit** beträgt ((2+6+7+8+9)/5) = 6,4 sec. Wenn davon ausgegangen wird, dass alle Prozesse zum Zeitpunkt 0 bekannt sind, also C, D, E, A und B (siehe Abbildung 2.6b), beträgt die **durchschnittliche Durchlaufzeit** ((1+2+3+5+9)/5) = 4 sec ([2], S.156 - S.157).

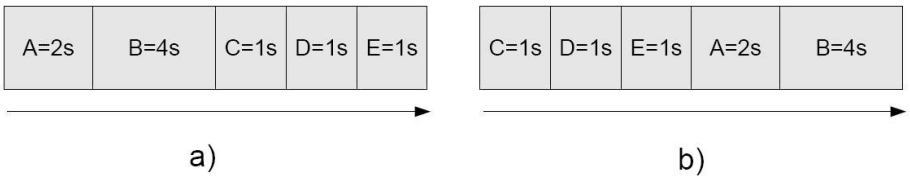

Abbildung 2.6: Shortest Job First [2]

Shortest Remaining Time Next (SRTN) ist eine präemptive Version von SJF. Sobald eine neue Aufgabe ankommt, wird deren Zeit mit der Zeit des gegenwärtigen Prozesses verglichen. Falls der neue Prozess weniger Zeit braucht um fertig zu werden als der gegenwärtige Prozess, wird der aktuelle Prozess gestoppt und der neue Prozess begonnen. Dieser Scheduler-Algorithmus bietet neuen Prozessen, die eine kurze Laufzeit haben, einen guten Dienst ([2], S. 157).

2.2.4 Scheduling in interaktiven Systemen

Interaktive Scheduling-Algorithmen werden dort benutzt, wo das System schnellstmöglich auf Anfragen von Nutzern antworten sollte. Weiterhin können sie in Stapelverarbeitungs-Systemen als CPU-Scheduler verwendet werden ([2], S.159).

Round-Robin (RR): Beim RR bekommt jeder Prozess einen Zeitabschnitt (Quantum) zugewiesen, in dem er laufen darf. Der Algorithmus funktioniert so, dass der Scheduler eine Liste der lauffähigen Prozesse hat (siehe Abbildung 2.7a). Sobald der Prozess sein Quantum aufgebraucht hat, wird er an das Ende der Liste gelegt (siehe Abbildung 2.7b) und der am längsten wartende Prozess wird aktiviert. Die einzig interessante Frage von RR betrifft

die Länge des Quantums. Das zu kurze Setzen des Quantums verursacht zu viele Prozesswechsel und vermindert die CPU-Effizienz. Dies ist nicht im Sinne von Stapelverarbeitungs-Systemen. Ist das Quantum zu lang, bedeutet das, dass kleine Jobs lange warten müssen ([2], S.159 - S.160).

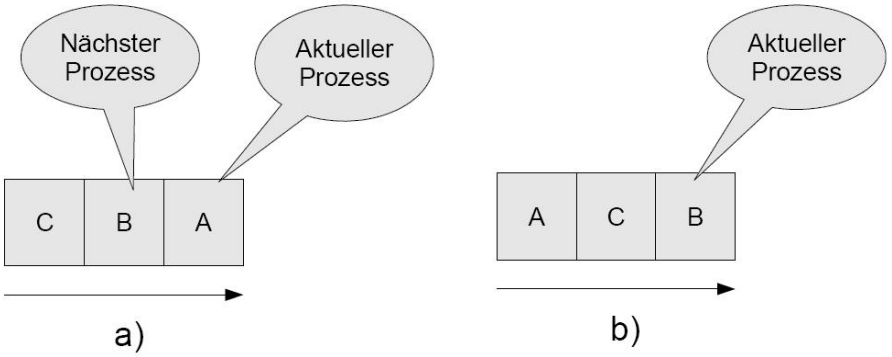

Abbildung 2.7: Round-Robin [2]

Fair-Share Scheduling: Bei der Strategie FCFS entsteht für kleine Jobs ein Nachteil, denn sie können nicht vor größeren Jobs, die als erstes eingetroffen sind, berechnet werden. Bei der Fairshare-Strategie liegt der Schwerpunkt auf der Gerechtigkeit zwischen den Anwendern bzw. Gruppen, so dass jeder einmal mit seinem Job an der Reihe ist. Bei drei Benutzern (A, B, C), die gleichzeitig jeweils einen Prozess ausführen, teilt der Prozess-Scheduler die verfügbaren CPU-Zyklen beispielsweise so, dass jeder Benutzer 33,3% aller Zyklen bekommt (100 % / 3=33,3%). Wenn der Benutzer A einen zweiten Prozess startet, wird jeder Benutzer immer noch 33,3% aller Zyklen bekommen. Da A zwei Prozesse gestartet hat, wird für jeden Prozess 16,65% CPU-Zyklen bereitgestellt (33,3% / 2= 16,65%). Wenn aber ein neuer Benutzer D einen Prozess im System startet, wird der Scheduler die verfügbaren CPU-Zyklen neu aufteilen, so dass jeder Benutzer 25% aller Zyklen bekommt (100% / 4= 25%). Somit hat A 25% der CPU-Zyklen, da A aber zwei Prozesse gestartet hat, hat jeder Prozess 12,5% der CPU-Zyklen (25% / 2 = 12,5%) ([2], S.165).

2.3 Parallelrechner

2.3.1 Klassifizierung nach Flynn

Flynn charakterisiert Rechnerstrukturen als Operatoren auf zwei verschiedenartigen Informationsströmen: Dem Befehlsstrom und dem Datenstrom. Entsprechend führt er darauf aufbauend eine zweidimensionale Klassifizierung ein, deren Hauptkriterien die Zahl der Befehlsströme und der Datenströme sind. Diese beruht auf den Merkmalen:
1.) Der Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Befehl
2.) Der Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Datenwert
Daraus resultieren vier Klassen von Rechnerstrukturen: In Abbildung 2.8 werden die von Flynn beschriebenen Klassen in einer Tabelle dargestellt.

	Single Instruction	Multiple Instruction
Single Data	**SISD**	**MISD**
Multiple Data	**SIMD**	**MIMD**

Abbildung 2.8: Klassifizierung nach Flynn

Single Instruction - Single Data (SISD) sind Einprozessor-Rechner mit einem Programmspeicher, die ihre Aufgaben sequentiell abarbeiten. SISD-Rechner können z.B. als Personal-Computer nach der Von-Neumann-Architektur oder der Harvard-Architektur aufgebaut werden.

Single Instruction - Multiple Data (SIMD) ist eine Rechnerarchitektur mit Vektorprozessoren, bei der viele Prozessoren ihre Befehle von einem Befehlsprozessor erhalten und gleichzeitig unterschiedliche Daten verarbeiten. Ein Befehl wird also gleichzeitig auf mehrere Datensätze angewendet und parallel abgearbeitet.

Multiple Instruction - Single Data (MISD) ist eine Rechnerarchitektur mit Multiprozessorsystem. Hierbei bekommen alle Prozessoren gleichzeitig verschiedene Befehle, welche sie auf den Daten eines einzigen Datenstroms ausführen.

Muliple Instruction - Multiple Data (MIMD) ist eine Rechnerarchitektur, die aus vielen Prozessoren besteht, die als Teilrechner parallel arbeiten und von getrennten Befehlen gesteuert werden. Jeder Teilrechner führt gleichzeitig verschiedene Operationen auf verschieden gearteten Eingangsdatenströmen durch. Die Teilrechner selbst können in ihrer Rechnerarchitektur optimiert und als SISD oder SIMD ausgeführt werden. Bei der MIMD-Architektur

erhält jeder Prozessor eigene Befehle und hat für die Speicherung einen eigenen Speicher ([1], S.197 - S.201).

2.3.2 Klassifikation von Parallelrechnern

In Abbildung 2.9 werden drei Parallelrechner vorgestellt. Bei diesen Typen, besonders bei Rechnern mit mehr als 100 Prozessoren, spricht man von massivem Parallelismus.

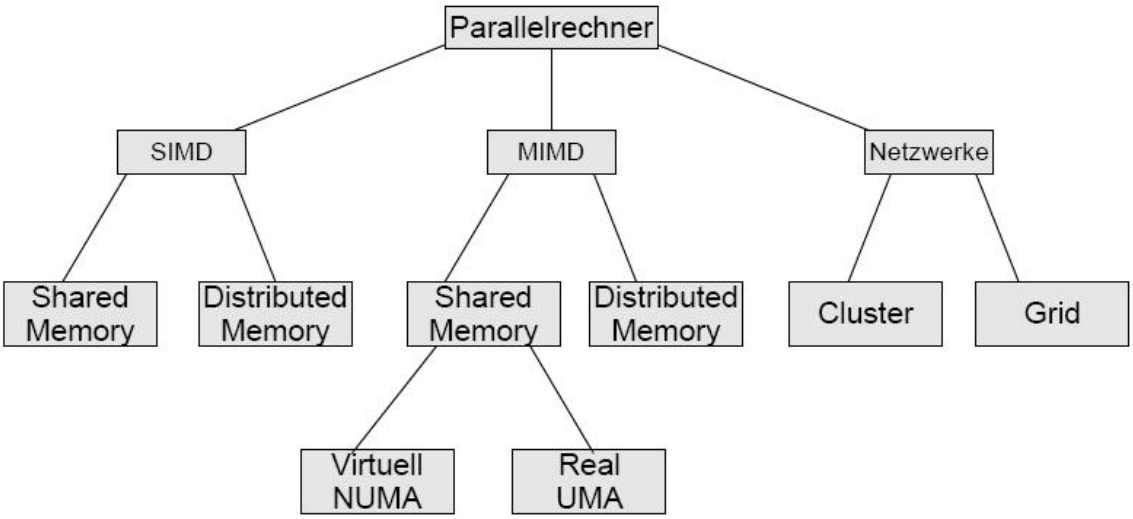

Abbildung 2.9: Klassifikation von Parallelrechnern [3]

Es wird in den Fällen von SIMD und MIMD danach klassifiziert, ob sich der jeweilige Rechner eines gemeinsam benutzten Speichers (Shared Memory) oder eines verteilten Speichers (Distributed Memory) bedient. Bei Rechnern, die sich eines gemeinsamen Speichers bedienen, spricht man auch von speichergekoppelten Systemen. Bei Rechnern, bei denen jeder Prozessor über einen lokalen Speicher verfügt, spricht man von nachrichtengekoppelten Systemen, da die Prozessoren jetzt untereinander über einen Nachrichten-Austausch (Message Passing) kommunizieren. Bei MIMD-Rechnern erfolgt bei Systemen mit gemeinsamen Speicher die weitere Unterteilung in reale und virtuelle gemeinsame Speicher. Die realen bzw. Uniform Memory Acces (UMA) basierten gemeinsamen Speicher werden in Kapitel 2.3.4 detailliert beschrieben. Bei Netzwerken unterscheidet man zwischen Cluster und Grid-Comuting. Beim Grid-Computing wird eine Vielzahl von unterschiedlichen Rechnern vernetzt, um ihre Rechenleistung für eine gemeinsam zu bearbeitetende Anwendung einzusetzen. Als Cluster von Rechnern bezeichnet man die koordinierte Benutzung von weitgehend gleichartigen und vernetzten Rechnern ([3], S.415 - S.417).

2.3.3 Multiprozessorsysteme

In Abbildung 2.10 werden drei Ansätze vorgestellt, mit deren Hilfe die Rechengeschwindigkeit von test_240ms (siehe Abbildung 2.2) um etwa das 16 Fache erhöht wird. Der **Shared-Memory-Multiprozessor** in Abbildung 2.10a, ist ein System, dass aus 2 bis n CPU's besteht, die über einen gemeinsamen Speicher kommunizieren.

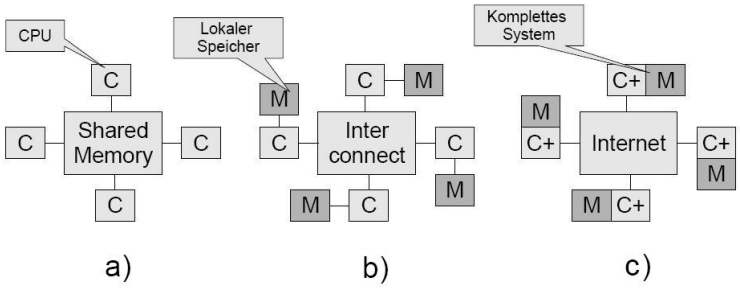

Abbildung 2.10: Multiprozessorsysteme [2]

Hierbei haben alle CPU's gleichermaßen Zugriff auf den gesamten physischen Speicher und können individuelle Wörter mit Hilfe von LOAD- und STORE-Instruktionen lesen und schreiben. Der Zugriff auf den Speicher benötigt in der Regel 10-50 ns. Als Nächstes wird das System **Message-Passing-Multicomputer** in Abbildung 2.10b betrachtet. In diesem System hat jede CPU ihre eigene lokale Speichereinheit und die Kommunikation erfolgt über Nachrichten auf einer Hochgeschwindigkeitsverbindung zwischen den Rechnern, da dieser Ansatz keinen gemeinsamen Speicher unterstützt. Mit einer guten Verbindung kann eine kurze Nachricht in 10-50 μs verschickt werden. Dies ist weitaus länger als der Speicherzugriff aus Abbildung 2.10a. Da es in diesem Design keinen gemeinsamen Speicher gibt, ist die Realisierung viel einfacher als bei Shared-Memory-Systemen, jedoch schwerer zu programmieren. Demzufolge hat jedes Design seine Vor- und Nachteile. Das dritte Modell **Wide-Area-System**, welches in Abbildung 2.10c dargestellt wird, verbindet ganze Rechneranlagen über ein Wide Area Network (WAN), z.B. das Internet. Die Kommunikation zwischen den Rechnern wird anhand von Nachrichtenaustausch realisiert. Hier hat jeder Rechner seinen eigenen Speicher. Der Austausch einer kurzen Nachricht zwischen zwei Rechnern auf diesem Modell dauert 10-50 ms. Die drei vorgestellten Typen unterscheiden sich in der Länge der Zeitverzögerung, die beim Nachrichtenaustausch auftritt, um jeweils den Faktor tausend ([2], S.539 - S.541).

2.3.4 Uniform Memory Acces (UMA)-busbasierte Symmetrische Multiprozessoren (SMP)-Architekturen

In Abbildung 2.11a ist ein einfaches Multiprozessorsystem dargestellt. Die Kommunikation zwischen den CPU's wird durch den Speicher realisiert. Liest eine CPU ein Speicherwort, so überprüft sie zuerst, ob der Binary Unit System (BUS) belegt ist. Sobald er frei ist, legt die CPU die Adresse des Speicherwortes auf den BUS, fügt ein paar Kontrollsignale hinzu und wartet, bis der Speicher das gewünschte Wort seinerseits auf den BUS legt. Ist der BUS durch eine andere CPU belegt, so muss dies warten bis der BUS wieder frei ist. Bei zwei oder drei CPU's ist der gemeinsame Zugriff auf den BUS nicht sehr zeitkritisch. Wenn aber mehr als drei CPU's um den BUS konkurrieren kann es vorkommen, dass die CPU's den größten Teil der Rechenzeit ungenutzt bleiben. Die Lösung für dieses Problem, wird anhand von Abbildung 2.11b erklärt. Hierbei hat jede CPU einen Cache. Da nun viele Lesezugriffe aus dem lokalen Cache bedient werden können, gibt es viel weniger Zugriffe auf den BUS und das System ist in der Lage mehr CPU's zu unterstützen ([2], S.542 - S.543).

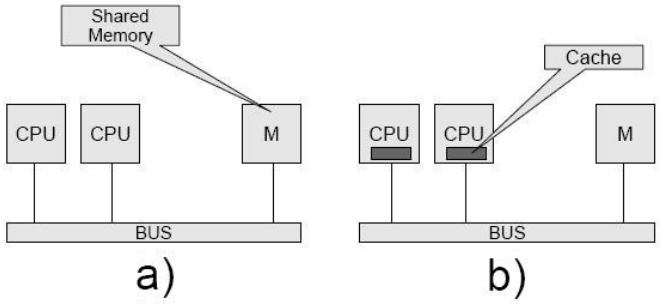

Abbildung 2.11: UMA-busbasierte SMP-Architekturen [2]

2.3.5 Vergleich zwischen Shared-Memory-Multiprozessor und Message-Passing-Multicomputer

In Kapitel 2.3.3 wird erläutert, dass der Performance-Unterschied beim Nachrichtenaustausch zwischen den Rechnersystemen Shared-Memory-Multiprozessor und Message-Passing-Multicomputer ungefähr beim Faktor 1000 liegt. Wiederum wird in Kapitel 2.3.4 beschrieben, dass Shared-Memory Systeme, die mehrere CPU's besitzen langsamer werden, da die CPU's um den BUS konkurrieren. Anhand eines Beispiels, wird das Verhalten der beiden Systeme verglichen. In Abbildung 2.12 ist eine Message-Passing-Multicomputer mit zwei Rechnern zu sehen. Die Rechner bestehen aus einem Shared-Memory System mit acht CPU's.

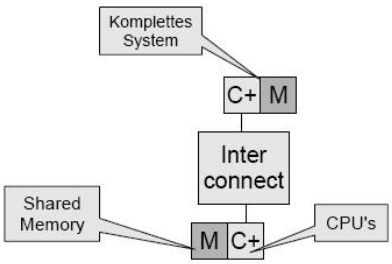

Abbildung 2.12: Message-Passing-Multicomputer [2]

In Abbildung 2.13 sind zwei Graphen dargestellt. Der Shared-Memory-Multiprozessor wird durch den blauen Graphen simuliert. Hierbei wird die Berechnung mit einer unterschiedlichen Anzahl an CPU's berechnet. Das Message-Passing-Multicomputer wird durch den roten Graphen simuliert. Hier wird die gleiche Berechnung auf zwei Rechner verteilt und berechnet. Dies bedeutet aber auch, dass die CPU-Anzahl zwischen den beiden Rechnern geteilt wird, wobei die Anzahl der CPU's mit der des blauen Graphen übereinstimmt.

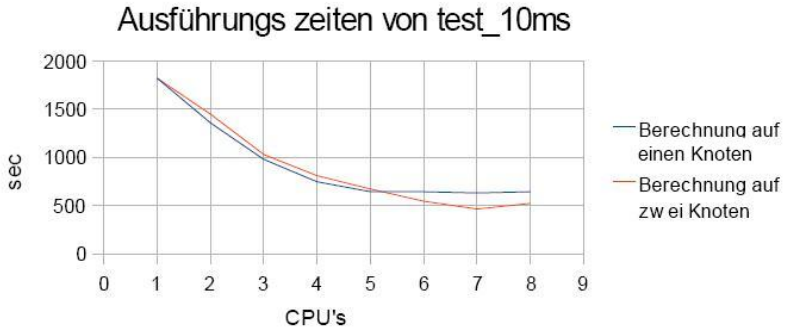

Abbildung 2.13: Ausführungszeiten von test_10ms.k

Anhand der beiden Graphen ist zu erkennen, dass die Ausführungszeiten von Shared-Memory System, die zwischen zwei bis fünf CPU's liegen, geringer sind als beim Message-Passing-Multicomputer. Hieraus wird abgeleitet, dass der Nachrichtenaustausch beim Message-Passing-Multicomputer länger dauert (siehe Kapitel 2.3.3) als die Konkurrierenden CPU's in einem Shared-Memory System. Wenn aber die Anzahl der CPU's mehr als fünf beträgt, verringern sich die Ausführungszeiten von Message-Passing-Multicomputer gegenüber dem Shared-Memory System, da die Menge der CPU's, die um den BUS konkurrieren, im Shared-Memory System gestiegen ist.

2.4 Parallele Leistungsmaße

Um unter den gleichen Bedingungen die Effizienz und den Speedup von parallelen Rechnern zu einem sequentiellen Rechner zu ermitteln, werden folgende Formeln benutzt:

$Speedup(p) = \frac{Tseq}{Tpar}$

$Effizienz(p) = \frac{Speedup(p)}{p}$

$ideal(p) : Speedup(p) = p \; und \; Effizienz(p) = 1$

p: Anzahl der Prozessoren.
Tseq: Ausführungszeit von Job A auf einem Prozessor.
Tpar: Ausführungszeit von Job A auf p Prozessoren.

Der Speedup-Faktor s eines parallelen Rechners gibt an, wieviel mal schneller ein paralleler Algorithmus im Vergleich zum sequentiellen abgearbeitet werden kann.

Effizienz ist das Verhältnis von Nutzen zum Aufwand, mit dem der Nutzen erzielt wird ([4], S.122).

In Abbildung 2.14 sind zwei Speedup-Graphen dargestellt. Im Idealfall hat der Graph eine lineare Steigung. Typischerweise jedoch divergiert die Speedup-Kurve immer weiter vom Ideal bei steigender Anzahl an CPU's.

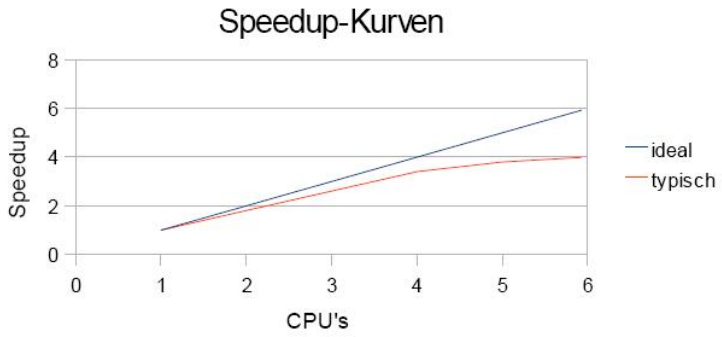

Abbildung 2.14: Parallele Leistungsmaße

2.5 Finite Element Methode (FEM)

In der folgenden Bachelorarbeit wird die FEM als Black-Box (ein geschlossenes System unter Vernachlässigung des inneren Aufbaus) angesehen. An dieser stelle soll dem Leser nur eine grobe Vorstellung davon vermittelt werden, welche Eigenschaft diese mathematische Methode hat und in welchen Bereichen diese mathematische Methode angewendet wird.

Mit der FEM können nahezu alle Vorgänge der Technik auf dem Computer simuliert werden, wie zum Beispiel, die Vorausbestimmung des Verhalten von Bauwerken. Bei der Konstruktion einer Brücke ist es von Interesse, ob diese ihr Eigengewicht tragen kann und auch Windböen und Erdbeben standhalten wird. Das neu entwickelte Bügeleisen soll sich möglichst gleichmäßig und schnell erwärmen. Der Hubmagnet muss ausreichend starke Magnetfelder erzeugen, um die gewünschten Kräfte zu ermöglichen. Dabei wird der Körper in möglichst viele kleine Elemente in Form von Linien, Dreiecke, Vierecke, Tetraeden, Pentaeden oder Hexaeden zerlegt, die an ihren Eckpunkten, den sogenannten Knoten, miteinander fest verbunden sind. Kleine Elemente deshalb, weil das näherungsweise über lineare Gleichungen formulierte Verhalten der Elemente nur für das unendlich kleine (infinitesimale) Element gilt, die Realität jedoch endlich große (finite) Elemente verlangt. Die Annäherung an die Realität ist daher umso besser, je kleiner die Elemente sind.
Diese Zerlegung erfolgt heute auf der Basis von einer rechnerunterstützten Konstruktion oder englisch Computer Aided Design (CAD). Ein CAD-Anwendungsprogramm ist ein EDV-Programm, welches den Anwender bei der Arbeit in Entwicklung, Konstruktion und Arbeitsvorbereitung, also bei Berechnungsaufgaben, bei der Informationsbereitstellung, beim Zeichnen und beim rechnerunterstützten Entwurf unterstützt [5].

2.5.1 CAD und FEM

In Abbildung 2.15 wird das Zusammenspiel von CAD und FEM beschrieben.

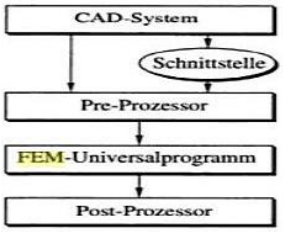

Abbildung 2.15: CAD und FEM [5]

Das Bauteil ist mit CAD erstellt worden. Hierbei kann es sein, dass die Hersteller zwischen dem CAD-System und dem FEM-System eine Direktkopplung realisiert haben. Das würde heißen, dass das Bauteil sofort vom FEM-System übernommen wird. Wenn hier aber zwei völlig autonome Systeme vorliegen, so muss das Bauteil über eine Standardschnittstelle wie IGES (Initial Graphics Exchange Specification) oder STEP (Standard for the Exchange of Product Model Data) transportbereit gemacht werden. Die Aufgabe des Pre-Prozessors ist es, ein berechenbares FE-Modell aus dem Bauteil zu generieren, d.h. Zuweisung der Elementdaten und der Materialwerte sowie Einbringung der Kräfte und Randbedingungen. Für die Anzeige des Ergebnisse, wird ein Post-Prozessor genutzt. Dieser stellt als Ergebnis z.B. die verformte Struktur sowie die Dehnungen und Spannungen in dem Bauteil dar ([5], S.5 - S.6).

2.6 LS-Dyna

LS-Dyna ist ein Simulationsprogramm, das mit Hilfe der Finiten-Element-Methode, (siehe Kapitel 2.5) arbeitet. Die Software wird von der Livermore Software Technology Corporation (LSTC) in Kalifornien entwickelt. LS-Dyna bietet die Möglichkeit, in der frühen Entwurfsphase bereits eine Aussage zum Strukturverhalten und der Optimierung des Designs zu geben. Weiterhin unterstützt LS-DYNA Massively Parallel Processing (MPP) (siehe Abbildung 2.16). Es wird dabei eine Gebietszerlegung des gesamten Simulationsmodells in kleine Teilmodelle vorgenommen, die jeweils auf einem eigenen Prozessor gelöst werden. Während des Zeitschrittes und vor allem am Ende des Zeitschrittes müssen die beteiligten CPU's miteinander kommunizieren. Hierfür ist dann das sogenannte Message Passing Interface (MPI) nötig [18].

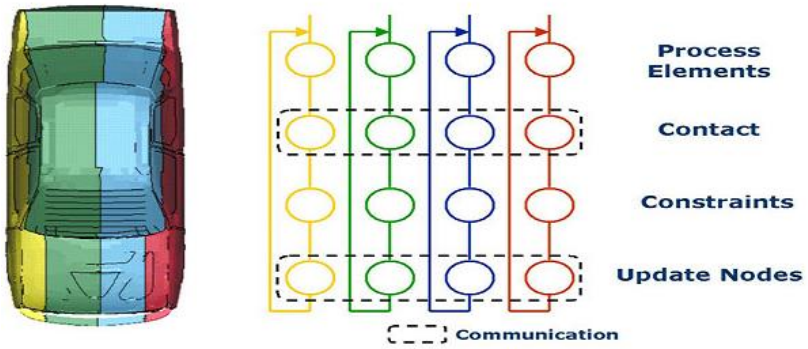

Abbildung 2.16: MPP [18]

2.6.1 Anwendungsgebiete von LS-Dyna

Crashsimulation, Insassensicherheit und Fußgängersicherheit

LS-Dyna hat seinen größten Anwenderkreis in der Automobilindustrie. Die Software wird sowohl zur Berechnung eines Gesamtfahrzeug-Crashs als auch für detaillierte Berechnungen der sicherheitsrelevanten Komponenten, wie der aktiven und passiven Fahrzeugsicherheit, sowie des Insassen und Fußgängerschutzes eingesetzt. Hierbei werden Untersuchungen zur Insassensicherheit unter Berücksichtigung von Airbags, Rückhaltesystemen und Dummy-Interaktionen, sowie Verbesserung der Fahrzeug-Frontendstruktur im Hinblick auf Verletzungsrisiken von Fußgängern unterstützt. Die Software erlaubt die Berechnung verschiedenster Crashlastfälle wie Frontalcrashs mit unterschiedlichen Überdeckungen, Seitencrashs mit verschiedenen Geschwindigkeiten und Fahrzeugüberschläge. Durch die Crashsimulationen wird die passive Crashsicherheit der heutigen Fahrzeuggenerationen entscheidend beeinflusst und verbessert. Hierbei entstehen kürzere Entwicklungszeiten. Desweiteren unterstützt LS-Dyna Crashsimulationen im Bereich von Schiffsbau, Schienenfahrzeugen und Flugzeugen [17].

Metallumformung

Eine weitere Domäne von LS-Dyna ist die Simulationen von Metallumformvorgängen. Hierbei werden Optimierungsansätze hinsichtlich der Oberflächeneigenschaften, Falten- oder Rissbildung unterstützt [17].

Falltest

Außerhalb der Automobilindustrie wird LS-Dyna im Bereich der Produktentwicklung eingesetzt. So lassen sich mit LS-Dyna die Auswirkungen des Aufpralls von Gegenständen simulieren. Hierzu gehören Falltest-Untersuchungen von Konsumgütern, nuklearen Transportbehältern und Vogelschlag-Simulationen gegen Flugzeugen, Turbinen oder Windschutzscheiben [17].

3 Auswahl eines Jobmanagement-Systems

3.1 Erwartungen an ein Jobmanagement-System

Ein Jobmanagement-System wird durch Queues, Scheduler, Ressourcen-Management, Job-Management, Dienste und Sicherheit, welche das System anbietet, charakterisiert (siehe Abbildung 3.1) ([11], S.51, S.60-S.61). Der Dienst Ressourcen-Management findet Anwendung bei der vom Verfasser entwickelten Zuteilungssoftware, und spielt daher eine wichtige Rolle im Rahmen dieser Arbeit. Weiterhin werden die Dienste Job-Management, Queues, Scheduler, Dienste und Sicherheit eines Jobmanagement-Systems untersucht.

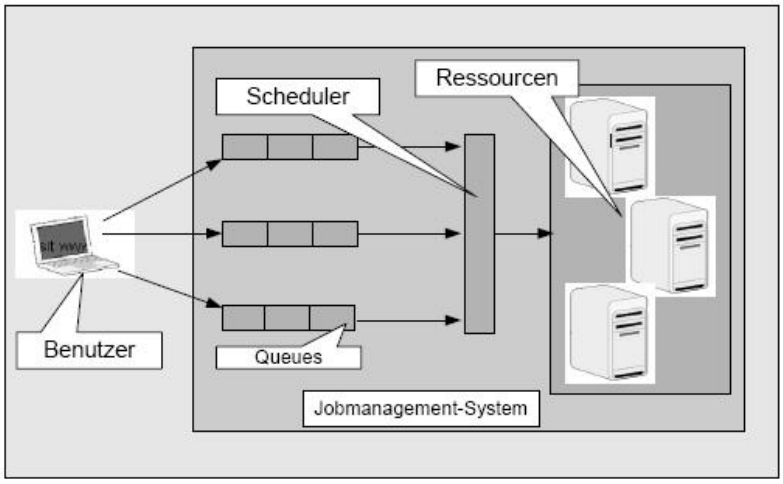

Abbildung 3.1: Jobmanagement-Systeme

Jobmanagement-Systeme dienen der Abstimmung zwischen den Anforderungen der Anwender nach IT-Ressourcen mit den real verfügbaren Möglichkeiten. Dies soll den Unternehmen helfen, eine optimale Ressourcenverteilung zu erzielen. Der Verfasser erwartet desweiteren von einem Jobmanagement-System, dass dieses verschiedenene Scheduling-Algorithmen anbietet.

Auf Grund der Tatsache, dass die einzelnen Jobs unterschiedlich relevant sind, ist es von Vorteil, dass ein Jobmanagement-System bei Ressourcenknappheit, unwichtige Jobs abbricht und die Ressourcen wichtigeren Jobs zuteilt. Die Erwartungen an einer Queue, werden durch unterschiedliche Attribute charakterisiert, wie zum Beispiel den Namen einer Queue, welcher die Queue eindeutig identifiziert, und die Queue-Begrenzung, welche die maximale Anzahl der Jobs definiert etc. Weiterhin soll ein Jobmanagement-System die Möglichkeit bieten, Ressourcen für einen festzulegenden Zeitpunkt zu reservieren oder einzelne Ressourcen einem bestimmten Benutzer oder einer Benutzergruppe zuzuordnen.

Die Berechnungen werden von LS-Dyna berechnet und mit Hilfe von HP-MPI parallelisiert. Daher ist es notwendig, dass die jeweiligen Jobmanagement-Systeme, LS-Dyna und HP-MPI unterstützen. Desweiteren werden die Dienste und die Sicherheit eines Jobmanagement-Systems unterschieden. Hierbei wird erwartet, dass ein Jobmanagement-System parallele Berechnungen, Job Array (eine Zusammenfassung von Jobs, die sich nur durch einem Indexparameter unterscheiden), Checkpointing (Speichern eines laufenden Jobs) und Job Migration (Unterbrechung laufender Prozesse, den Transfer auf einen anderen Rechner und die Wiederaufnahme) als Dienste anbietet. Um die Zuverlässigkeit des Systems zu gewährleisten, soll das System so agieren, dass es kontinuierlich arbeitet, auch wenn einige Rechner des Clusters nicht verfügbar sind.

3.2 Jobmanagement-Systeme

3.2.1 Load Sharing Facility (LSF)

Einführung

Load Sharing Facility ist ein generelles und verteiltes Jobmanagement-System des kanadischen Herstellers Platform Computing, welches die Lastverteilung in einer vernetzten Rechnerumgebung organisiert. Die Software ist plattformunabhängig und ermöglicht die optimale Nutzung der Ressourcen eines heterogenen Unix, Linux und Windows Netzwerks [20]. LSF funktioniert so, dass die Benutzer ihre Rechenaufgabe (Job) beim Server anmelden. Der Server platziert den Job in dem Queue, der für diesen Job geeignet ist. LSF übernimmt ab hier die Verwaltung und die Abarbeitung des Jobs ([8], S.20 - S.21). LSF Queues die in Tabelle 3.1 dargestellt werden, sollten nicht mit einfachen Warteschlangen (Queues) verwechselt werden, da sie durch ihre Konfigurationsattribute verschiedene Formen annehmen können ([9], S.60). Queues unter LSF können bei laufendem Betrieb konfiguriert, geöffnet, geschlosssen, aktiviert oder deaktiviert werden ([9], S.118).

Parameter	express	int	long	normal
QUEUE NAME	express	int	long	normal
PRIORITY	120	200	50	70
USERS	usersA	usersB	usersA	usersC

Tabelle 3.1: LSF Queues

Die Benutzer können ihre Berechnungen interaktiv oder mit einem Shell-Skript dem Cluster übergeben. Ist dies geschehen, benutzt LSF die Informationen über die Auslastung der Maschinen, wie CPU-Auslastung, Hauptspeicher-Auslastung, virtueller-Speicher, freier Platz in temporären Verzeichnissen, I/O-Rate, Anzahl der aktiven Benutzer und die Vorgabekriterien, welche der Benutzer durch sein Shell-Skript oder interaktive Eingaben dem Cluster übergeben hat ([12], S.2). Durch diese Informationen kann LSF die Jobs starten, suspendieren oder neu starten ([8], S.56). Bei Ausfall der Server verständigen sich die restlichen Maschinen automatisch auf einen neuen Server. Wenn alle Maschinen im Cluster beispielsweise wegen eines Stromausfalles abstürzen, werden noch nicht gestartete Jobs abgearbeitet, sobald der Cluster wieder online ist. Jobs, die zum Zeitpunkt des Absturzes einer Maschine liefen, werden auf anderen Rechnern automatisch wieder gestartet, verlieren also nur die bisher verbrauchte Rechenzeit. Desweiteren ist LSF so entworfen, dass es das Arbeiten fortsetzt, auch wenn einige Rechner des Clusters nicht verfügbar sind ([11], S.60 - S.61). LSF bietet zahlreiche Dienste, wie das vollständige Speichern des Jobs während der Ausführung (Checkpointing) ([9], S.326), bewegen der eigenen Jobs innerhalb eines Queues, sowie zwischen den Queues. LSF bietet eine Vielzahl von Befehlen (siehe Tabelle 3.2), um einen Job, der in den Queues angemeldet ist, zu beobachten oder zu kontrollieren. Es ist möglich die Ausgaben, sowie die Historie von laufenden Jobs zu beobachten ([13], S.66).

LSF befehle	Bedeutung der Befehle
bsub -q queue	Übergibt einen Job an einen LSF-Queue
bsub -m host	Übergibt einen Job an einen bestimmten Host
bsub -n proc	Legt die Anzahl der benötigten Prozessoren fest
bstop jobid	Stoppt alle Prozesse eines LSF-Jobs
bresume jobid	Setzt die Ausführung eines zuvor gestoppten LSF-Jobs fort
bstp jobid	Setzt einen Job an die Spitze der eigenen Jobs

Tabelle 3.2: LSF Befehle

LSF Benutzeroberflächen

LSF bietet zwei Programme mit graphischer Benutzeroberfläche (GUI) für das X-Windows-System, was den Benutzern erlaubt, die einzelnen Queues und Jobs anzusehen und LSF-Kommandos per Mausklick abzusetzen, sowie die verbundenen Systeme und deren Auslastung graphisch darzustellen. In Abbildung 3.2 ist die graphische Benutzeroberfläche xlsbatch Main Window zu sehen, welche dem Benutzer die zum Cluster gehörenden Rechner, Queues, User-Jobs, sowie seinen Status anzeigt ([8], S.24).

Abbildung 3.2: xlsbatch [8]

Durch Doppelklick auf den Rechner, Queue bzw. auf die User-Jobs erhält man in einem neuen Fenster die Hosts, Queues bzw. Jobinformationen in Longformat (siehe Abbildung 3.3) ([8], S.25).

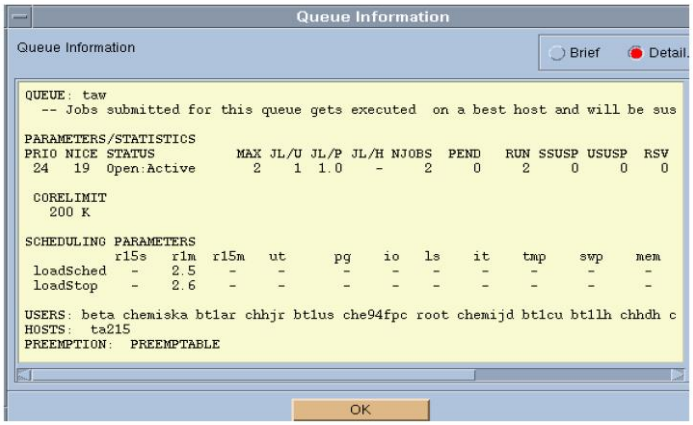

Abbildung 3.3: xlsbatch-Doppelklick [8]

3.2.2 OpenPBS, Portable Batch System

Einführung

Das Portable Batch System (PBS) ist ein generelles und verteiltes Jobmanagement-System, das in den Jahren 1993 bis 1997 für die National Aeronautics and Space Administration (NASA) entwickelt wurde. Das Portable Batch System (PBS) existiert in zwei Versionen. OpenPBS ist die frei verfügbare OpenSource Version des Portable Batch Systems und PBS Professional ist die kommerzielle Version des PBS, die Altair Engineering seit dem Jahr 2003 betreut. PBS ist plattformunabhängig und ermöglicht die optimale Nutzung der Ressourcen eines heterogenen Unix, Linux und Windows Netzwerkes. Allerdings ist die Funktionalität von OpenPBS im Vergleich zu PBS Professional stark eingeschränkt und für einen Produktionsbetrieb daher nur bedingt nutzbar. Darüberhinaus bietet Altair nur technischen Support für die Professional Variante an. OpenPBS bietet keine Unterstützung für Windows. Außerdem bietet OpenPBS keine Möglichkeit, Ressourcen für einen festzulegenden Zeitpunkt zu reservieren oder einzelne Ressourcen exklusiv bestimmten Benutzern oder Benutzergruppen zuzuordnen ([13], S.47-S.48). Desweiteren unterstützt OpenPBS kein preemptives Schedulingverfahren, Checkpointing , Job Migration, Job Arrays und Ausfallsicherheit des Servers [16]. Die Arbeitsweise von PBS Professional und OpenPBS funktioniert wie beim LSF. Die Benutzer übergeben ihre Rechenaufgabe (Jobs) interaktiv oder mit einem Shell-Skript dem Server. Dieser fügt den Job an einen Queue, der für diesen Job geeignet ist. PBS übernimmt ab hier die Verwaltung und die Verarbeitung des Jobs. In Tabelle 3.3 werden PBS Queues dargestellt. PBS Queues haben die gleiche Eigenschaft wie LSF Queues und unterscheiden sich somit in gleicher Weise von einfachen Queues. Queues unter PBS Professional können bei laufendem Betrieb konfiguriert, geöffnet, geschlosssen , aktiviert oder deaktiviert werden ([15], S.103). OpenPBS bietet diesen Dienst nicht an [16].

Parameter	test	shared	large	excl
QUEUE NAME	test	shared	large	excl
queue type	Execution	Execution	Execution	Execution
PRIORITY	120	200	50	70
USERS	usersA	usersB	usersA	usersC

Tabelle 3.3: PBS Queues

Desweiterin bietet PBS eine Vielzahl von Befehlen, die in Tabelle 3.4 dargestellt sind, um einen Job, der in den Queues angemeldet ist, zu beobachten oder zu kontrollieren. Es ist möglich die Ausgaben sowie die Historie von laufenden Jobs zur beobachten.

PBS Befehle	Bedeutung der Befehle
qsub	Übergibt einen Job in eine PBS-Queues
qdel	Entfernt den Jobskript aus dem PBS
qstat	Zeigt Job, Queue und Server Status
qstat -B	Zeigt Server Status
qstat -Q	Zeigt alle Queues
pbsnodes -a	Zeigt alle Knoten des Clusters an
qstat -a	Zeigt eine Liste aller laufenden und wartenden Jobs
qstat -au<user>	Zeigt eine Liste alle Jobs des Nutzers <user>
qout <jobid>	Ausgabe laufender Jobs anzeigen

Tabelle 3.4: PBS Befehle

PBS Benutzeroberflächen

Das Portable Batch System bietet zwei Programme mit graphischer Benutzeroberfläche (GUI) für das X-Windows-System an, welches den Benutzern erlaubt, die einzelnen Queues und Jobs anzusehen und PBS-Kommandos per Mausklick abzusetzen, sowie die verbundenen Systeme und deren Auslastung graphisch darzustellen. In Abbildung 3.4 ist die graphische Benutzeroberfläche xpbsmon zu sehen, welche die verbundenen Systeme und deren Auslastung graphisch darstellt.

Die xpbsmon Oberfläche besteht aus drei Spalten. In der ersten Spalte werden für den Administrator mehrere Buttons zur Verfügung gestellt, welche den Administrator bei der Konfiguration des Systemes unterstützen. In der zweiten Spalte werden die einzelnen Rechner durch Quadrate dargestellt. Der Zustand eines Rechners wird durch die verschiedenen Farben charakterisiert. Durch einen Doppelklick auf ein Quadrat (Rechner) erhält man in einem neuen Fenster (siehe Abbildung 3.5) die Rechnerinformationen in Langformat. Die Zustände, in denen sich ein Rechner befinden kann, werden durch Farben dargestellt. Die Farbe Grün bedeutet, dass der Rechner zur Zeit frei ist und keine Berechnung verarbeitet. Die Farbe Rot bedeutet, dass der Rechner nicht zur Verfügung steht. Die Farbe Braun bedeutet, dass der Rechner rechnet, wobei noch mehrere CPU´s zur Verfügung stehen. Die Farbe Türkis bedeutet, dass zwei oder mehrere Berechnungen auf einem Rechner laufen. In der dritten

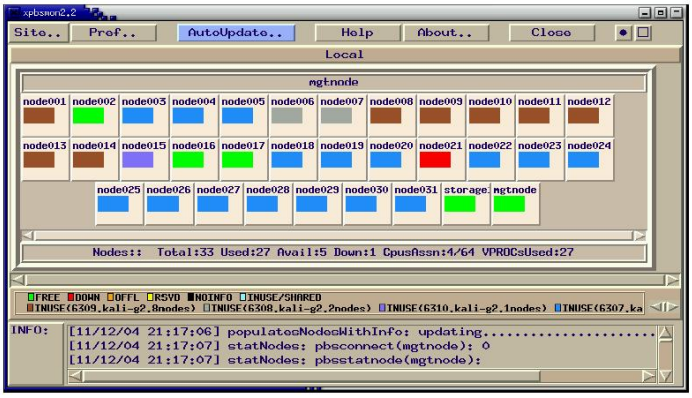

Abbildung 3.4: xpbsmon [13]

Spalte werden aktuelle Informationen angezeigt, wie zum Beispiel Fehlermeldungen ([14], S.326).

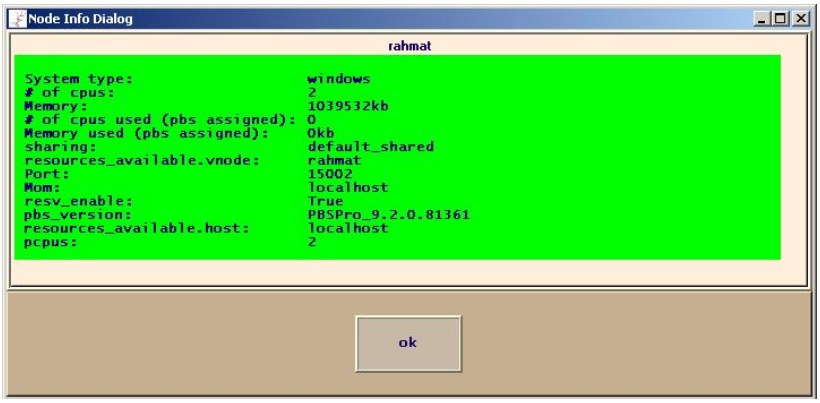

Abbildung 3.5: xpbsmon-Doppelklick

PBS stellt mittels xpbs (siehe Abbildung 3.6) eine Benutzeroberfläche zur Verfügung, welche den Benutzern erlaubt, die einzelnen Queues und Jobs anzusehen und PBS-Kommandos per Mausklick abzusetzen.

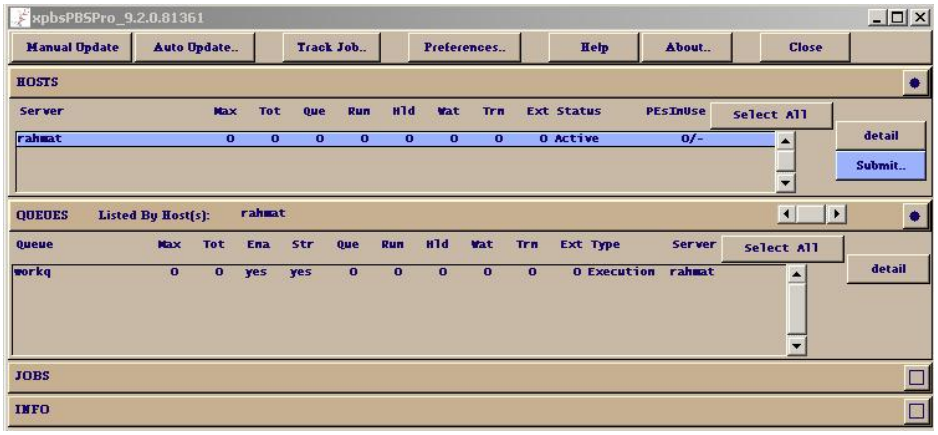

Abbildung 3.6: xpbs

3.3 Bewertung von PBS Professional, OpenPBS und LSF

Wie im Kapitel 3.1 besprochen, werden Jobmanagement-Systeme durch Queues, Scheduler, Ressourcen-Management, Job-Management , Dienste und Sicherheit, welche das System anbietet, charakterisiert. Anhand von Tabellen werden die jeweiligen Jobmanagement-Systeme verglichen und bewertet.

In Tabelle 3.5 werden Informationen über Hersteller und Webseiten der jeweiligen Jobmanagement-Systemen gegeben. Desweiteren wird unterschieden zwischen Plattform, Dokumentation und technischem Support.

	LSF	PBS	OpenPBS
Hersteller	Platform computing	Altair Engeineering	Altair Engeineering
Webseiten	www.platform.com	www.pbspro.com	www.openpbs.org
Betriebssysteme	Linux Solaris Windows	Linux Solaris Windows	Linux Solaris
Dokumentation	Ja	Ja	teilweise
Technischer support	Ja	Ja	Nein

Tabelle 3.5: Jobmanagement-System [16]

In Tabelle 3.6 werden Scheduler-Strategien und Dienste des Schedulers unterschieden.

	LSF	PBS	OpenPBS
Schedule-Strategien	Fairshare, Preemption FCFC, exklusiv, RR	Fairshare, Preemption FCFC, exklusiv, RR	Fairshare FCFC, exklusiv
Backfill	Ja	Ja	Nein
Load Balancing	Ja	Ja	Ja

Tabelle 3.6: Scheduler [16]

In Tabelle 3.7 wird unterschieden, in welchen Zuständen sich die Queues bei laufendem Betrieb befinden können und mit welchen wichtigen Konfigurationsattributen sie konfiguriert werden können.

	LSF	PBS	OpenPBS
Bei laufende Betrieb Konfigurien, öffnen, Schliessen, aktivieren und deaktivieren	Ja	Ja	Nein
Priorität	Ja	Ja	Ja
Interaktive Jobs	Ja	Ja	Ja

Tabelle 3.7: Queue [16]

In Tabelle 3.8 wird dargestellt, welche Sicherheits-Strategien das jeweilige Jobmanagement-System anbietet.

	LSF	PBS	OpenPBS
Authentifizierung	Ja	Ja	Ja
Autorisierung	Ja	Ja	Ja
Kerberos	Ja	Ja	Ja
Ausfallsicherheit des Servers	Ja	Ja	Nein

Tabelle 3.8: Sicherheit [16]

In Tabelle 3.9 wird dargestellt, welche Dienste die jeweiligen Jobmanagement-Systeme für einen laufenden oder wartenden Job (Berechnung) anbieten.

	LSF	PBS	OpenPBS
Checkpointing, Job Migration und Job-Array	Ja	Ja	Nein
Job Calendar	Ja	Ja	Ja
Parallele Jobs	Ja	Ja	Ja
Job-Abhängigkeit	Ja	Ja	Ja
Suspend/Resume	Ja	Ja	Nein

Tabelle 3.9: Job Management [16]

In Tabelle 3.10 wird gezeigt, welche Möglichkeiten die Benutzer haben, um Ressourcen in Anspruch zu nehmen.

	LSF	PBS	OpenPBS
Beliebige Ressourcen definierbar	Ja	Ja	Nein
Ressourcen Reservierung	Ja	Ja	Nein
Dynamischer CPU-Request	Ja	Ja	Nein
Dynamische Ressourcen	Ja	Ja	Nein

Tabelle 3.10: Ressourcen Management [16]

In Tabelle 3.11 wird gezeigt, welche weiteren Dienste die jeweiligen Jobmanagement-Systeme anbieten.

	LSF	PBS	OpenPBS
GUI	Ja	Ja	Ja
API	Ja	Ja	Ja
LS-Dyna und HP-MPI	Ja	Ja	Nein
Unterstützung für heterogene Cluster	Ja	Ja	Ja
Multicluster	Ja	Ja	Nein

Tabelle 3.11: Weitere Dienste [16]

3.4 Auswertung der Auswahl eines Jobmanagement-Systems

Alle drei Jobmanagement-Systeme LSF, PBS Professional und OpenPBS entsprechen den Anforderungen eines Jobmanagement-Systems. Bezüglich der Erwartungen an die verschiedenen Dienste eines Jobmanagement-Systems gibt es jedoch einige Unterschiede. Betrachtet der Verfasser die Jobmanagement-Systeme ergeben sich beim OpenPBS einige Nachteile. Die Nachteile sind die fehlende Windows-Kompatibilität, sowie der fehlende technische Support durch den Hersteller (siehe Tabelle 5). Verglichen zu den LSF und PBS Systemen unterstützt das OpenPBS beim Scheduling kein preemptives Scheduling. Zudem wird kein Load Balancing unterstützt (siehe Tabelle 6). LSF sowie PBS Professional ermöglichen bei der Bearbeitung von Queues eine Vielzahl verschiedener Vorgänge, die durch das OpenPBS nicht unterstützt werden (siehe Tabelle 7). Auch bei Betrachtung der Sicherheit fällt auf, das OpenPBS im Vergleich zu den anderen Systemen keine Ausfallsicherheit des Servers gewährleistet (siehe Tabelle 8). Beim Job-Management kann das OpenPBS, Dienste wie Checkpointing, Job Migration und Job-Array nicht ausführen (siehe Tabelle 9). Bei der Betrachtung vom Ressourcen-Management fällt auf, dass das OpenPBS, verglichen mit den anderen beiden Systemen, über keine Möglichkeiten des Ressourcenverwendung verfügt (siehe Tabelle 10). OpenPBS ist ein Open-Source-Code und somit die kostengünstigere Variante. Wenn die oben besprochenen Punkte nicht relevant sind, können mittels OpenPBS relativ große Cluster konstruiert werden. Durch die Kriterien die in Kapitel 3.1 vom Verfasser aufgelistet wurden, kann kein Unterschied zwischen PBS Professional und LSF erkannt werden. Der Verfasser entscheidet sich für das PBS, da bereits Lizenzen für diese Software zur Verfügung stehen.

4 Analyse des Ressourcen-Bedarfs von LS-Dyna Berechnungen

Wie in Kapitel 3.1 diskutiert wurde, dienen Jobmanagement-Systeme (PBS) der Abstimmung zwischen den Anforderungen der Anwender nach IT-Ressourcen mit den real verfügbaren Möglichkeiten. Dies soll dem Unternehmen helfen, eine optimale Ressourcenverteilung zu erzielen. Was das PBS als Dienst nicht anbietet, ist die Vorhersage von CPU's für einen LS-Dyna Job, um dessen optimale Durchlaufzeit zu realisieren. Desweiteren muss ein Benutzer die Shell-Programmierung und ein Verständnis für HP-MPI und LS-Dyna aufweisen, um Jobs starten zu können. Aufgrund der hier beschriebenen Nachteile, ist es erforderlich, eine Zuteilungssoftware zu entwickeln. Weiterhin benötigt das PBS die Anzahl der CPU's und Rechner als Eingangsparameter, um einen LS-Dyna Job durchführen zu können. Die hier beschriebenen Parameter werden mit Hilfe der entwickelten Zuteilungssoftware an das PBS weitergeleitet. Wie in Kapitels 4.2.1, Fall eins dargestellt, werden die Jobs berechnet, ohne dass diese sich den selben Rechner teilen. Somit ist es nicht notwendig, den Speicherbedarf für einen LS-Dyna-Job an das PBS weiterzuleiten, weil die Jobs nicht um den Speicher eines Rechners konkurrieren. Anhand der LS-Dyna Jobs, die dem Verfasser zur Verfügung stehen (siehe Abbildung 2.2), wird der vom PBS nicht unterstützte Dienst entwickelt (siehe Kapitel 5). Weiterhin wird der Inhalt des Kapitels 4.1.1 benutzt, um die Theorien in Kapitel 4.1.2 und 4.2 darzulegen.

4.1 Einfluß der CPU

In diesem Kapitel wird anhand der Ausführungszeiten der Berechnungen (siehe Kapitel 4.1.1) und der verschiedenen Scheduling-Algorithmen (siehe Kapitel 2.2) dargestellt, dass die optimale Auslastung des Clusters nicht nur von Scheduling-Algorithmen abhängig ist, sondern von der Anzahl der CPU´s die ein Job (Berechnung) in Anspruch nimmt. Hierbei werden zwei Szenarien unterschieden, die die gleichen Jobs mit unterschiedlicher CPU-Anzahl ausführen. Die hier verwendeten Files, werden anhand der Kapitel 4.2.1, Fall eins, berechnet und bewertet.

4.1.1 LS-Dyna-Inputfile

Bewertung von test_10ms

In Abbildung 4.1 sind die unterschiedlichen Ausführungszeiten des Jobs test_10ms dargestellt, die auf einer Veränderung der CPU-Anzahl beruhen. Es ist zu erkennen, dass es bei einer Variation der CPU-Anzahl bis 5 zu einer drastischen Reduktion der Rechenzeit kommt. Zwischen 5 und 40 CPU's kommt es zu einer linearen Reduktion der Rechenzeit, die aber deutlich geringer ausfällt als im vorherigen Abschnitt. Bei einer weiteren Erhöhung der Anzahl auf über 40 CPU's kommt es zu keiner weiteren Reduktion der Rechenzeit.

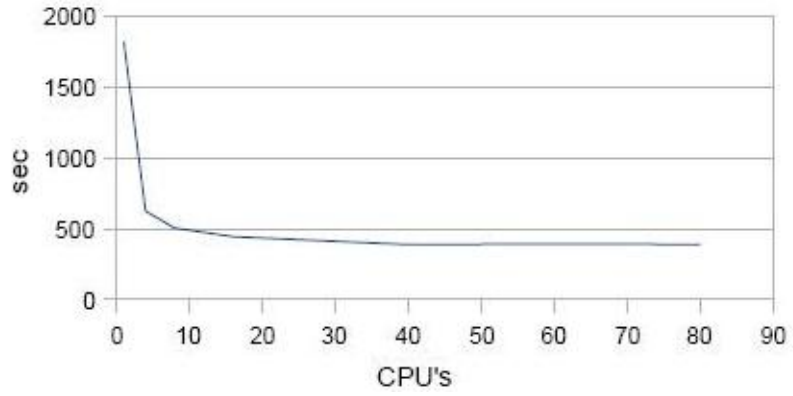

Abbildung 4.1: Ausführungszeiten von test_10ms

Abbildung 4.2 stellt die Effizienz in Abhängigkeit von der Anzahl der CPU´s da. Es wird auch hier deutlich, dass bei steigendem Ressourceninput die Effizienz abnimmt.

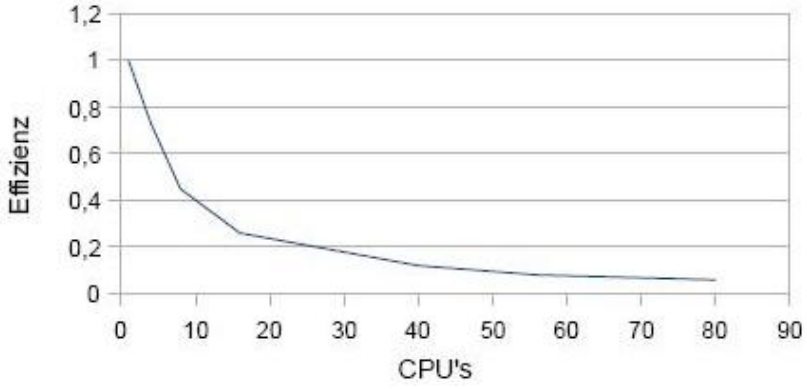

Abbildung 4.2: Effizienz von test_10ms

In Abbildung 4.3 ist zu erkennen, dass mit steigender Anzahl an CPU´s die Speedup-Kurve immer weiter vom Ideal divergiert.

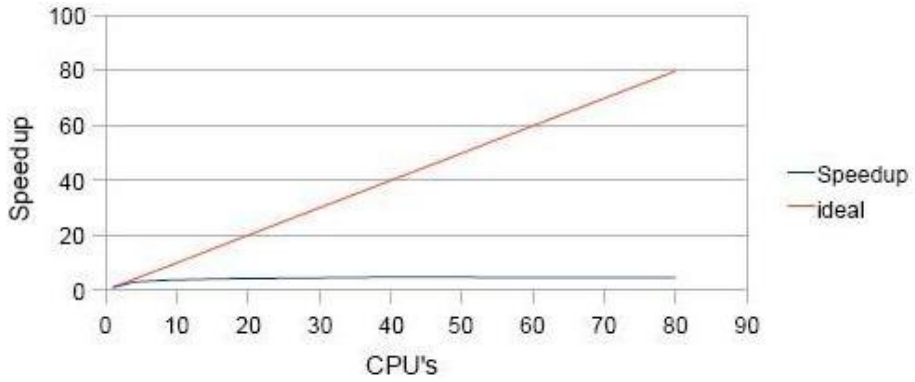

Abbildung 4.3: Speedup von test_10ms

Bewertung von mat_spring.belted.dummy

Mat_spring.belted.dummy ist ein sehr kleiner Job (siehe Ausführungszeiten) und somit für das Parallelisieren nicht geeignet. In Abbildung 4.4 ist zu erkennen, dass sich die Ausführungszeiten mit steigender Anzahl an CPU´s verschlechtern.

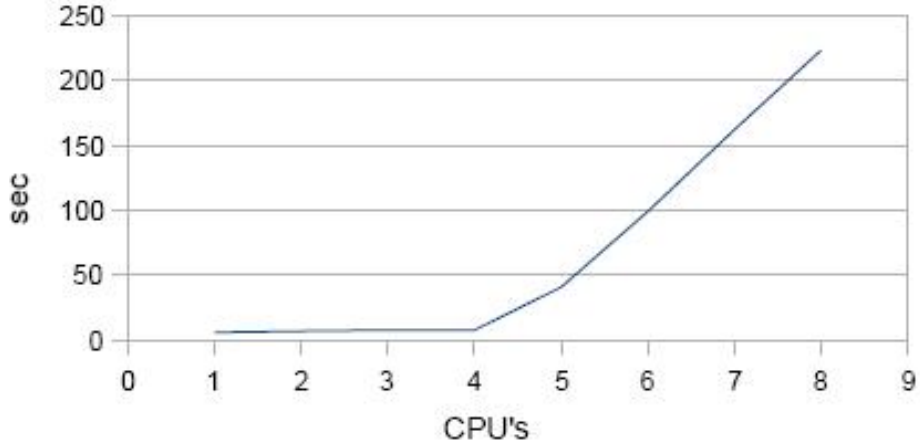

Abbildung 4.4: Ausführungszeiten von mat_spring.belted.dummy

Wie in Abbildung 4.4 dargestellt, steigt die Ausführungszeit bei zunehmender CPU-Anzahl. Daraus folgt, dass die Speedup-Kurve gegen null konvergiert und somit auch die Effizienz (siehe Abbildung 4.5 und 4.6).

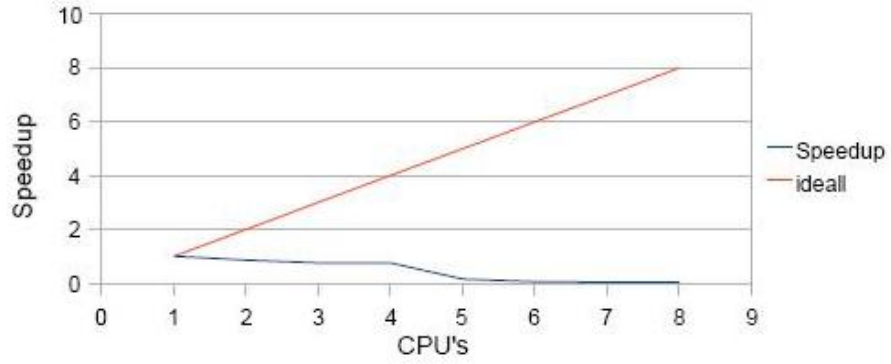

Abbildung 4.5: Speedup von mat_spring.belted.dummy

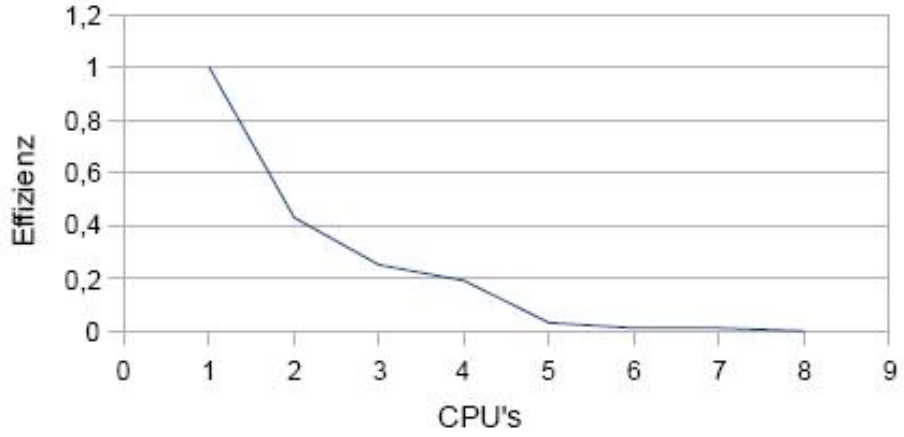

Abbildung 4.6: Effizienz von mat_spring.belted.dummy

Bewertung von Audi-A2_Vorderwagen

In Abbildung 4.7 sind die Ausführungszeiten in Abhängigkeit von der Anzahl an CPU´s dargestellt. Es ist ersichtlich, dass die Ausführungszeit, die aus der Benutzung von 16 CPU's resultiert, zu einer drastischen Reduktion der Rechenzeit führt. Weiterhin ist durch Abbildung 4.9 zu erkennen, dass ab 16 CPU's die Speedup-Kurve immer weiter vom Ideal abweicht. Dies hat zur Folge, dass die Effizienz-Kurve gegen Null konvergiert (siehe Abbildung 4.8).

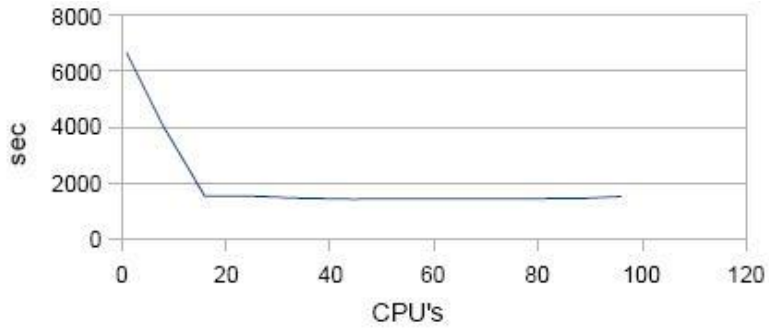

Abbildung 4.7: Ausführungszeiten von Audi-A2_Vorderwagen

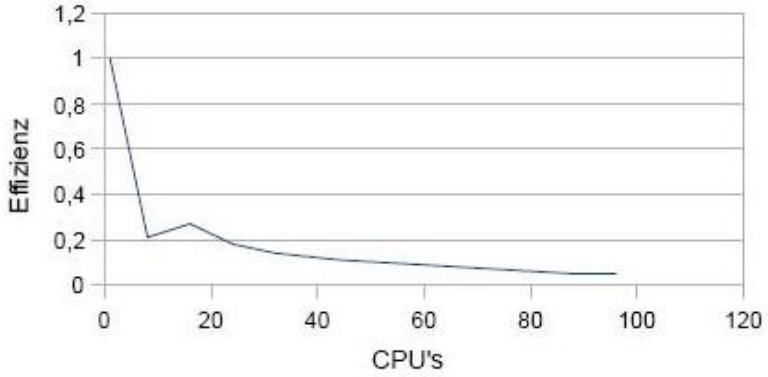

Abbildung 4.8: Effizienz von Audi-A2_Vorderwagen

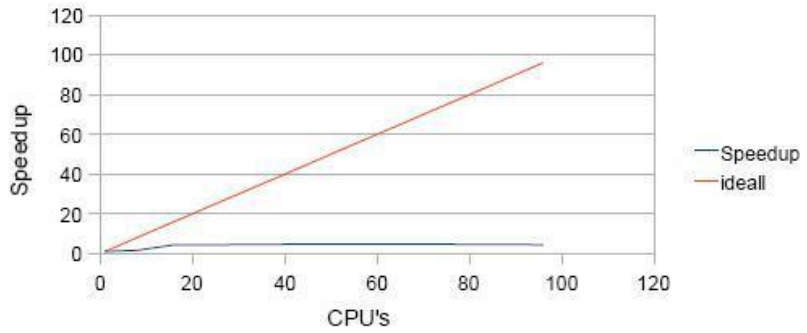

Abbildung 4.9: Speedup von Audi-A2_Vorderwagen

4.1.2 Szenarien

Szenario eins

In Tabelle 4.1 sind drei Prozesse (Jobs) zu sehen, deren Ausführungszeiten untersucht wurden sind (siehe Kapitel 4.1.1). Diese Jobs werden auf einem Cluster mit 96 CPU´s berechnet (siehe Kapitel 2.1). Auf dem Cluster ist PBS installiert (siehe Kapitel 3). Hierbei werden die unterschiedlichen Scheduling-Algorithmen von PBS untersucht. Desweiteren ist die CPU-Anzahl, Ankunftszeit und die Bearbeitungszeit eines Jobs in der Tabelle 4.1 dargestellt.

Prozess	CPU's	Ankunft	Bearbeitungszeit	Scheduling-Zeit
A = (mat_spring.belted_dummy)	8	0 sec	223 sec	2039 sec = 20 sec 223 sec = 2 sec
B = (test_10ms)	56	1 sec	391 sec	2039 sec = 20 sec 391 s = 4 s
C = (Audi-A2_Vorderwagen)	44	2 sec	1425 sec	2039 sec = 20 sec 1425 sec = 14 sec

Tabelle 4.1: Bearbeitungszeiten des Szenarios eins

Untersuchung der Scheduling-Algorithmen

Die Farben verdeutlichen, dass ein Job in diesem Moment ausgeführt wird.

FCFS

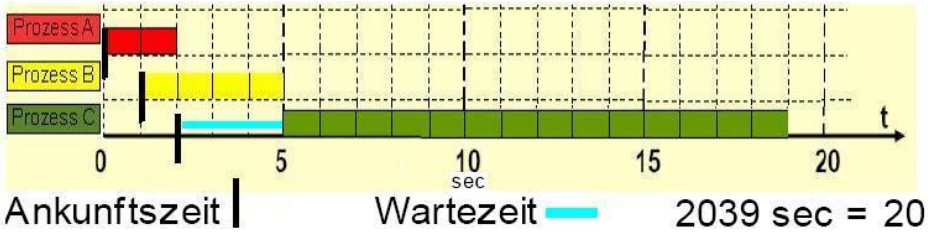

Abbildung 4.10: First-Come First-Served

Die Ausführung des FCFS Scheduling-Algorithmus (siehe Abbildung 4.10) wird anhand der Abbildung 4.11 verdeutlicht, in welcher die Ressourcenauslastung, also freie oder arbeitende CPU´s, sowie die Zustände der Prozesse, wartende und ausführende Prozesse, dargestellt sind.

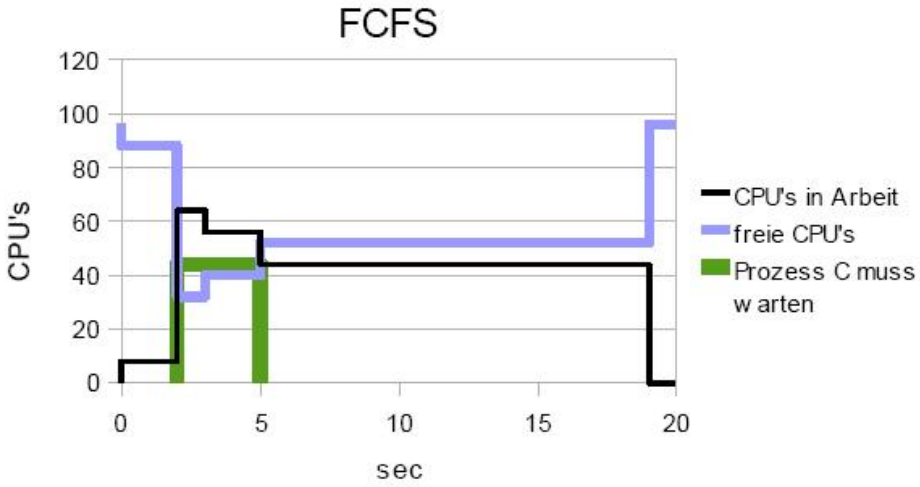

Abbildung 4.11: First-Come First-Served Cluster

Durch die steigende und konstante Flanke des grünen Graphen ist zu erkennen, dass der Prozess C warten muss. Durch den Mangel an verfügbaren CPU's ist eine sofortige Bearbeitung zum Zeitpunkt t = 2 sec nicht möglich, wie durch den lila Graphen dargestellt. Bei fallender Flanke des grünen Graphen, werden ihm seine benötigten Ressourcen zugeteilt.

SJF

Anhand der Zeichnung wird deutlich, dass SJF (siehe Abbildung 4.12) eine identische Reihenfolge der Prozessabwicklung liefert wie FCFS (siehe Abbildung 4.10). Somit ist auch Abbildung 4.13 mit dem vorherigen Ergebnissen (siehe Abbildung 4.11) identisch.

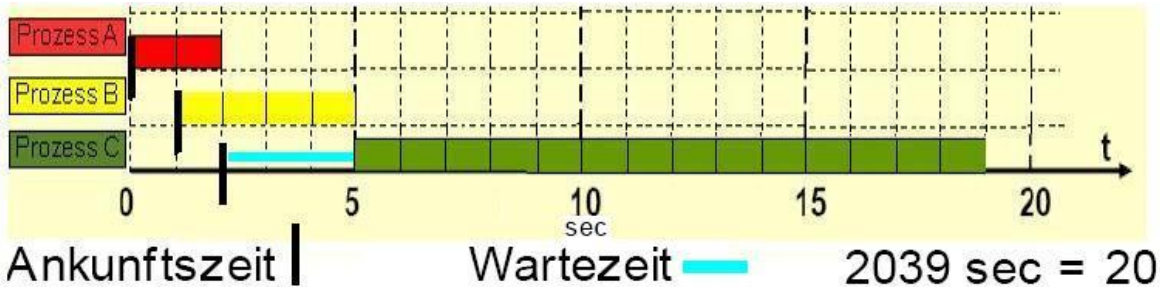

Abbildung 4.12: Shortest Job First

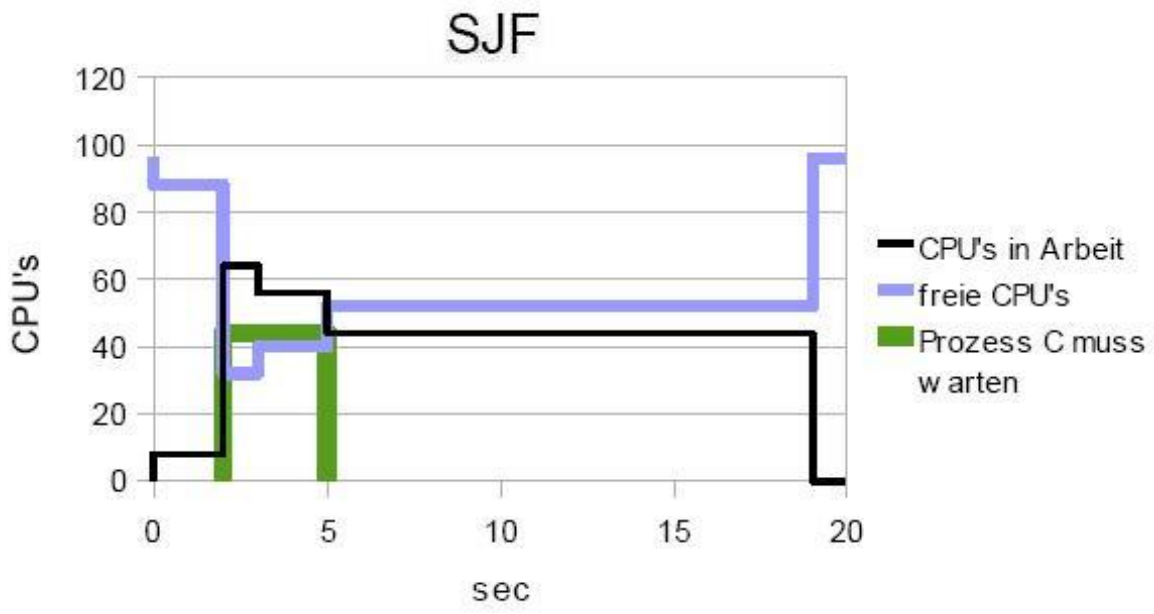

Abbildung 4.13: Shortest Job First Cluster

RR q = 1

Durch Abbildung 4.14 wird deutlich, dass der Scheduling-Algorithmus RR mit Quantum = 1 viele Prozesswechsel bewältigen muss. Hierbei ensteht ein Nachteil. Die Durchlaufzeit von Prozess B liegt bei t = 714 sec. In den beiden besprochenen Scheduling-Algorithmen FCFS und SJF lag die Durchlaufzeit von Prozess B jedoch bei t = 391 sec.

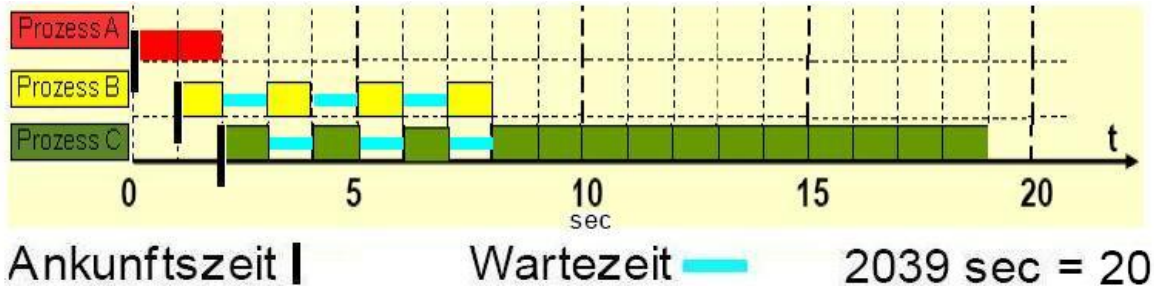

Abbildung 4.14: Round-Robin q=1

Durch Abbildung 4.15 ist zu sehen, dass der RR mit Quantum = 1 ein ineffizienter Algorithmus für diese Jobs darstellt, da hier Prozess B und C auf die Bearbeitung durch den Cluster warten müssen, welches durch die steigende und konstante Flanke des gelben und grünen Graphen dargestellt wird. Bei fallender Flanke des grünen bzw. gelben Graphens, erhalten die Prozesse ihre benötigten Ressourcen.

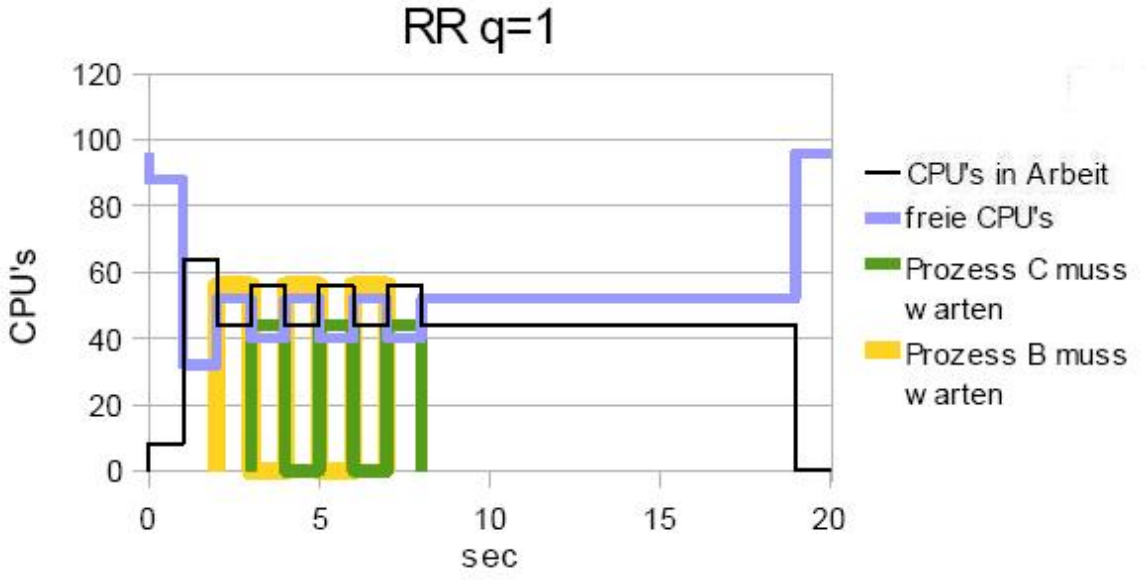

Abbildung 4.15: Round-Robin q=1 Cluster

RR q = 3

Der Scheduling-Algorithms RR mit Quantum = 3 (siehe Abbildung 4.16) verhält sich wie RR mit Quantum = 1, mit dem Unterschied, dass RR mit Quantum = 3 weniger Prozesswechsel durchführt als RR mit Quantum = 1 (siehe Abbildung 4.14).

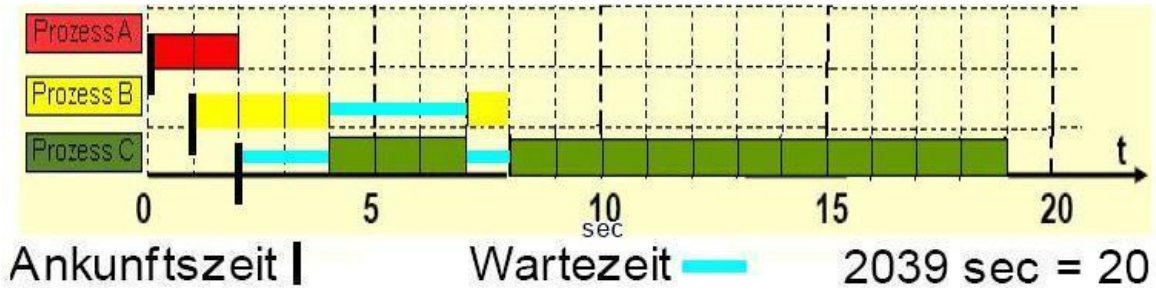

Abbildung 4.16: Round-Robin q=3

Durch Abbildung 4.17 ist zu sehen, dass durch diesen Scheduling-Algorithmus auch Prozesse warten müssen, da nicht ausreichend Ressourcen zur Verfügung stehen.

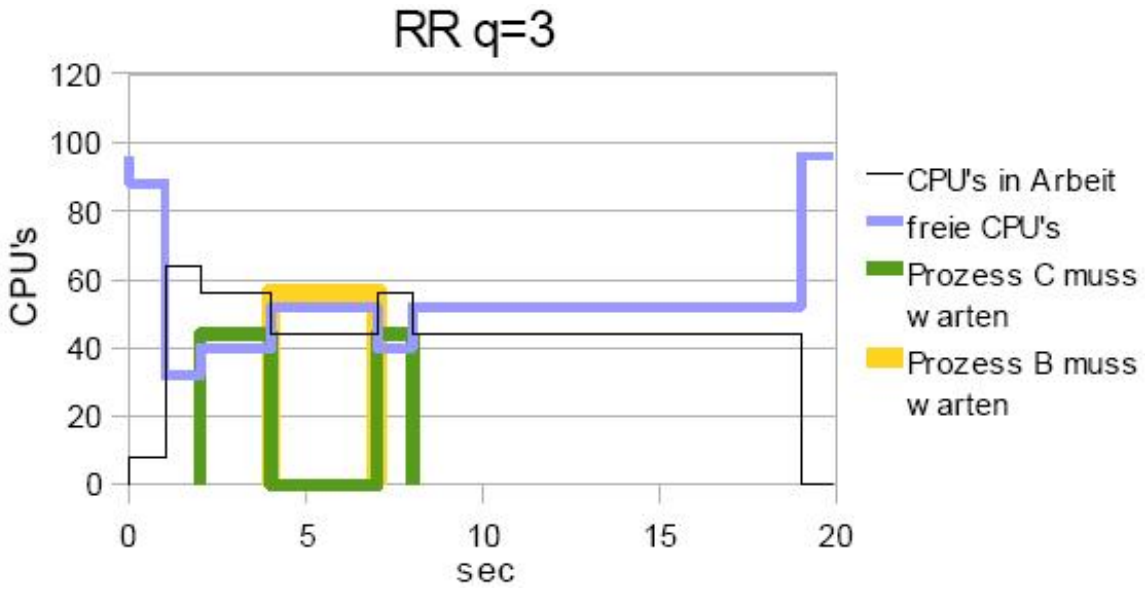

Abbildung 4.17: Round-Robin q=3 Cluster

RR q = 4

Der Scheduling-Algorithms RR mit Quantum = 4 (siehe Abbildung 4.18-4.19) verhält sich wie FCFS und SJF (siehe Abbildung 4.10-4.11 und 4.12-4.13), welche oben besprochen wurden.

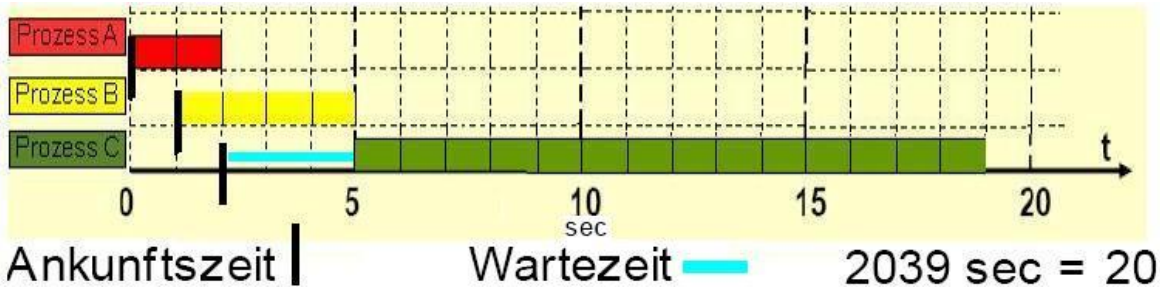

Abbildung 4.18: Round-Robin q=4

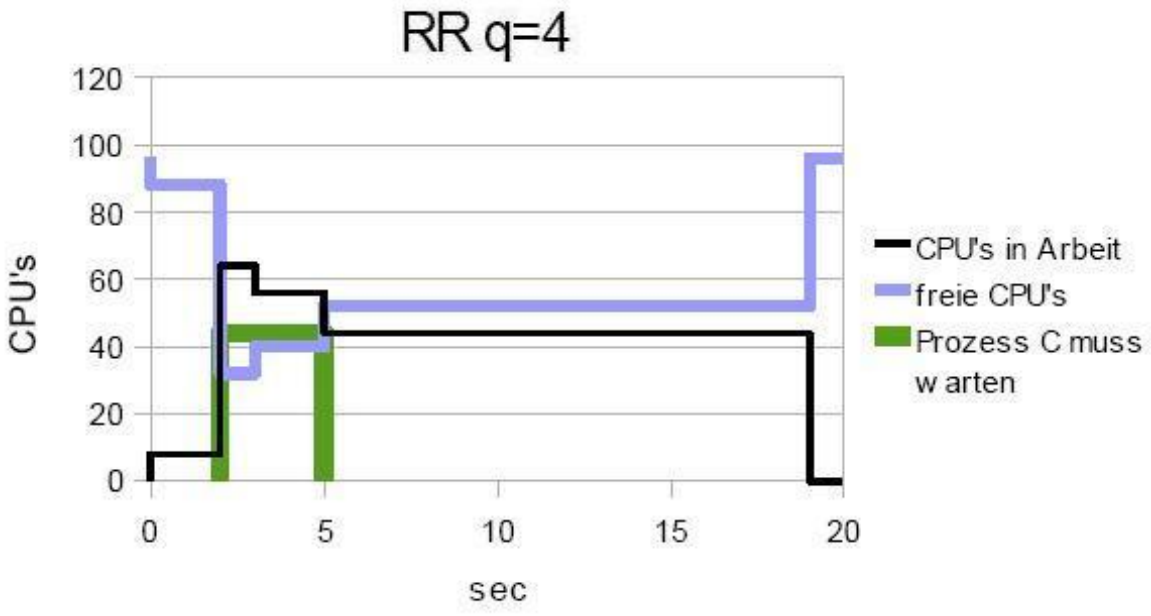

Abbildung 4.19: Round-Robin q=4 Cluster

Szenario zwei

Hier wird für die gleichen Jobs eine unterschiedliche Anzahl von CPU´s verwendet, wodurch sich auch die Bearbeitungzeit der jeweiligen Jobs verändert. Die Ankunftszeiten bleiben gleich (siehe Tabelle 4.2).

Prozess	CPU's	Ankunft	Bearbeitungszeit	Scheduling-Zeit
A = (mat_spring.belted_dummy)	1	0 sec	6 sec	1753 sec = 20 sec 6 sec = 1 sec
B = (test_10ms)	7	1 sec	630 sec	1753 sec = 20 sec 630 sec = 7 sec
C = (Audi-A2_Vorderwagen)	8	2 sec	1117 sec	1753 sec = 20 sec 1117 sec = 12 sec

Tabelle 4.2: Bearbeitungszeiten des Szenarios zwei

Untersuchung der Scheduling-Algorithmen

FCFS, SJF, RR q = 1, 3, 4

Es ist ersichtlich, dass bei allen Scheduling-Algorithmen, welche in Abbildung 4.20 dargestellt sind, die Ausführungszeiten kleiner werden als im ersten Szenario.
Die Variation des Quantums beim RR (q = 1, 3, 4) (siehe Abbildung 4.20) hat keinen Einfluss auf die Bearbeitung der Jobs. Somit führen alle drei Quanten zum selben Resultat. Auch das SJF zeigt das gleiche Verhalten wie die beiden oberen Schelduling-Algorithmen.

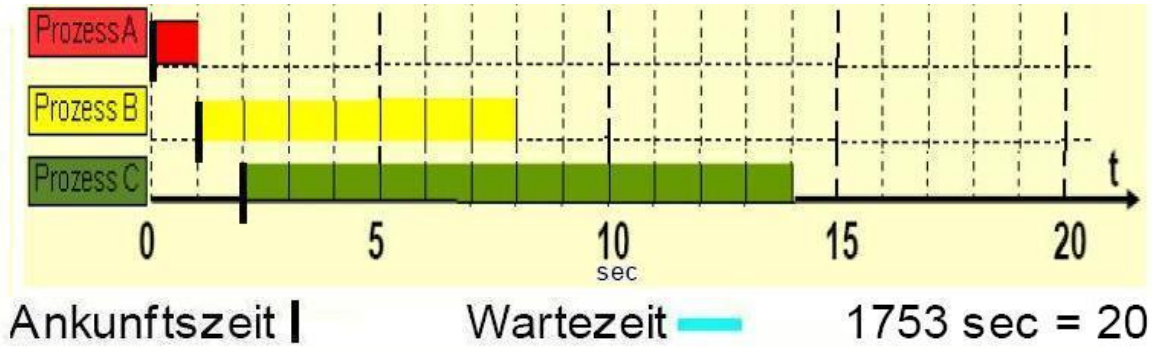

Abbildung 4.20: FCFS, SJF, RR q=1, 3, 4

Durch Abbildung 4.21 wird deutlich, dass kein Prozess mehr auf Ressourcen warten muss. Desweiteren ist zu sehen, dass für die Bearbeitung der drei Jobs durchschnittlich 10 CPU's benutzt werden. Für die Bearbeitung weiterer Jobs stehen durchschnittlich 86 CPU's zur Verfügung.

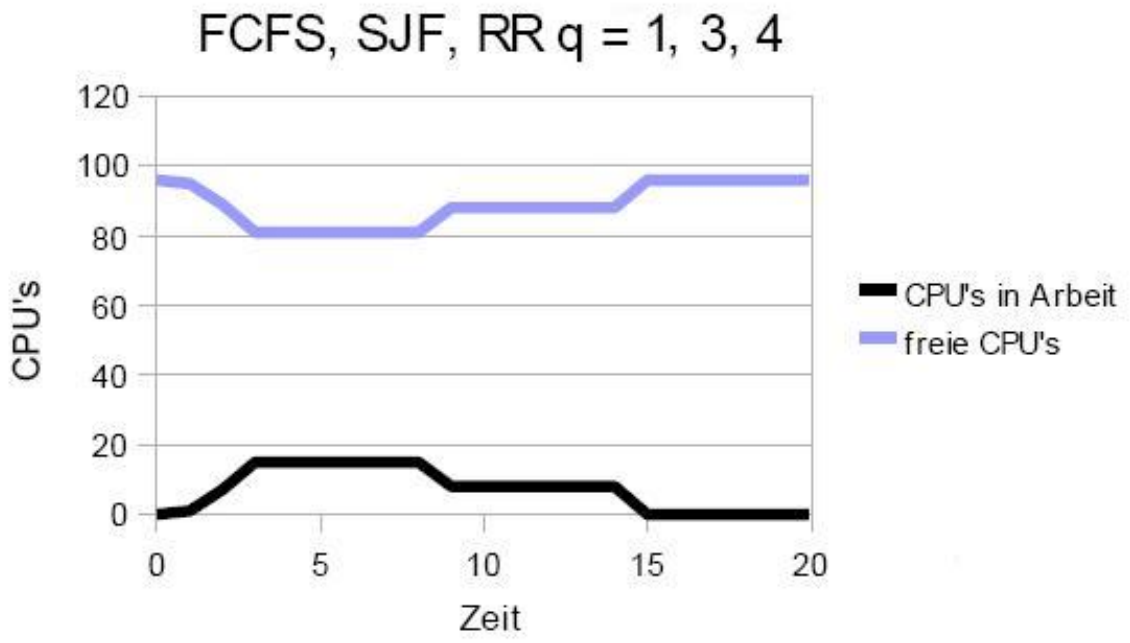

Abbildung 4.21: FCFS, SJF, RR q=1=3=4 Cluster

4.1.3 Bewertung der Szenarien

Bewertung von Senario eins

In Tabelle 4.3 sind die Durchschnittszeiten von Szenrio eins zu sehen. Die Jobs werden mit durchschnittlich 43 CPU´s berechnet. Die Anzahl der verfügbaren CPU´s beträgt durchschnittlich 53. Die Summe der in Anspruch genommenen CPU´s beträgt mehr als 96 CPU´s, weshalb die Jobs gezwungen sind auf Ressourcen (CPU´s) zu warten. Diese Werte gelten für alle Scheduling-Algorithmen. Bei einem preemtiven Scheduling-Algorithmus, wie beim RR unter der Verwendung von Quanten eins oder drei, sind Prozesswechsel zu erkennen. Weiterhin ist aus der Tabelle ersichtlich, dass die durchschnittliche Wartezeit von FCFS, SJF und RR q = 4 identisch ist. RR q = 1 und RR q = 3 haben eine doppelt so große durchschnittliche Wartezeit. Auch bei der durchschnittliche Durchlaufzeit ist zu beobachten, dass diese für FCFS, SJF und RR q = 4 geringer ausfällt als für RR q = 1 und RR q = 3.

	FCFS	SJF	RR q = 1	RR q = 3	RR q = 4
Prozesswechsel	0	0	7	3	0
durchschnittliche Wartezeit	102 sec	102 sec	204 sec	204 sec	102 sec
durchschnittliche Durchlaufzeit	782 sec	782 sec	890 sec	890 sec	782 sec
durchschnittliche freie CPU's	53	53	53	53	53
durchschnittliche verbrauchte CPU's	43	43	43	43	43

Tabelle 4.3: Bewertung des Senarios eins

Bewertung von Senario zwei

In Szenario zwei werden die Jobs mit durchschnittlich 10 CPU´s berechnet (siehe Tabelle 4.4). Die Anzahl der verfügbaren CPU's beträgt durchschnittlich 86. Die Summe der für die drei Jobs in Anspruch genommenen CPU's beträgt 16. Dadurch wird erreicht, dass die Jobs nicht mehr auf Ressourcen warten müssen und direkt bearbeitet werden. Bei allen Scheduling-Algorithmen ist die durchschnittliche Durchlaufzeit identisch und beträgt t=584 sec. Durch diese Art von Ressourcenverteilung sind die Ziele der Stapelverarbeitungssysteme, die besagen, minimiere die Zeit vom Start bis zur Beendigung eines Prozesses und maximiere die Jobs pro Stunde, erfüllt (siehe Kapitel 2.2.2).

	FCFS	SJF	RR q = 1	RR q = 3	RR q = 4
Prozesswechsel	0	0	0	0	0
durchschnittliche Wartezeit	0 sec	0 sec	0 sec	0 s	0 sec
durchschnittliche Durchlaufzeit	584 sec	584 sec	584 sec	584 sec	584 sec
durchschnittliche freie CPU's	86	86	86	86	86
durchschnittliche verbrauchte CPU's	10	10	10	10	10

Tabelle 4.4: Bewertung des Senarios zwei

4.2 Einfluß von Shared-Memory und Message-Passing-Multicomputer Systemen

Bezüglich des Vergleiches zwischen dem **Shared-Memory** und dem **Message-Passing-Multicomputer** System aus Kapitel 2.3.5, wird der LS-Dyna Job in zwei Szenarien behandelt. Im ersten Szenario wird ein Job in drei Fällen unterschieden. In allen drei Fällen ist die Anzahl der CPU´s identisch und die Jobs laufen allein auf den Rechnern des Clusters. Wenn beispielsweise ein Job mit 32 CPU´s berechnet wird, dann werden im ersten Fall alle CPU´s eines Rechners belegt. Wenn also 32 CPU´s benötigt werden, werden 4 Rechner mit jeweils 8 CPU´s belegt. Im zweiten Fall werden 8 Rechner mit 4 CPU´s belegt und im dritten Fall werden 32 CPU´s auf 11 Rechner verteilt. Anhand des zweiten Szenarios wird mit Hilfe eines Beispieles dargestellt, wie sich die Ausführungszeiten von Jobs unter gleicher Anzahl an CPU´s ändern, wenn sich mehrere Jobs die Rechner teilen.

4.2.1 Isolierte Nutzung der Rechner

Szenario eins

Bewertung von test_10ms

Es ist deutlich zu sehen, dass die Ausführungszeit im zweiten und dritten Fall geringer ausfällt, als im ersten Fall (siehe Abbildung 4.22). Daraus lässt sich folgern, dass der Speedup der CPU´s im zweiten und dritten Fall fast um das doppelte größer ist als im ersten Fall. Die Anzahl der CPU's, die im ersten Fall um den BUS konkurrieren, beträgt 8. Dies führt dazu, dass die CPU's im ersten Fall länger warten müssen als im zweiten und dritten Fall.

Erster Fall: 2 Rechner mit je 8 CPU‘s
Zweiter Fall: 8 Rechner mit je 2 CPU´s
Dritter Fall: 5 Rechner mit je 2 CPU´s und 6 Rechner mit je 1 CPU

Abbildung 4.22: test_10ms in drei Fällen

Bewertung von Audi-A2_Vorderwagen

Die Ausführungszeiten von Audi-A2_Vorderwagen werden geringer (siehe Abbildung 4.23), je mehr Rechner zum Einsatz kommen. Dabei wird die verwendete Anzahl an CPU's auf diese Rechner verteilt. Somit steigt der Speedup der CPU's.

Erster Fall: 2 Rechner mit je 8 CPU´s
Zweiter Fall: 8 Rechner mit je 2 CPU´s
Dritter Fall: 5 Rechner mit je 2 CPU´s und 6 Rechner mit je 1 CPU

Abbildung 4.23: Audi-A2_Vorderwagen in drei Fällen

Bewertung von test_240ms

Wie erwartet, fällt die Ausführungszeit im zweiten Fall geringer aus als im ersten Fall (siehe Abbildung 4.24), weil die Anzahl der konkurrierenden CPU´s im zweiten Fall halb so groß ist wie im ersten Fall. Die Differenz der Ausführungszeiten zwischen dem ersten und dem zweiten Fall beträgt 206 sec. Die Ausführungszeit des dritten Falles ist besser als die Ausführungszeit im ersten Fall, aber schlechter als die im zweiten Fall. Diese Tatsache widerspricht der These, dass der Speedup der CPU´s größer wird, wenn weniger CPU´s auf einem Rechner um den BUS konkurrieren. Dies resultiert daraus, dass beim dritten Fall die Anzahl der Rechner gestiegen ist und dadurch der Kommunikationsaufwand zwischen den CPU's unverhältnismäßig ansteigt.

Erster Fall: 4 Rechner mit 8 CPU´s
Zweiter Fall: 8 Rechner mit 4 CPU´s
Dritter Fall: 10 Rechner mit 3 CPU´s und 1 Rechner mit 1 CPU

Abbildung 4.24: test_240ms in drei Fällen

4.2.2 Gemeinsame Nutzung der Rechner

Szenario zwei

Die selben Jobs werden unter fast gleichen Bedingung wie im Fall eins aus Szenario eins berechnet. Der einzige Unterschied besteht darin, dass sich in diesem Szenario die drei Jobs den Rechner des Clusters teilen. Also wenn ein Rechner 8 CPU's besitzt, dann teilen sich die drei Jobs A, B und C die CPU's mit jeweils 4, 2, 2. Wenn die drei Jobs auf acht Rechner verteilt werden, dann wird Job A mit 32 CPU's, Job B und C mit jeweils 16 CPU's berechnet. Durch diese Aufteilung der drei Jobs wird das Verhalten im ersten Fall aus Szenario eins simuliert. In diesem Szenario werden die drei Jobs folgendermaßen berechnet. test_240ms wird auf 8 Rechnern mit jeweils 4 CPU´s berechnet. test_10ms und Audi-A2_Vorderwagen werden auf den gleichen 8 Rechnern wie test_240ms mit jeweils 2 CPU's berechnet. In diesem Szenario werden die Ausführungszeiten mit dem ersten Fall aus Szenario eins verglichen und bewertet.

Bewertung von Audi-A2_Vorderwagen und test_240ms

Es ist zu erkennen, dass die Ausführungszeiten von Audi-A2_Vorderwagen und test_240ms im ersten Fall aus Szenario eins geringer ausfallen, als wenn sie sich den Rechner teilen (siehe Abbildung 4.25 und 4.26). In diesem Szenario gibt es einen Nachteil. Der Nachteil besteht darin, dass die Berechnungen auf mehrere Rechner verteilt sind. Daraus resultiert,

dass der Nachrrichtenaustausch in diesem Fall länger dauert. Um diese Situation zu vermeiden, ist es sinnvoll bei einem Cluster, bei dem mehrere Benutzer auf die Ressourcen des Clusters zugreifen, die CPU's wie im ersten Fall aus Szenario eins zu belegen. Durch diese Art von Ressourcenverteilung werden die Ziele der Stapelverarbeitungssysteme erfüllt.

Erster Fall aus Szenario eins: 1528 sec
Szenario zwei: 1628 sec

Abbildung 4.25: Audi-A2_Vorderwagen aus Szenario zwei

Erster Fall aus Szenario eins: 3482 sec
Szenario zwei: 3661 sec

Abbildung 4.26: test_240ms aus Szenario zwei

Bewertung von test_10ms

Von der Theorie ausgehend, wird bei test_10ms erwartet, dass die Ausführungszeit von Fall eins bei Szenario eins geringer ausfällt als bei dem jetzt bertachteten Fall. Bei der praktischen Umsetzung ergibt sich jedoch ein anderes Resultat (siehe Abbildung 4.27). Die Ausführungszeiten bei Szenario zwei fallen unerwarteter Weise geringer aus als bei Fall eins von Szenario eins. Der Verfasser begründet dieses Ergebnis mit der sehr geringen Größe des Jobs. Also hoher Kommunikationsaufwand im Verhältnis zur Rechenzeit.

Erster Fall aus Szenario eins: 444 sec
Szenario zwei: 274 sec

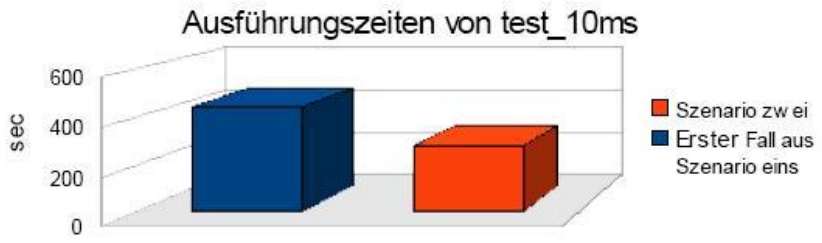

Abbildung 4.27: test_10ms aus Szenario zwei

5 Zuweisung von Cluster-Ressourcen

Wie bereits in der Einführung erwähnt, ist es, um den Cluster optimal auszulasten, erforderlich eine Zuteilungssoftware zu entwickeln, die als Schnittstelle zwischen Benutzer und Jobmanagement-System agiert. Das heißt, dass die Benutzer ihre Berechnungen der Zuteilungssoftware übergeben. Ab diesem Punkt übernimmt die Zuteilungssoftware die weiteren Bearbeitungsschritte. Die Aufgabe der Zuteilungssoftware ist die optimale Verteilung der Ressourcen zwischen den Benutzern. Um dies realisieren zu können, muss die Zuteilungssoftware eine Vorhersage über den Ressourcenverbrauch eines Jobs (Berechnung) machen. Desweiteren ist notwendig, dass die Zuteilungssoftware die Informationen über die Auslastung des Clusters ermittelt, um die freien Rechner für diesen Job herauszufiltern. Aufgrund der Voruntersuchungen in Kapitel 4 kann die geforderte Zuteilungssoftware entwickelt werden (siehe Kapitel 5.2).

5.1 Aufbau des Systems

Das gesamte System besteht aus zwei Servern (siehe Abbildung 5.1), der BENUTZER/SERVER der in Abbildung 5.1a dargestellt ist, verwaltet alle Benutzerdateien. Der LS-DYNA/SERVER (siehe Abbildung 5.1b) ist für die Durchführung der Berechnungen verantwortlich. Auf diesem Server agieren PBS, LS-Dyna, HP-MPI und das Perl-Skript, welches die Zuteilungssoftware beinhaltet. Die Kommunikation zwischen den zwei Servern wird über das Netz realisiert.

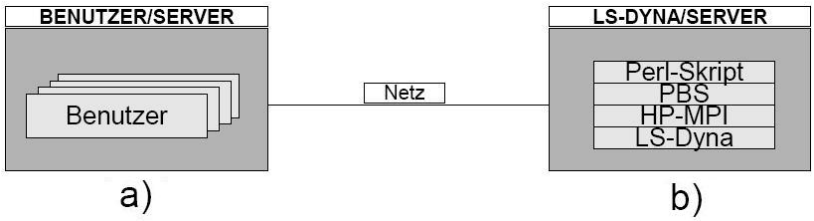

Abbildung 5.1: Aufbau des Systems

5.2 Zuteilungssoftware

5.2.1 Ein und Ausgangsparameter

Die Zuteilungssoftware basiert auf der Vorhersage von LS-Dyna. Hierbei wird die Vorhersage von LS-Dyna als Eingangsparameter benutzt (siehe Abbildung 5.2). Durch den Eingangsparameter kann im nächsten Schritt die Anzahl an Ressourcen (CPU's) ermittelt werden. Mit Hilfe der ermittelten Ressourcen, kann der Ausgangsparameter (Rechner) festgestellt werden. Hierbei werden die Ressourcen (CPU's) anhand des in Kapitel 4.2.1 dargestellten Szenarios eins Fall eins auf den Ausgangsparameter veteilt. Dadurch wird erreicht, dass ein Job auf so wenig Rechner wie möglich verteilt wird, um alle CPU's der beteiligten Rechner für diesen Job in Anspruch zu nehmen. Dieser Ansatz führt dazu, dass die Ausführungszeiten der Jobs geringer werden. Hierbei wird weiterhin davon ausgegangen, dass mehrere Jobs den Cluster gleichzeitig in Anspruch nehmen können.

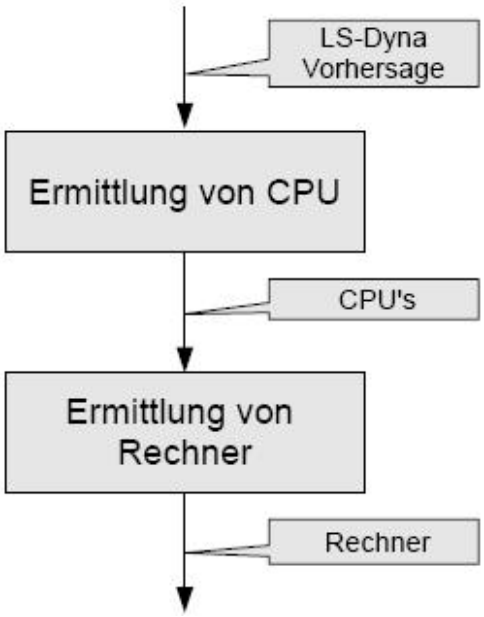

Abbildung 5.2: Ein und Ausgangsparameter

5.2.2 Vorhersagen von LS-Dyna

In Abbildung 5.3 wird die Anzahl an CPU's in Abhängigkeit der Vorhersagen von LS-Dyna dargestellt. Wenn jetzt die Vorhersage von LS-Dyna für einen neuen Job angewendet wird,

können anhand von Abbildung 5.3 die Ressourcen für den neuen Job ermittelt werden. Liegt nun die Vorhersage der Ausführungszeit von LS-Dyna für den neuen Job zwischen 45000 sec und 1500000 sec, werden mit Hilfe der Abbildung 5.3 nun für diesen neuen Job 32 CPU's zur Verfügung gestellt. Dabei beruht Abbildung 5.3 auf Ergebnisse, die durch Abbildung 5.4 entnommen werden. Die Anzahl der Ressourcen, welche für die Vorhersage von LS-Dyna

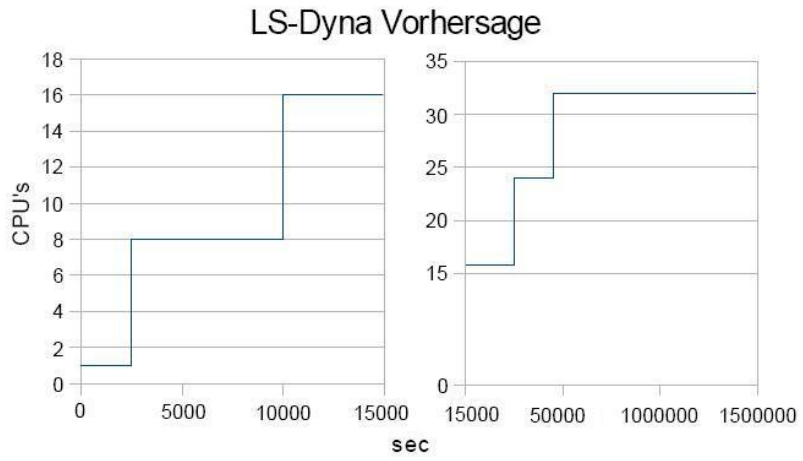

Abbildung 5.3: Vorhersagen von LS-Dyna

ermittelt wird, beruht auf die Auswertung der Rechenzeiten von LS-Dyna Jobs (siehe Abbildung 5.4). Weiterhin kann aus dem Diagramm (siehe Abbildung 5.4) entnommen werden,

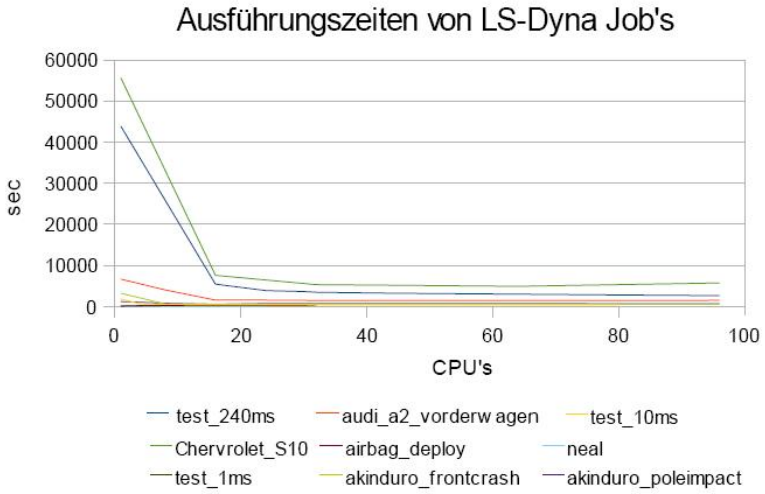

Abbildung 5.4: Ausführungszeiten von LS-Dyna Jobs

dass die maximale Ressourcenanzahl, die für die Durchführung größerer Jobs (siehe Ausführungszeiten von test_240 und Chervrolet_S10) benötigt wird, bei 32 CPU's liegen sollte.

Der Grund liegt darin, dass es bei einer weiteren Erhöhung der Anzahl auf über 32 CPU's, zu keiner bzw. einer sehr geringen Reduktion der Rechenzeit kommt. Durch diese Art der Ressourcenverteilung, werden die in Kapitel 4.1.2 (siehe Szenario eins) beschriebenen langen Wartezeiten und die ineffiziente Ressourcenverwendung vermieden. Daher werden für alle neuen Jobs die im Intervall von 45000 sec und unendlich liegen (siehe Abbildung 5.3) 32 CPU's zur Verfügung gestellt. Weiterhin haben kleinere Jobs, die im Intervall von 0 bis 10000 sec liegen eine höhere Priorität. Dass heißt, sie können Jobs die im Bereich von 10000 sec und unendlich liegen unterbrechen (siehe Kapitel 2.2.1). Dies ist dann der Fall, wenn alle Ressourcen des Clusters vollständig ausgeschöpft sind.

5.2.3 Ressourcenverteilung

Anhand von zwei Fällen wird gezeigt, wie die Ressourcen (CPU's) für die Jobs zugeteilt werden.
Die Ausführungszeiten von Job A betragen für 88 CPU's 2700 sec und für 30 CPU's 3400 sec (siehe Abbildung 5.5).
Die Ausführungszeiten von Job B betragen für 88 CPU's 2300 sec und für 30 CPU's 3000 sec (siehe Abbildung 5.5).
Job A wird kurz vor Job B zur Bearbeitung an den Cluster geschickt.

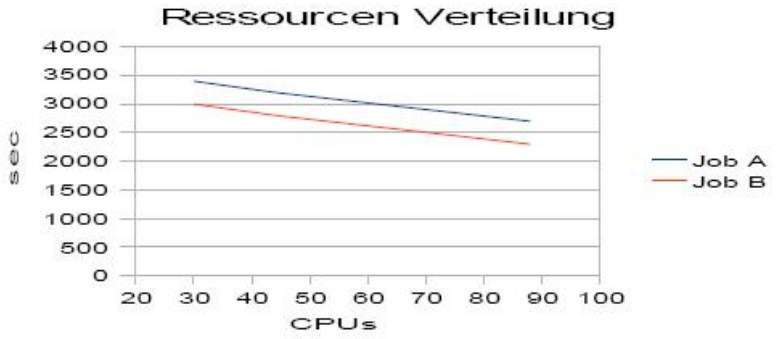

Abbildung 5.5: Ressourcen Verteilung

In Fall eins werden beide Jobs mit jeweils 88 CPU's auf einem Cluster, der 96 CPU's zur Verfügung hat, zur Bearbeitung gesendet (siehe Abbildung 5.6).

Hier ist Job B gezwungen wegen Ressourcenknappheit zu warten. In diesem Fall beträgt die durchschnittliche Durchlaufzeit 3650 sec, die durchschnittliche Wartezeit 2500 sec und

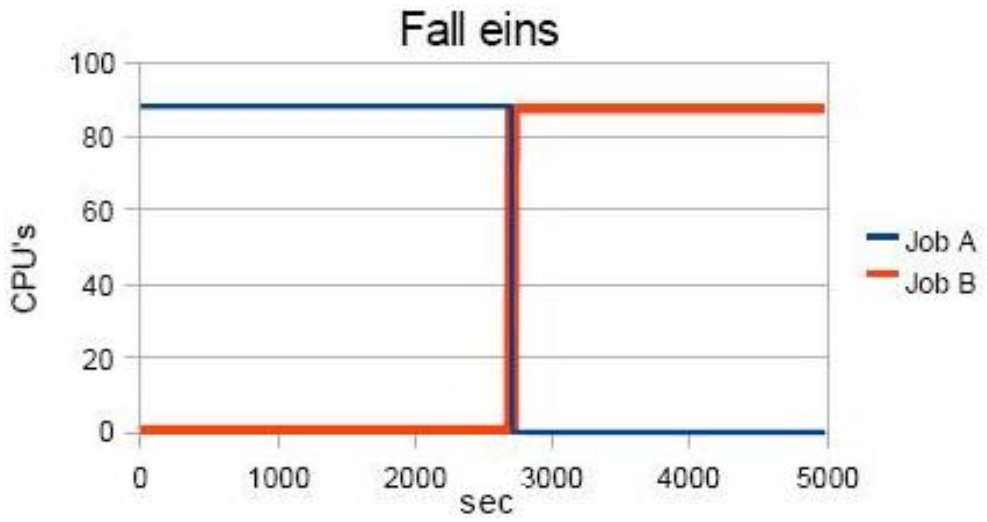

Abbildung 5.6: Fall eins Optimierung

die durchschnittlichen freien Ressourcen liegen bei 8 CPU's. Im zweiten Fall (siehe Abbildung 5.7) werden beide Jobs mit jeweils 30 CPU's berechnet. Hier liegt die durchschnittliche Durchlaufzeit bei 3200 sec, die durchschnittliche Wartezeit bei 0 sec und die durchschnittlichen freien Ressourcen liegen bei 36 CPU's. Wenn aber davon ausgegangen wird, dass nur Job A durch den Cluster bearbeitet werden kann, dann ist es sinvoll, dass der Job mit 88 CPU's berechnet wird, um die optimale Durchlaufzeit von Job A zu realisieren. Dies ist nicht realisierbar, da nicht vorhergesagt werden kann, wann der nächste Job den Cluster in Anspruch nehmen wird. Um mehreren möglichen Benutzern gerecht zu werden, wird die Implementierung anhand der im Fall zwei beschriebenen Situation realisiert.

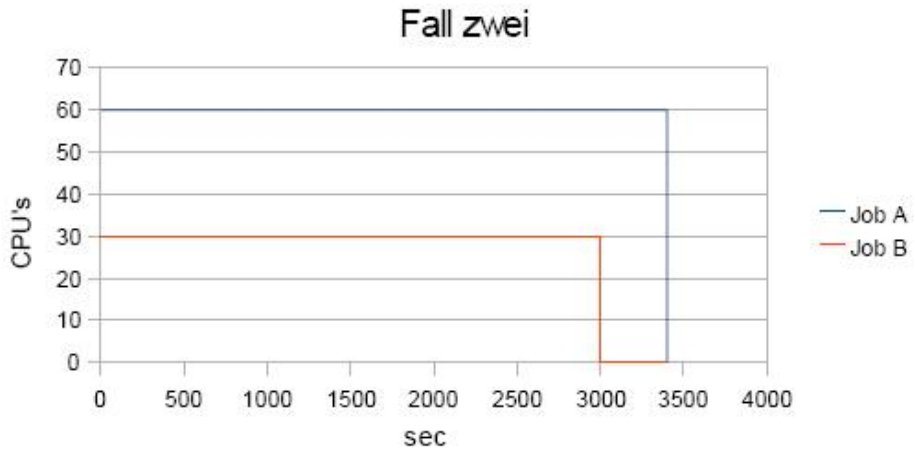

Abbildung 5.7: Fall zwei Optimierung

5.2.4 Arbeitsweise des Algorithmus

Die Vorhersage, welche der Algorithmus durchführt, wird anhand eines Beispiels dargestellt. In Abbildung 5.8 wird der Verlauf der Ausführungszeiten in Abhängigkeit der Anzahl von unterschiedlichen Jobs dargestellt.

Abbildung 5.8: Ausführungszeiten von Job 1 bis 8

Die in Anspruch genommenen Ressourcen für diese Jobs sind in Abbildung 5.9 zu sehen.

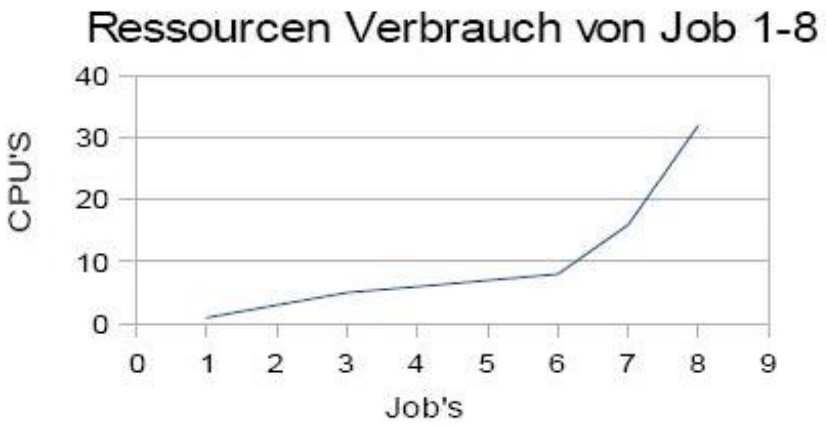

Abbildung 5.9: Ressourcen Verbrauch von Job 1 bis 8

In Abbildung 5.9 ist zu erkennen, dass für Job 8 32 CPU's zur Verfügung gestellt wurden.

Desweiteren werden in Abbildung 5.10 die Ausführungszeiten von Job 8 mit einer unterschiedlichen Anzahl an CPU's dargestellt. Durch die Bewertung der Ressourcen die oben

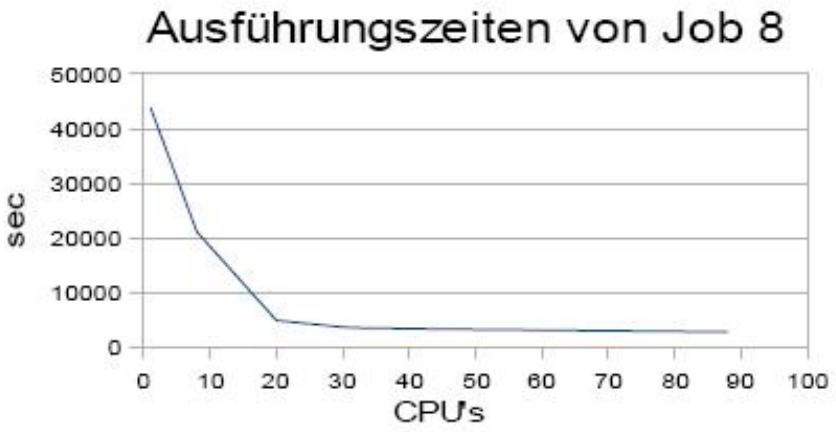

Abbildung 5.10: Ausführungszeiten von Job 8

analysiert wurde (siehe Kapilel 5.2.2 und 5.2.3), bekommt Job 8 32 CPU's für die Abarbeitung zugeteilt. Die Vorhersage der Ausführungszeit von LS-Dyna für Job 8 beträgt bei Einsatz einer CPU, 71188 sec. Anhand eines Beispiels wird dargestellt, wie die Ressourcen für einen neuen Job verteilt werden. Wenn jetzt die Vorhersage von LS-Dyna für einen neuen Job angewendet wird, wobei dieser neue Job in etwa mit Job 8 vergleichbar ist, dann kann davon ausgegangen werden, dass die Berechnung des neuen Jobs mit 32 CPU's am zweckmäßigsten ist, da die Ausführungszeit, die aus der Benutzung von 32 CPU's resultiert, zu einer drastischen Reduktion der Rechenzeit führt. Nach diesem Schema, werden alle neuen Jobs bearbeitet. Dabei können zwei Jobs jedoch niemals vollkommen identisch sein. Deshalb wird immer von einer Abweichung ausgegangen, die jedoch beim Vergleich mit Job 8 unerheblich ist. Solange sich die Jobgröße in einem festgesetzten Intervall bewegt, kann ein Vergleich zu Job 8 hergestellt werden. Sind die verfügbaren Ressourcen des Clusters geringer als 32 CPU's, dann entscheidet der Algorithmus, ob der Job die geringe Anzahl an Ressourcen bekommt, um vom Cluster bearbeitet zu werden oder doch auf die Ressourcen warten soll, da dies für den Job am zweckmäßigsten ist. Diese beschriebene Situation wird anhand eines Flussdiagramms verdeutlicht (siehe Abbildung 5.11).

Wie oben besprochen, soll Job 8 32 CPU's für die Bearbeitung bekommen. In der ersten Verzweigung wird kontrolliert, ob die freien CPU's des Clusters mehr als 27 betragen. Sollte dies der Fall sein, werden dem Job 8 mindestens 28 CPU's zugeteilt. Ist die Anzahl der freien CPU's des Clusters kleiner 28 gibt es zwei weitere Abfragen der freien CPU's, nähmlich größer 19 und größer 13, um eine sofortige Bearbeitung durch den Cluster zu ermöglichen. Ist die Anzahl an freien CPU´s kleiner 14, werden dem Job 8 32 CPU's zugewiesen. Da in

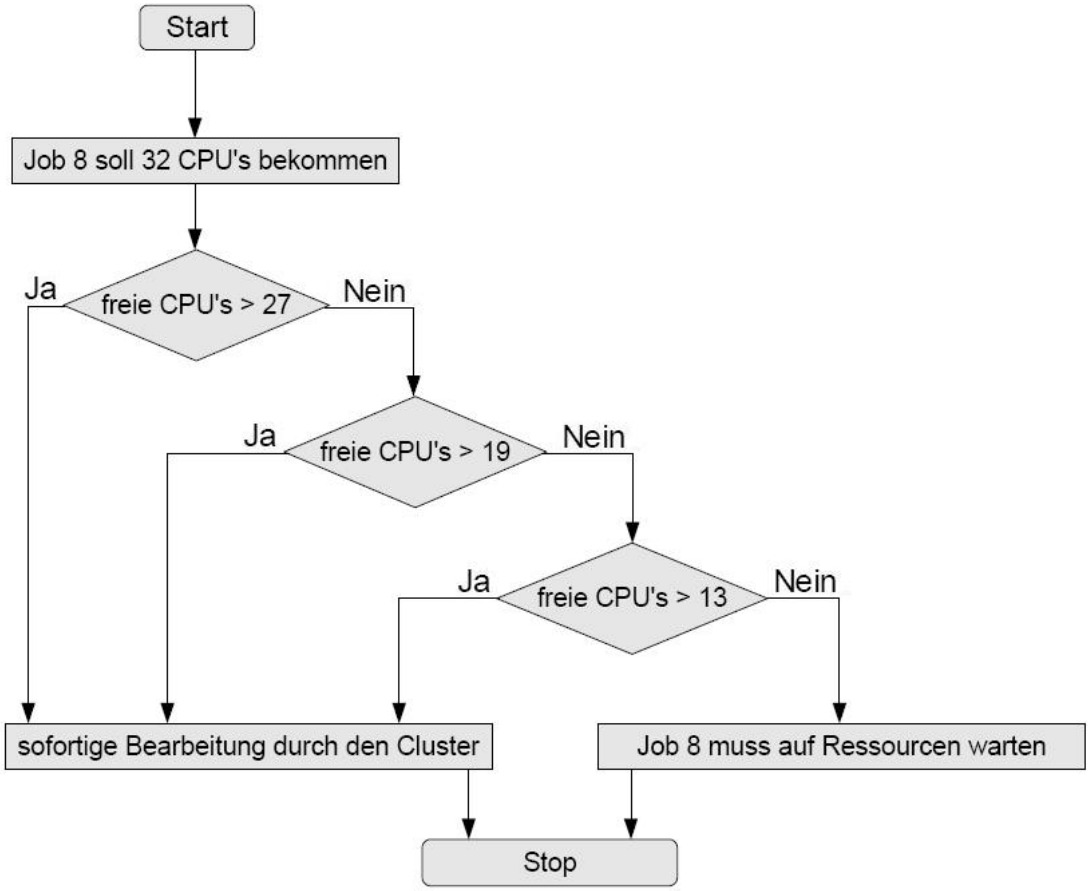

Abbildung 5.11: Flussdiagramm

diesem Fall, die freien CPU's unter 32 liegen, muss Job 8 auf Ressourcen des Clusters warten. Das Minimum an Ressourcen, welche für Job 8 zur Verfügung gestellt werden beträgt 14 CPU's. Diese Schranke ist vom Verfasser definiert, da hier mit einer geringen Anzahl an Ressourcen es zu einer deutlichen Reduzierung der Rechenzeit kommt (siehe Ausführungszeiten von Job 8). Dadurch sind die Ziele der Stapelverarbeitungssysteme, die besagen, minimiere die Zeit vom Start bis zur Beendigung eines Prozesses und maximiere die Jobs pro Stunde, erfüllt (siehe Kapitel 2.2.2).

5.3 Implementierung der Zuteilungssoftware

Die Zuteilungssoftware die mit einem Perl-Skript implementiert wurde, wird anhand eines Sequenzdiagramms erläutert (siehe Abbildung 5.12).

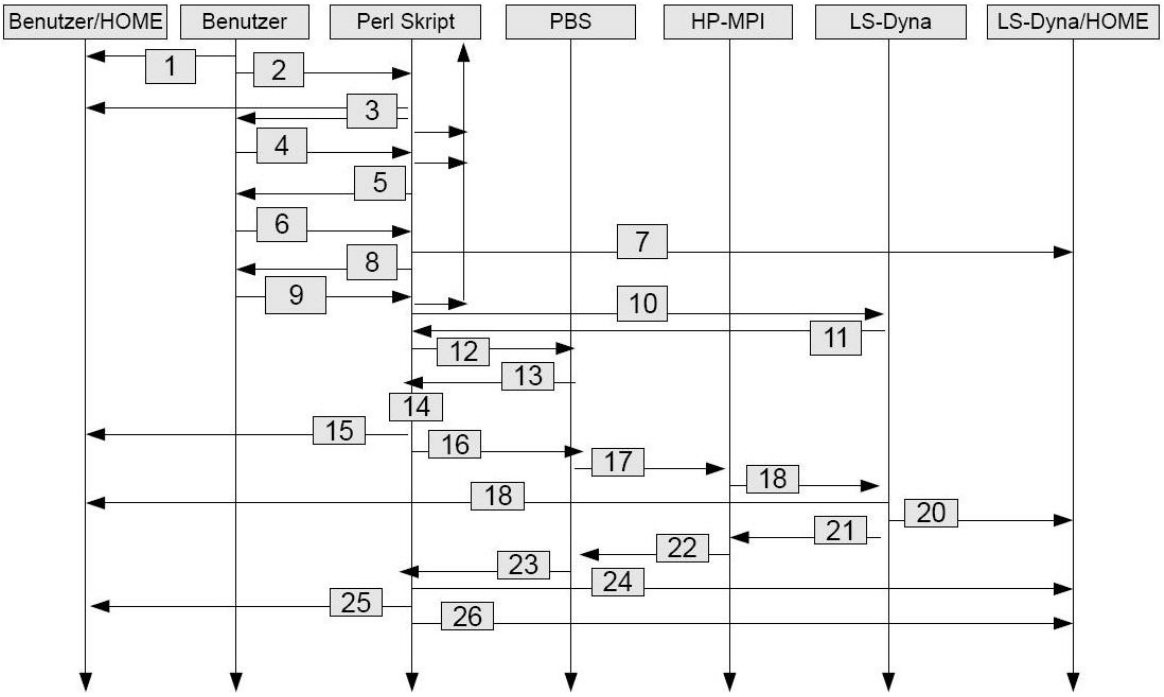

Abbildung 5.12: Implementierung des Algorithmus

Schritt 1: Der Benutzer wechselt in das Verzeichnis, indem sein LS-Dyna Inputfile liegt. Dabei muss das Verzeichnis im HOME-Bereich des Benutzers liegen. Weiterhin darf das Verzeichnis keinen appfile von HP-MPI beinhalten, da davon ausgegangen wird, dass hier eine Berechnung an PBS geschickt wurde und diese in einer Queue des PBS wartet.

Schritt 2: Der Benutzer startet das Perl-Skript.

Schritt 3: Das Perl-Skript durchsucht das Verzeichnis, welches der Benutzer in Schritt 1 ausgewählt hat nach LS-Dyna Inputfiles. Wenn ein oder mehrere Inputfiles vorhanden sind, dann wird dies dem Benutzer als Liste ausgegeben. Sind keine LS-Dyna Files vorhanden, wird dies dem Benutzer mitgeteilt und das Programm wird abgebrochen.

Schritt 4: In Schritt 4 teilt der Benutzer dem Perl-Skript mit, ob er ein LS-Dyna Inputfile zur Berechnung an den Cluster schicken möchte, oder das Programm beendet werden soll.

Schritt 5: Hat sich der Benutzer für die Berechnung entschieden, fordert das Perl-Skript den Benutzer auf, einen Namen für ein Verzeichnis anzugeben, welches vom Perl-Skript erzeugt wird. Das Verzeichnis dient dazu, um später die Ergebnisse von LS-Dyna zu speichern.

Schritt 6: Der Benutzer gibt einen Namen für ein Verzeichnis ein.

Schritt 7: Das Perl-Skript erstellt zwei Verzeichnisse für LS-Dyna. Zum Einen ein Verzeichnis, welches die Zwischenergebnisse speichert und zum Anderen ein zweites Verzeichnis, welches dazu dient das Endergebnis zu speichern. Um schneller zu sein (siehe Kapitel 2.3.3), werden die beiden Verzeichnisse im LS-DYNA/SERVER abgelegt und nicht im BENUTZER/SERVER.

Schritt 8: Das Perl-Skript gibt eine Liste zurück, die die bisherigen Eingaben des Benutzers darstellt. Der Benutzer kann sich jetzt entscheiden, ob die Berechnung ausgeführt oder abgebrochen werden soll.

Schritt 9: Der Benutzer entscheidet sich für eine der in Schritt 8 aufgeführten Alternativen.

Schritt 10: Das Perl-Skript macht eine Anfrage an LS-Dyna. Hierbei benutzt LS-Dyna die SMP Version, da diese eine Vorhersage geben kann.

Schritt 11: Die bei Schritt 10 ermittelte Vorhersage wird an das Perl-Skript weitergeleitet.

Schritt 12: Um die optimalen Rechner für die Berechnung herauszufinden, setzt sich das Perl-Skript mit PBS in Verbindung.

Schritt 13: Anhand der zurückgegebenen Informationen über die Auslastung des Clusters entscheidet das Perl-Skript über die weiteren Schritte für die Bearbeitung des Jobs. Sind genügend Ressourcen vorhanden, sucht das Perl-Skript die Rechner aus, die am effizientesten für die Abarbeitung des Jobs sind. Die Verteilung des Jobs an die Rechner des Clusters wird anhand des Kapitels 4.2.1 Szenario eins Fall eins dargestellt. Dieser besagt, dass die Jobs auf so wenig Rechner wie möglich verteilt werden sollen, um geringe Durchlaufzeiten der Jobs zu realisieren, da davon ausgegangen wird, dass mehrere Benutzer den Cluster in Anspruch nehmen. Diese Situation wird in Szenrio zwei des Kapitels 4.2.2 dargestellt und vom Perl-Skript vermieden.

Schritt 14: Als nächstes wird durch den entwickelten Algorithmus vorhergesagt, welche Ressourcen für diesen Job optimal sind (siehe Kapitel 5.2.4).

Schritt 15: Das Perl-Skript benutzt die gespeicherten Informationen (siehe Schritt 4 bis Schritt 14), um die Dateien für PBS (pbs.sh), HP-MPI (appfile) und LS-Dyna (pfile, names) zu erzeugen. Die Dateien werden vom Perl-Skript im Home-Verzeichnis des Benutzers abgelegt.

Schritt 16: Das Perl-Skript übergibt dem PBS ein Skript mit dem Befehl (qsub pbs.sh) eine Kopie des Skriptes pbs.sh, die alle Informationen über Ressourcen, Job-Name und den Startvorgang von HP-MPI beinhaltet.

Schritt 17: Sind die Ressourcen verfügbar, startet PBS HP-MPI. Anderenfalls kommt der Job in eine der vordefinierten Queus, bis so viele Ressourcen vorhanden sind, dass der Job abgearbeitet werden kann.

Schritt 18: HP-MPI startet LS-Dyna, der für die Berechnung des Jobs zuständig ist.

Schritt 19: LS-Dyna liest das HOME-Verzeichnis des Benutzers nach names (siehe Schritt 15). In names sind die Informationen über Jobname und pfile vorhanden. In pfile stehen die Pfade der beiden Verzeichnisse, in der LS-Dyna die Zwischen-und Endergebnisse speichert.

Schritt 20: LS-Dyna legt die Zwischen - und Endergebnisse in dem Verzeichnis ab, die im Schritt 7 durch das Perl-Skript erzeugt wurde.

Schritt 21: LS-Dyna meldet HP-MPI die Beendigung der Berechnung.

Schritt 22: HP-MPI meldet PBS die Beendigung der Berechnung.

Schritt 23: PBS meldet dem Perl-Skript die Beendigung der Berechnung.

Schritt 24: Das Perl-Skript kopiert das Endergebnis aus dem LS-DYNA/SERVER.

Schritt 25: Das kopierte Endergebnis aus Schritt 24, wird im Homeverzeichnis des Benutzers gespeichert, aus dem das Perl-Skript gestartet worden ist (siehe Schritt 1).

Schritt 26: Die Zwischen-und Endergebnisse werden aus dem LS-DYNA/SERVER gelöscht, da die Dateien viel Speicherplatz in Anspruch nehmen.

5.4 Bewertung der Zuteilungssoftware

Wie besprochen, verwendet die Zuteilungssoftware die Zeitvorhersage von LS-Dyna, um die optimalen Ressourcen für einen Job zu ermitteln. Dabei liest die SMP-Version von LS-Dyna den Inputfile, welcher durch den Benutzer im dritten Schritt übergeben wurde und macht eine Vorhersage über die Länge der Ausführungszeit für diesen Job. Es kann vorkommen, dass die SMP-Version von LS-Dyna keine Vorhersage über einen Job machen kann. Dies ist dann der Fall, wenn ein Job mehr als 16 GB Speicherplatz benötigt. Der Grund hierfür ist, dass ein Rechner auf dem Cluster nur 16 GB Speicherplatz zur Verfügung hat. LS-Dyna holt diese Information von einem Rechner. Die übrigen Rechner werden ignoriert. Ein weiterer möglicher Grund dafür, dass die SMP-Version von LS-Dyna keine Vorhersage über einen Job machen kann, liegt darin, dass das Inputfile fehlerhaft ist. Hier kann die Zuteilungssoftware nicht unterscheiden, ob das Inputfile fehlerhaft ist oder mehr als 16 GB Speicherplatz benötigt. Deswegen werden diese Art von Inputfiles mit 32 CPU's belegt und an das PBS zur Bearbeitung geschickt. Wenn das Inputfile fehlerhaft ist, dann wird dieser Vorgang durch PBS abgebrochen und der Fehler an den Benutzer geschickt. Wenn das Inputfile mehr als 16 GB Speicherplatz benötigt, dann kann dies bearbeitet werden, da nun 4 Rechner mit jeweils 16 GB zur Verfügung stehen. Sollten alle CPU's des Front-End Servers am Rechnen sein, kann ein weiteres Problem auftauchen. Hierbei kann es vorkommen, das der Front-End Server eins bis zwei Minuten benötigt, um die Anfragen vom Benutzer zu bearbeiten (Schritt 1 bis Schritt 14). Ist der Cluster durch größere Jobs belegt, kann das System kleinere Jobs sofort dran nehmen, um deren optimale Durchlaufzeit zu garantieren. Desweitern erlaubt die Zuteilungssoftware mit Hilfe der Voruntersuchungen aus Kapitel 4 die optimale Auslastung des Clusters. Das bedeutet, dass die Nachteile die in Kapitel 4 beschrieben wurden, durch die Zuteilungssoftware vermieden werden. Um Jobs starten zu können, muss ein Benutzer die Shell-Programmierung und ein Verständnis für HP-MPI und LS-Dyna aufweissen. Mit Hilfe der Zuteilungssoftware, bleibt diese dem Benutzer erspart. Die Zuteilungssoftware übernimmt die Aufräumarbeiten von Dateien, die durch LS-Dyna erzeugt wurden und für Benutzer nicht mehr relevant sind. Weiterhin ist die Zuteilungssoftware so aufgebaut, dass die verschiedenen Lösungsansätze die durch die Methoden implementiert wurden, voneinander unabhängig sind. Somit ist es möglich, Methoden die nicht mehr gebraucht werden zu deaktivieren oder neue Methoden die für ein neues Berechnungsprogramm angewendet werden zu aktivieren. Die Zuteilungssoftware unterstützt die beiden beschriebenen Ansätze aus Kapitel 4.2.1 (Isolierte Nutzung der Rechner) und Kapitel 4.2.2 (Gemeinsame Nutzung der Rechner). Dabei ist die Methode (Jobs_teilen_die_Rechner) deaktiviert.

6 Zusammenfassung

Im Rahmen der Bacheloararbeit wurde in Kapitel 3 über eine Auswahl von Jobmanagement-Systemen diskutiert. Durch die Kriterien die vom Verfasser aufgelistet wurden, konnte kein Unterschied zwischen PBS Professional und LSF erkannt werden. Der Verfasser entschied sich für das PBS, da bereits Lizenzen für diese Software zur Verfügung standen. Um Jobs starten zu können, muss ein Benutzer die Shell-Programmierung und ein Verständnis für HP-MPI und LS-Dyna aufweisen. Weiterhin werden in Kapitel 4 die grundlegenden Erkenntnisse zur Optimierung des Clusters thematisiert. Es ist ersichtlich geworden, dass die optimale Auslastung des Clusters von der CPU-Anzahl, welche durch einen Job in Anspruch genommen wird und die Aufteilung des Jobs auf die Rechner abhängig ist. Diese beschriebenen Nachteile führten dazu, dass eine Zuteilungssoftware entwickelt werden musste. Die Zuteilungssoftware basiert auf der Vorhersage von LS-Dyna. Hierbei wird die Vorhersage von LS-Dyna als Eingangsparameter benutzt um die Anzahl an Ressourcen (CPU's) für ein LS-Dyna Job zu ermitteln.

6.1 Forschungsbedarf

Die Effizienz des Clusters kommt dann zur Geltung, wenn ein Job auf mehrere Rechner verteilt wird. Dadurch wird erreicht, dass die Anzahl der CPU's, die um den BUS konkurrieren, geringer wird da die CPU's auf mehrere Rechner verteilt werden. Das hat zur Folge, dass der Speedup der CPU's größer wird. Dieser beschriebene Ansatz wird durch die implementierte Zuteilungssoftware nicht unterstützt, da davon ausgegangen wird, dass der Cluster immer voll ausgelastet ist. Wenn davon ausgegangen wird, dass der Cluster an manchen Tagen nicht voll ausgelastet ist, dann ist es sinvoll an diesen Tagen, die Jobs auf mehrere Rechner zu verteilen, um die optimale Durchlaufzeit der Jobs zu realisieren. Der Verfasser schlägt hier vor, die Historie der Auslastung des Clusters über Tage bzw. Monate zu speichern. Anhand der gespeicherten Informationen soll sich die implementierte Zuteilungssoftware anpassen. Somit können die beiden oben diskutierten Szenarien unterstützt werden.

6.2 Fazit

Durch das Jobmanagement-System kann der Cluster für mehrere Benutzer zur Verfügung gestellt werden. Mit Hilfe der Zuteilungssoftware können Benutzer bequem ihre Berechnungen dem Cluster übergeben. Die Zuteilungssoftware ermöglicht eine Vorhersage über den Ressourcenverbrauch eines Jobs. Daraus resultiert, dass die durchschnittliche Durchlaufzeit der Jobs minimiert wird. Ist der Cluster durch größere Jobs belegt, kann das System kleinere Jobs sofort dran nehmen, um deren optimale Durchlaufzeit zu garantieren. Weiterhin ist die Zuteilungssoftware so aufgebaut, dass die verschiedenen Lösungsansätze die durch die Methoden implementiert wurden, voneinander unabhängig sind. Somit ist es möglich, Methoden die nicht mehr gebraucht werden zu deaktivieren oder neue Methoden die für ein neues Berechnungsprogramm angewendet werden zu aktivieren.

Tabellenverzeichnis

3.1 LSF Queues 27
3.2 LSF Befehle 27
3.3 PBS Queues 29
3.4 PBS Befehle 30
3.5 Jobmanagement-Systeme 32
3.6 Scheduler 33
3.7 Queues 33
3.8 Sicherheit 33
3.9 Job Management 34
3.10 Ressourcen Management 34
3.11 Weitere Dienste 34

4.1 Bearbeitungszeiten des Szenarios eins 41
4.2 Bearbeitungszeiten des Szenarios zwei 47
4.3 Bewertung des Senarios eins 49
4.4 Bewertung des Senarios zwei 50

Abbildungsverzeichnis

2.1 Cluster der HAW-Hamburg . . . 7
2.2 LS-Dyna Jobs . . . 8
2.3 nicht-preemptives Scheduling-Verfahren . . . 9
2.4 preemptiven Scheduling-Verfahren . . . 10
2.5 First-Come First-Served [2] . . . 12
2.6 Shortest Job First [2] . . . 13
2.7 Round-Robin [2] . . . 14
2.8 Klassifizierung nach Flynn . . . 15
2.9 Klassifikation von Parallelrechnern [3] . . . 16
2.10 Multiprozessorsysteme [2] . . . 17
2.11 UMA-busbasierte SMP-Architekturen [2] . . . 18
2.12 Message-Passing-Multicomputer [2] . . . 19
2.13 Ausführungszeiten von test_10ms.k . . . 19
2.14 Parallele Leistungsmaße . . . 20
2.15 CAD und FEM [5] . . . 22
2.16 MPP [18] . . . 23

3.1 Jobmanagement-Systeme . . . 25
3.2 xlsbatch [8] . . . 28
3.3 xlsbatch-Doppelklick [8] . . . 28
3.4 xpbsmon [13] . . . 31
3.5 xpbsmon-Doppelklick . . . 31
3.6 xpbs . . . 32

4.1 Ausführungszeiten von test_10ms . . . 37
4.2 Effizienz von test_10ms . . . 37
4.3 Speedup von test_10ms . . . 38
4.4 Ausführungszeiten von mat_spring.belted.dummy . . . 38
4.5 Speedup von mat_spring.belted.dummy . . . 39
4.6 Effizienz von mat_spring.belted.dummy . . . 39
4.7 Ausführungszeiten von Audi-A2_Vorderwagen . . . 40
4.8 Effizienz von Audi-A2_Vorderwagen . . . 40
4.9 Speedup von Audi-A2_Vorderwagen . . . 41

4.10 First-Come First-Served 42
4.11 First-Come First-Served Cluster 42
4.12 Shortest Job First 43
4.13 Shortest Job First Cluster 43
4.14 Round-Robin q=1 44
4.15 Round-Robin q=1 Cluster 44
4.16 Round-Robin q=3 45
4.17 Round-Robin q=3 Cluster 45
4.18 Round-Robin q=4 46
4.19 Round-Robin q=4 Cluster 46
4.20 FCFS, SJF, RR q=1, 3, 4 48
4.21 FCFS, SJF, RR q=1=3=4 Cluster 48
4.22 test_10ms in drei Fällen 51
4.23 Audi-A2_Vorderwagen in drei Fällen 52
4.24 test_240ms in drei Fällen 53
4.25 Audi-A2_Vorderwagen aus Szenario zwei 54
4.26 test_240ms aus Szenario zwei 54
4.27 test_10ms aus Szenario zwei 55

5.1 Aufbau des Systems 56
5.2 Ein und Ausgangsparameter 57
5.3 Vorhersagen von LS-Dyna 58
5.4 Ausführungszeiten von LS-Dyna Jobs 58
5.5 Ressourcen Verteilung 59
5.6 Fall eins Optimierung 60
5.7 Fall zwei Optimierung 60
5.8 Ausführungszeiten von Job 1 bis 8 61
5.9 Ressourcen Verbrauch von Job 1 bis 8 61
5.10 Ausführungszeiten von Job 8 62
5.11 Flussdiagramm 63
5.12 Implementierung des Algorithmus 64

Literaturverzeichnis

Buchreferenzen

[1] Möller Dietmar: Rechnerstrukturen, Springer-Verlag, 2002.

[2] Tanenbaum Andrew S.: Moderne Betriebssysteme, Pearson Studium, 2002.

[3] Oberschelp Waltern, Vossen,Gottfied: Rechneraufbau und Rechnerstrukturen, Oldenbourg-Verlag ,2003.

[4] Schumacher Axel: Optimierung mechanischer Strukturen, Springer-Verlag, 2004.

[5] Klein Bernd: FEM, Vieweg+Teubner Verlag, 2007.

[6] Stallings William: Betriebssysteme, Pearson Studium, 2005.

Artikelreferenzen

[7] Altair GRID TECHNOLOGIES (Hrsg.): PBS PRO-Administrator Guide 5.4, http://www.mta.ca/torch/pdf/pbspro54/pbsproag_5_4_0.pdf, -Aktualisierungsdatum: 2004.02.11.

[8] Platform Computing Corporation (Hrsg.): LSF Batch User's Guide, http://ls11-www.informatik.uni-dortmund.de/people/hermes/manuals/LSF/users.pdf, -Aktualisierungsdatum: 1998.08.01.

[9] Platform Computing Corporation (Hrsg.): Administering Platform LSF, http://www.qub.ac.uk/directorates/media/Media,54678,en.pdf, -Aktualisierungsdatum: April 2006.

[10] Altair GRID TECHNOLOGIES (Hrsg.): PBS PRO-User Guide 5.3
http://help.chem.upenn.edu/docs/pbspro/PBSproUG_53.pdf,
-Aktualisierungsdatum: 2003.03.07

[11] Christine Otto, Thomas Sesselmann: Analyse und Test
unterschiedlicher Grid-und Clusersysteme zum verteilten Rechnen,
Technische Universität Ilmenau, Wintersemester 2003/2004.

[12] Günther Lange: Bundesanstalt für Wasserbau-Dienststelle Hamburg,
Supercomputing News, Dezember 2002.

[13] Christine Baun, Analyse vorhandener Grid-Technologien
zur Evaluation eines Campus Grid an der Hochschule Mannheim,
Hochschule Mannheim Fakultät für Informatik, Wintersemester 2005.

[14] Altair GRID TECHNOLOGIES (Hrsg.): PBS PRO-Administrator Guide 9.2.

[15] Altair GRID TECHNOLOGIES (Hrsg.): PBS PRO-User Guide 9.2.

[16] Bernward Platz: LSF, OpenPBS und GridEngine: Jobmanagement-Systeme im Vergleich, Linux Magazin, Mai 2002.

Web-Quellen

[17] CADFEM-Homepage,
http://www.cadfem.de/produkte/ls-dyna.html, Stand von 24.03.2009.

[18] CADFEM-Homepage,
http://www.cadfem.de/seminare/informationstage.html?tx_
cfseminar_pi1[single]=2860&cHash=1ef7b7002b,
Stand von 24.03.2009.

[19] LSTC-Homepage,
http://www.lstc.com/lsdyna.htm, Stand von 23.03.2009.

[20] Platform-Homepage,
http://www.platform.com/Products/platform-lsf/technical-information,
Stand von 23.03.2009.

[21] CADFEM-Homepage,
http://www.cadfem.de/seminare/informationstage.html?tx_cfseminar_pi1[single]=2860\&cHash=1ef7b7002b, Stand von 24.03.2009.
[22] clucon-Homepage, Jobmanagement-Systeme im Vergleich,
http://www.clucon.de, Stand von 02.04.2009.

[23] PBS-Homepage,
http://www.pbspro.com, Stand von 02.04.2009.

[24] OpenPBS-Homepage,
http://www.openpbs.org, Stand von 02.04.2009.

[25] LSF-Homepage,
http://www.platform.com, Stand von 02.04.2009.

[26] NATIONAL CRASH ANALYSIS CENTER-Homepage,
http://www.ncac.gwu.edu, Stand von 02.04.2009.

Anhang

Die Tabelle zeigt den Inhalt der beigefügten CD.

Verzeichnis	Inhalt des Verzeichnisses
Zuteilungssoftware	Beinhaltet die implementierte Zuteilungssoftware
PBS	Beinhaltet zwei Verzeichnisse von PBS (PBS_EXEC, PBS_HOME) und eine PDF-Datei (Konfiguration) der die wichtigen Punkte der Konfiguration von PBS erläutert
LS-Dyna Jobs	Beinhaltet alle LS-Dyna Jobs mit ihren Inputs, Outputs und Ausführungszeiten
Bachealor	Beinhaltet die Bachealoararbeit als PDF und Latex

Abkürzungsverzeichnis

API	Application Programming Interface
BUS	Binary Unit System
CAD	Computer Aided Design
CPU	Central Processing Unit
FCFS	First-Come First-Served
FEM	Finite Element Methode
GUI	Graphical User Interface
HP-MPI	Hewllett-Packard-Message Passing Interface
IGES	Initial Graphics Exchange Specification
LS-Dyna	Livermore Software-Dyna
LSF	Load Sharing Facility
LSTC	Livermore Software Technology Corporation
MISD	Multiple Instruction - Single Data
MIMD	Muliple Instruction - Multiple Data
MPP	Massively Parallel Processing

RAM	Random Access Memory
RR	Round-Robin
SIMD	Single Instruction - Multiple Data
SISD	Single Instruction - Single Data
SJF	Shortest Job First
SMP	Symmetrisches Multiprozessorsystem
SNMP	Simple Network Management Protocol
SRTN	Shortest Remaining Time Next
STEP	Standard for the Exchange of Product Model Data
PBS	Portable Batch System
UMA	Uniform Memory Acces
WAN	Wide Area Network

Glossar

Algorithmus ist eine genau definierte Handlungsvorschrift zur Lösung eines Problems in endlich vielen Schritten.

Authentifizierung ist der Vorgang der Überprüfung einer behaupteten Identität, beispielsweise Eine Person oder eines Objekts, wie beispielsweise eines Computersystems.

Autorisierung bezeichnet die Zuweisung und Überprüfung von Zugriffsrechten auf Daten und Dienste an Systemnutzer. Die Autorisierung erfolgt meist nach einer erfolgreichen Authentifizierung.

Application Programming Interface (API): ist eine Programmierschnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.

Backfill ist ein Scheduling-Algorithmus für die Optimale Ausnutzung des Clusters durch Auffüllen von Lücken.

BUS ist in der Datenverarbeitung ein Leitungssystem mit zugehörigen Steuerungskomponenten, das zum Austausch von Daten und/oder Energie zwischen Hardware-Komponenten dient.

Cache bezeichnet in der Informatik eine Methode, um Inhalte, die bereits einmal vorlagen, beim nächsten Zugriff schneller zur Verfügung zu stellen. Caches sind als Puffer-Speicher realisiert, die die Kopien zwischenspeichern. Sie können als Hardware- oder Softwarestruktur ausgebildet sein.

Central Processing Unit (CPU) ist die zentrale Verarbeitungseinheit eines Computers, die in der Lage ist, ein Programm auszuführen.

Checkpointing speichert eine laufenden Job. Ein Job kann mittels eines Kommandos einmalig Oder auch periodisch checkpointed werden.

Cluster bezeichnet eine Anzahl von vernetzten Computern, die von außen als ein Computer gesehen werden können.

Harvard-Architektur bezeichnet in der Informatik ein Schaltungskonzept zur Realisierung besonders schneller CPUs und Signalprozessoren. Der Befehlsspeicher ist physisch vom Datenspeicher getrennt und beide werden über getrennte Busse angesteuert. Der Vorteil dieser Architektur besteht darin, dass Befehle und Daten gleichzeitig geladen, bzw. geschrieben werden können.

Heterogene Cluster können unterschiedliche Betriebssysteme oder Hardware eingesetzt werden.

InfiniBand ist eine Spezifikation zur Beschreibung einer seriellen Hochgeschwindigkeitsübertragungstechnologie.

Job Arrays ist ein Zusammenfassung von Jobs, die sich nur durch einer Indexparameter Unterscheiden.

Job Migration Unterbrechung laufende Prozesse, den Transfer auf einen anderen Rechner und die Wiederaufnahme.

Kerberos ist ein verteilter Authentifizierungsdienst (Netzwerkprotokoll), der für offene und unsichere Computernetze (zum Beispiel das Internet) zur Authentifizierung entwickelt wurde.

Load Balancing es kann vorkommen, dass ein Rechner überlastet ist, während andere warten oder nur schwach ausgelastet sind. Das System kann einen oder mehrere Jobs (Berechnung) am überlasteten Rechner Checkpointen und diese Jobs an anderen Rechner weiter laufen lassen

Load/Store-Architektur ist eine Computerarchitektur, deren Befehlssatz Daten-Speicherzugriffe ausschließlich mit speziellen Lade- (Load) und Speicher- (Store) Befehlen erlaubt.

Message Passing Interface (MPI) ist ein Standard, der den Nachrichtenaustausch bei parallelen Berechnungen auf verteilten Computersystemen beschreibt.

Massively Parallel Processing (MPP) bezeichnet in der Informatik die Verteilung einer Aufgabe auf mehrere Hauptprozessoren, die jeweils auch über eigenen Arbeitsspeicher verfügen können.

Symmetrisches Multiprozessorsystem (SMP) ist eine Multiprozessor-Architektur, bei der zwei oder mehr identische Prozessoren einen gemeinsamen Adressraum besitzen.

Simple Network Management Protocol (SNMP) ist ein Netzwerkprotokoll, um Netzwerkelemente z.B: Router, Server, Switches, Drucker, Computer usw. von einer zentralen Station aus Überwachen und steuern zu können.

Random Access Memory (RAM) ist ein Speicher, der besonders bei Computern als Arbeitsspeicher Verwendung findet.

Prozess ist in der Informatik ist ein ablaufendes Programm.

Prozessor ist eine Maschine oder eine elektronische Schaltung, welche gemäß übergebener Befehle andere Maschinen oder elektrische Schaltungen steuert. Am populärsten sind Prozessoren als zentrale Recheneinheiten von Computern, in denen sie Befehle von Software ausführen.

Von-Neumann-Architektur ist ein Referenzmodell für Computer, wonach ein gemeinsamer Speicher sowohl Computerprogrammbefehle als auch Daten hält.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 4. Mai 2009	
Ort, Datum	Unterschrift