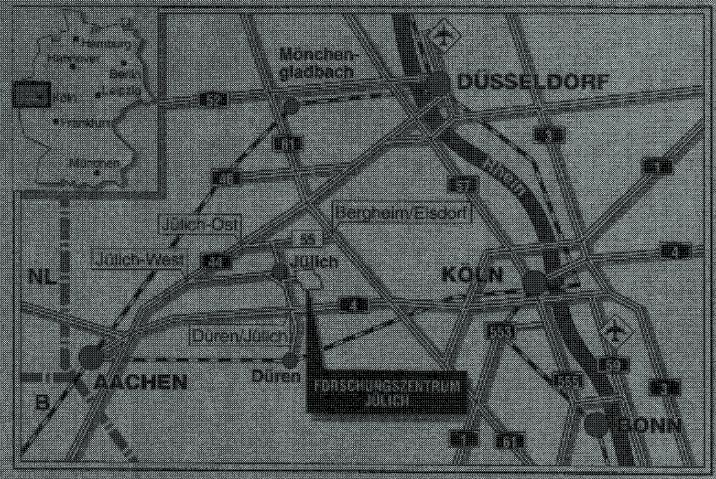
Forschungszentrum Jülich

Zentralinstitut für Angewandte Mathematik

Untersuchung von Strategien zum Job-Scheduling in massiv-parallelen Systemen

Daniel Mallmann

Jül-3458



Berichte des Forschungszentrums Jülich ; 3458
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-3458

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461/61-6102 · Telefax: 02461/61-6103 · e-mail: zb-publikation@fz-juelich.de

Untersuchung von Strategien zum Job-Scheduling in massiv-parallelen Systemen

Daniel Mallmann

Zusammenfassung

Der Job-Scheduler eines massiv-paralleles Systems, das Jobs im Batch-Betrieb ausführt, soll sowohl kurze Wartezeiten der Jobs als auch eine hohe Auslastung des Systems erzielen. In der vorliegenden Arbeit werden verschiedene Scheduling-Strategien auf diese Forderungen hin untersucht. Dabei wird die Zeitplanerstellung der Scheduling-Strategien für reale Arbeitslasten der im Forschungzentrum Jülich verfügbaren massiv-parallelen Rechnersysteme Intel Paragon XP/S 10 und Cray T3E simuliert. Die Ergebnisse der Simulation werden mit den Werten der realen Systembelegung verglichen.

Abstract

The job scheduler of a massively parallel processor which executes jobs in batch operation should achieve short job waiting times and a high load of the system. In this paper the different scheduling strategies are examined with regard to these requirements. To simulate the schedule the different strategies use the real workload of the massively parallel processors Intel Paragon and Cray T3E at Forschungzentrum Jülich. The results of the simulation are compared with the values of the real allocation of the system.

Inhaltsverzeichnis

1. Einleitung

Die Aufgabe eines Job-Schedulers besteht generell darin, zu entscheiden wann welcher Job ausgeführt wird. In einem massiv-parallelen System kommt als zusätzliche Schwierigkeit hinzu, daß viele dieser Entscheidungen gleichzeitig getroffen werden müssen, und der Job-Scheduler darüber hinaus den Ort festlegen muß, auf welchen Prozessoren welche Jobs ausgeführt werden sollen. Für diese Entscheidungen benötigt der Job-Scheduler Informationen über das Rechnersystem (zum Beispiel Knotenanzahl, reservierte Zeiten etc.) und Informationen über die einzelnen Jobs (zum Beispiel Laufzeit, Knotenzahl, Anzahl der parallelen Prozesse eines Jobs und Kommunikation zwischen den Prozessen).

In der vorliegenden Arbeit wird die Auswirkung der Scheduling-Strategie auf die Auslastung eines massiv-parallelen Rechners und auf die Wartezeiten der Jobs untersucht. Um die Untersuchung auf die Job-Scheduling-Strategien zu beschränken, werden zur Simulation der verschiedenen Strategien Job-Informationen realer Arbeitslasten der im Forschungzentrum Jülich verfügbaren massiv-parallelen Rechnersysteme Intel Paragon XP/S 10 und Cray T3E benutzt. Die Daten für die Intel Paragon erfassen alle Jobs, die zwischen September 1995 und Mai 1997 (21 Monate) ausgeführt wurden. Für die Cray T3E liegen die Jobinformationen für den Zeitraum von März 1997 bis Mai 1997 (3 Monate) vor. Mit Hilfe dieser Daten wird die Belegung der massiv-parallelen Systeme unter Anwendung der verschiedenen Scheduling-Strategien simuliert.

Die Zeitplanerstellung der auf den massiv-parallelen Systemen Intel Paragon und Cray T3E eingesetzten Scheduling-Systeme wird durch manuelles Eingreifen der Systemadministratoren optimiert. Dadurch erreichen diese Systeme eine sehr hohe Auslastung, die nur durch wenige automatisierte Job-Scheduling-Strategien erreicht werden kann. Für die manuellen Eingriffe in die Zeitplanerstellung nutzen die Systemadministratoren zum Beispiel Informationen über die Job-Laufzeiten die sie durch Rückfragen bei den Benutzern oder durch Beobachtung der bisherigen Laufzeiten der Programme erhalten. Diese Informationen stehen den untersuchten Job-Scheduling-Strategien natürlich nicht zur Verfügung.

Die aufgezeichneten Daten der massiv-parallelen Systeme Intel Paragon und Cray T3E enthalten lediglich Informationen über den Job, wie Knotenanzahl, Laufzeit und Ankunftzeit im System. Informationen über die einzelnen Prozesse eines Jobs werden nicht aufgezeichnet. Durch diese Informationen wird die Anzahl der zu

betrachtenden Scheduling-Strategien bereits stark eingegrenzt, da viele Strategien Informationen über die einzelnen Prozesse eines Jobs benötigen[1].

Die in der Simulation untersuchten Scheduling-Strategien sind Wartesysteme, die die Schlange der wartenden Jobs nach bestimmten Kriterien sortieren. Einige Strategien verwenden die Ausführungszeit der Jobs, um die Warteschlange zu sortieren. In den betrachteten massiv-parallelen Systemen steht die tatsächliche Ausführungszeit eines Jobs natürlich erst nach der Fertigstellung fest, so daß die Ergebnisse dieser Strategien nur eingeschränkt als real betrachtet werden können. Allerdings können diese Strategien eingesetzt werden, um den Einfluß des Sortierkriteriums „Ausführungszeit“ auf die Auslastung der Systeme und die Wartezeiten der Jobs zu beurteilen.

Die Auslastung der Rechnersysteme, die hier betrachtet wird, berücksichtigt aufgrund der vorhandenen Daten lediglich die Zuordnung von Knoten zu Jobs, das heißt die Auslastung der Knoten durch die einzelnen Jobs ist nicht enthalten. Es wird also nicht unterschieden, ob ein Knoten rechnet oder zum Beispiel an Synchronisationstellen auf andere Prozesse wartet.

In der Simulation erhält jeder Job die von ihm geforderte Zahl an Knoten, die bis zum Ende seiner Ausführung ausschließlich von ihm genutzt werden. Da auf der Intel Paragon die Frage nach dem Ort der Ausführung, die *Prozessorallokation*, keine Rolle spielt, können jedem Job Knoten in beliebiger Kombination zur Verfügung gestellt werden. Es entsteht durch diese Art der Partitionierung keine externe Fragmentierung. Auf der Cray T3E müssen die Knoten eines Jobs fortlaufend numeriert sein, wodurch externe Fragmentierung entstehen kann. Allerdings gibt es auf diesem System die Möglichkeit, durch Job-Migration freie Prozessorbereiche zusammenzufassen. Die Simulation der Systembelegung wird für die Cray T3E sowohl mit als auch ohne Job-Migration durchgeführt.

In den aufgezeichneten Daten der massiv-parallelen Systeme sind unter anderem die Übermittlungszeit jedes einzelnen Jobs an das System enthalten. Durch die Berücksichtigung dieses Wertes wird die Auslastung der Systeme beschränkt, deshalb werden für beide Systeme Belegungen mit und ohne Berücksichtigung der Übermittlungszeit simuliert und ausgewertet.

Neben den untersuchten Job-Scheduling-Strategien, die die Warteschlange der Jobs anhand von unterschiedlichen Merkmalen sortieren, werden noch weitere Strategien wie *Gang-Scheduling*, *Microeconomic-Scheduling* und *Lotterie-Scheduling* vorgestellt, die ebenfalls zu den Job-Scheduling-Strategien zu zählen sind. Diese Systeme werden jedoch nicht im Detail untersucht, da sie entweder keine reproduzierbaren Ergebnisse liefern (*Lotterie-Scheduling*) oder Informationen fehlen, die zu

[1] Verfahren, die alle Informationen über die Prozesse eines Jobs erhalten und als Kriterien zur Zeitplan-Bildung verwenden, werden *deterministische Scheduler-Modelle* genannt. Diese Modelle sind jedoch nur in seltenen Fällen realistisch (siehe [HB84] S. 592), da diese Informationen in realen Systemen zum Zeitpunkt der Entscheidung des Schedulers meistens nicht zur Verfügung stehen.

ihrer Simulation benötigt werden, wie zum Beispiel die Vergabe der Prioritäten eines Benutzers an seine Jobs (*Microeconomic-Scheduling*).

In Kapitel 2 werden zunächst *Partitionierungsverfahren* und *Strategien zum Job-Scheduling* vorgestellt. Kapitel 3 führt dann die Randbedingungen für die Simulation der Belegung der Intel Paragon (Abschnitt 3.1) und der Cray T3E (Abschnitt 3.2) ein. Anschließend folgt in Abschnitt 3.3 die Beschreibung der Implementierung des Simulators, der im Rahmen dieser Arbeit für die Untersuchung der Job-Scheduling-Strategien entwickelt wurde. Am Ende des Kapitels werden die Möglichkeiten der Visualisierung der simulierten Belegung vorgestellt. In Kapitel 4 werden die Ergebnisse der Simulation für die Intel Paragon und die Cray T3E ausgewertet und mit den aufgezeichneten Ergebnissen verglichen.

2. Partitionierung, Allokation und Job-Scheduling-Strategien

Dieses Kapitel gibt einen Überblick über räumliche (*Partitionierung und Allokation*) und zeitliche (*Scheduling*) Strategien für die Zuteilung von Knoten an Jobs. Aufgrund der Vielzahl der in der Literatur beschriebenen Verfahren kann in dieser Arbeit nur eine repräsentative Auswahl vorgestellt und beschrieben werden. Die Grenze zwischen zeitlichen und räumlichen Strategien wird von manchen Verfahren durchbrochen. So gibt es Ansätze, die beispielsweise *Gang Scheduling* (siehe Abschnitt 2.2.5) mit einer Neupartitionierung des MPP nach jeder Zeitscheibe verbinden (siehe [Feit95]).

2.1 Partitionierung und Allokation

Obwohl der Fokus dieser Arbeit auf der genaueren Betrachtung von Job-Scheduling-Strategien liegt, werden hier zunächst die Partitionierungsverfahren vorgestellt, da diese Verfahren die Scheduling-Strategien beeinflussen. Einen ausführlicheren Überblick über Partitionierungsverfahren und Allokations-Strategien bietet D.G. Feitelson in seiner Arbeit [Feit95]. Für die Partitionierung eines massiv-parallelen Systems gibt es drei verschiedene Ansätze, die sich durch die Dauer der Partitionierung unterscheiden. Die statische Partitionierung ändert die Partitionen zu fest vorgegeben Zeiten. Die variable und adaptive Partitionierung führen zum Ausführungsbeginn und -ende eines Jobs eine Neupartitionierung durch, und die dynamische Partitionierung ändert die Partitionsgrößen während der Laufzeit eines Jobs, in Abhängigkeit von der momentan benötigten Knotenzahl.

2.1.1 Feste/Statische Partitionierung

Für die *feste* oder *statische Partitionierung* wird das massiv-parallele System bei jedem Neustart des Systems partitioniert. Die Partitionen bleiben bis zur manuellen Änderung durch den Administrator bestehen, das heißt, daß die Knotenzahl der Jobs keinen Einfluß auf die Partitionierung hat. Zu den statisch partitionierten Systemen zählen auch Systeme, die tagsüber zwei Partitionen, eine für Batch-Betrieb

und eine für interaktive Jobs eingerichtet haben und nachts die gesamte Maschine in einer Partition für Jobs im Batch-Betrieb nutzen, da sie ebenfalls die Partitionen unabhängig von den Anforderungen der Jobs festlegen (siehe [Feit95]). Die statische Partitionierung hat mehrere Nachteile: zum einen kommt es zu interner Fragmentierung[1], zum anderen ist die maximale Knotenzahl, die ein Job beanspruchen kann, kleiner als die Gesamtzahl der Knoten des Systems. Dieses Verfahren wird auch als *Fixed Processors per Job Policy* (siehe [RSDS94]) bezeichnet.

Manche Implementierungen dieses Verfahrens erstellen für jede Job-Klasse eine Partition, sie werden *Multi-Class Fixed Partitioning* (siehe [NSS95]) genannt. Dies ist besonders dann sinnvoll, wenn alle Jobs anhand ihrer Größe in einige, wenige Klassen eingeteilt werden können[2].

Ein Extremfall ist die Erstellung einer einzigen Partition. In diesem Fall wird entweder nur eine Anwendung gleichzeitig auf dem MPP ausgeführt, oder in einer zweiten Ebene wird ein weiteres Partitionierungsverfahren eingesetzt, um die Fragmentierung zu vermindern.

2.1.2 Variable Partitionierung und adaptive Partitionierung

Die *variable Partitionierung* legt für jeden Job zu Beginn seiner Ausführung entsprechend der maximal benötigten Knotenzahl eine Partition fest. Diese Partition bleibt für die Dauer der Ausfürung des Jobs bestehen (siehe [Feit95]). Durch die Festlegung von Partitionen kann es dazu kommen, daß die restlichen, nicht zugewiesenen Knoten keinen der wartenden Jobs bedienen können. Dies wird als externe Fragmentierung bezeichnet. Häufig lassen die Rechnerarchitekturen nur Partitionen zu, deren Knotenzahl Potenzen von 2 sind, wodurch es zu interner Fragmentierung kommen kann.

Die *adaptive Partitionierung* legt ebenfalls die Partition für die Dauer der Ausführung eines Jobs fest. Der Unterschied zur variablen Partitionierung besteht darin, daß die Partitionsgröße nicht nur von der geforderten Knotenzahl, sondern auch von der Last abhängt (siehe [AWD96], [RSDS94], [SRDS93]). Eine lange Warteschlange führt beispielsweise dazu, daß einem Job weniger Knoten zugeteilt werden als er fordert. Außerdem wird, wenn möglich, jedem Job mindestens ein Knoten zugeteilt.

Für die adaptive Partitionierung gibt es, wie für die statische Partitionierung, die Möglichkeit das *Multi-Class Adaptive Partitioning* (siehe [NSS95]) einzuführen. Dazu wird der MPP zunächst statisch für die einzelnen Job-Klassen partitioniert und innerhalb der Partitionen wird eine adaptive Partitionierung vorgenommen.

[1] Interne Fragmentierung entsteht dann, wenn ein Job weniger Knoten benötigt als die ihm zugeteilte Partition umfaßt.

[2] Dies ist zum Beispiel der Fall, wenn alle Jobs entweder 8, 64 oder 256 Knoten benötigen und kein Job eine andere Knotenzahl fordert.

2.1.3 Dynamische Partitionierung

Das Verfahren der *dynamischen Partitionierung* ändert die Partitionsgröße während der Ausführungszeit des Jobs in Abhängigkeit von der momentan vom Job benötigten Knoten und der Auslastung des massiv-parallelen Systems (siehe [Squi95]). Durch die Änderung der Partitionsgröße während der Laufzeit in Abhängigkeit vom Parallelisierungsgrad des Programms gibt es keine interne Fragmentierung. Auch bei diesem Verfahren wird nach Möglichkeit jedem Job ein Knoten zugewiesen, wodurch jedem Job maximal ein Knoten zugewiesen wird, wenn mehr Jobs als Knoten im System sind. Wenn ein Job im Scheduler eintrifft, kann es durch die Zuweisung eines oder mehrerer Knoten zu einem hohen Aufwand beim Kontextwechsel kommen, da beim Wechsel eines Knotens aus einer Partition in eine andere alle Prozesse und Daten dieses Knotens auf die anderen Knoten der Partition verteilt werden müssen. Dadurch ergeben sich auch Einschränkungen im Programmiermodell. Die dynamische Partitionierung eignet sich besonders gut für Systeme mit gemeinsamem Speicher (Shared-Memory-Systeme), da dort der Overhead durch den Kontextwechsel wesentlich geringer ist als bei Systemen mit verteiltem Speicher (Distributed-Memory-Systeme).

2.1.4 Non-Work-Conserving Partitionierung

Die *Non-Work-Conserving* Partitionierung zeichnet sich dadurch aus, daß einige Knoten nicht allokiert werden, obwohl sie von einigen Jobs im System genutzt werden könnten, damit noch zu erwartende Jobs bearbeitet werden können ([RSSD95], [SRSD95]). Non-work-conserving wird zusammen mit der dynamischen oder der adaptiven Partitionierung genutzt und ist besonders für den interaktiven Betrieb geeignet.

2.1.5 Allokation

Die Knoten, die eine Partition bilden, können auf verschiedene Weise bestimmt werden. Grundsätzlich werden zwei Verfahren unterschieden:

- Die Knoten einer Partition müssen zusammenhängen (*Continous Processor Allocation*). In diesem Fall kann externe Fragmentierung entstehen. Der Vorteil ist, daß die Kommunikation zweier Knoten derselben Partition keine andere Partition belastet. Diese Allokations-Strategien werden in der Arbeit von J.H.T. Rottinghuis [Rott96] vorgestellt.
- Die Knoten einer Partition müssen nicht zusammenhängen (*Non-Continous Processor Allocation*). Dies hat den Vorteil, das durch die Allokation keine zusätzliche externe Fragmentierung entsteht, aber das Kommunikationsnetzwerk einer Partition wird unter Umständen von Knoten einer anderen Partition benutzt. Ausführlich werden diese Verfahren in [LWLN97] vorgestellt.

2.1.6 Partitionierung und Allokation auf Intel Paragon und Cray T3E

Die Partitionierung, Allokation und das Scheduling auf Intel Paragon und Cray T3E im Forschungszentrum Jülich werden vom *Network Queueing System* (*NQS*) des jeweiligen massiv-parallelen Systems durchgeführt. Auf beiden Systemen werden jedem Job die von ihm geforderte Zahl an Knoten zugeteilt, wie es bei der variablen Partitionierung der Fall ist.

Das System Intel Paragon befindet sich täglich von 20:00 Uhr bis 8:00 Uhr und an allen dienstfreien Tagen im Batch-Betrieb, das heißt, daß in dieser Zeit kein interaktiver Job ausgeführt wird. An Werktagen können morgens ab 8:00 Uhr interaktive Jobs gestartet werden. Alle Jobs, die im Batch-Betrieb gestartet wurden und um diese Zeit noch ausgeführt werden, laufen bis zu ihrer Fertigstellung weiter. Die Allokation der Knoten entspricht dem *Non-Continous Processor Allocation* Verfahren.

Auf dem massiv-parallelen System Cray T3E sind alle 518 Knoten täglich von 20:00 Uhr bis 7:00 Uhr und an allen dienstfreien Tagen für den Batch-Betrieb reserviert. Vormittags können von 7:00 Uhr bis 12:00 Uhr noch 383 Knoten, nachmittags ab 12:00 Uhr bis 20:00 maximal 4 Knoten von Jobs im Batch-Betrieb genutzt werden. Wie auf dem System Intel Paragon werden alle Jobs bis zu ihrem Ende ausgeführt, sie werden also bei der Verringerung der reservierten Knotenzahl um 7:00 Uhr und um 12:00 Uhr nicht unterbrochen. Am Vormittag werden Jobs mit geringer Zahl an Knoten ($\leq$ 64) bevorzugt. Die Zahl der für Batch-Betrieb reservierten Knoten wurde auf 383 festgelegt, um zu verhindern, daß nur zwei Jobs, einer mit 256 Knoten und einer mit 128 Knoten, das System Cray T3E zu dieser Zeit benutzen. Die Allokation der Knoten ist auf Cray T3E ähnlich dem *Continous Processor Allocation* Verfahren implementiert. Hier müssen die zugeteilten Knoten fortlaufend numeriert sein, das heißt aber nicht, daß die Knoten auch zusammenhängen. Dadurch ist der Nachteil des *Continous Processor Allocation* Verfahrens – externe Fragmentierung – auf dem massiv-parallelen System Cray T3E wirksam, der Vorteil der Kommunikation ohne Beeinträchtigung eines anderen Jobs wird jedoch nicht genutzt. Die externe Fragmentierung kann auf dem System Cray T3E verhindert werden, indem Jobs von einem Knoten-Bereich auf einen anderen migrieren. Dieses Verfahren wird in Abschnitt 3.2.7 beschrieben.

2.2 Job-Scheduling-Strategien

Massiv-parallele Systeme können sowohl räumlich als auch zeitlich unter den wartenden Jobs aufgeteilt werden. Bei der räumliche Aufteilung werden mehrere Jobs gleichzeitig auf verschiedenen Knoten ausgeführt. Job-Scheduling-Strategie, die das massiv-parallele System in dieser Art aufteilen, erstellen *Space Sharing Schemes* (siehe [SSG95]). Die zeitliche Aufteilung besteht darin, daß die Knoten eines massiv-parallelen Systems oder einer Partition jeweils für Zeitscheiben an unterschiedliche

Jobs vergeben werden. Der Ablaufplan der Scheduling-Strategien, die die zeitliche Aufteilung des massiv-parallelen Systems durchführen, wird als *Time Sharing Scheme* bezeichnet.

Die in den folgenden Abschnitten vorgestellten Job-Scheduling-Strategien erstellen *Space Sharing Schemes*, die Strategie *Gang Scheduling* erstellt ein gemischtes *Time & Space Sharing Scheme*.

Bei den meisten Job-Scheduling-Strategien (SPTF, SCDF, LJSF) besteht die Möglichkeit laufende Jobs zu unterbrechen (preemptive), wodurch jedoch ein hoher Aufwand für den Kontextwechsel entsteht. Deshalb werden in der Untersuchung nur nicht-unterbrechende (non-preemptive) Verfahren betrachtet.

2.2.1 First Come First Serve

Die Strategie *First Come First Serve* (*FCFS*) ist die einfachste der hier betrachteten Strategien. Die Jobs werden in der Reihenfolge ihres Eintreffens am Scheduler ausgeführt, ohne jegliche Änderung der Reihenfolge. Falls ein Job nicht genügend freie Knoten vorfindet, wartet er solange, bis laufende Jobs abgearbeitet wurden und die erforderliche Zahl an Knoten freigeben. Alle nachfolgenden Jobs werden hinter ihm in einer Warteschlange eingeordnet (siehe [LHR95]).

Eine Variante der FCFS-Strategie ist das *Lazy Scheduling*. Um freie Knoten, die dem ersten Job in der Warteschlange nicht ausreichen, zu nutzen, wird die Warteschlange nach einem Job durchsucht, der alle freien Knoten oder einen Teil davon nutzen kann (siehe [KMN96]). Bei diesem Verfahren kann es dazu kommen, daß durch das Vorziehen kleiner Jobs die übersprungenen Jobs später ausgeführt werden als dies bei der FCFS-Strategie der Fall wäre. Dadurch werden lange Jobs benachteiligt, kurze Jobs bevorzugt. Dieses Verfahrens kann leicht verbessert werden, indem nicht der erste Job zur Ausführung kommt, der einen Teil der freien Knoten nutzen kann, sondern der Job, der den größten Teil der freien Knoten nutzen kann.

Für die Job-Scheduling-Strategie *First Come First Serve* gibt es wie für die meisten anderen Job-Scheduling-Strategien eine Optimierung, die *Back-Filling* gennat wird. Die *Optimierung* der *FCFS* Strategie besteht darin, bestimmte Jobs vorzuziehen, falls ein oder mehrere Jobs die Warteschlange blockieren, weil sie nicht genügend freie Knoten vorfinden. Die vorgezogenen Jobs dürfen nicht mehr als die verbleibenden freien Knoten benötigen und ihre Ausführung darf den Startzeitpunkt der übersprungenen Jobs in der Warteschlange nicht verschieben. Dies bedeutet, daß ihre Ausführungszeit nicht länger sein darf als die Restausführungszeit der Jobs, die verhindern, daß der erste Job in der Warteschlange ausgeführt wird (siehe [LHR95]). Dieses Verfahren wird als *Back-Filling* (siehe [SSG95]) bezeichnet, da zuerst ein FCFS-Scheduling durchgeführt wird und anschließend die freibleibenden ($Knoten \times Zeit$)-Bereiche mit weiteren Jobs *aufgefüllt* werden. Dieses Verfahren bevorzugt zwar ebenso wie das Lazy-Scheduling Jobs mit geringer Knotenzahl, gleichzeitig werden aber große Jobs nicht benachteiligt.

Back-Filling kann bei den meisten Job-Scheduling-Strategien, die die Warteschlange anhand von Kriterien sortieren, angewandt werden. Neben der oben vorgestellten Variante des Back-Filling, die den ersten passenden Job der Warteschlange startet, gibt es eine Variante, die den Job mit der größten Zahl an geforderten Knoten startet, und eine weitere Variante, die aus allen wartenden Jobs die startet, die gemeinsam die meisten der freien Knoten belegen, wie bei der Job-Scheduling-Strategie *Best Package* (siehe Abbschnitt2.2.3). Diese drei Varianten der Job-Auswahl durch das Back-Filling werden in der Untersuchung betrachtet.

Da die exakte Ausführungszeit eines Jobs erst nach der Fertigstellung feststeht, kann Back-Filling auch mit Schätzwerten der Ausführungszeit durchgeführt werden. Dabei kann es jedoch vorkommen, daß der erste Job der Warteschlange später ausgeführt wird als es ohne Back-Filling der Fall wäre. In der Untersuchung der Job-Scheduling-Strategien wurden drei Fairneßbetrachtungen zum Back-Filling berücksichtigt (siehe Absschnitt 3.3.2). Die erste Variante entspricht dem Lazy Scheduling, es werden also keine Laufzeiten überprüft. In der zweiten Fairneßbetrachtung werden die exakten Laufzeiten bei der Job-Auswahl des Back-Filling betrachtet. Die dritte Variante nutzt eine Abschätzung der Job-Laufzeit zur Fairneßbetrachtung. Diese Abschätzung ist die maximale Laufzeit der Warteschlange des *NQS*, in die der Job eingereiht wurde.

2.2.2 Shortest Processing Time First, Largest Processing Time First

Die Laufzeit der Jobs ist ein Kriterium, daß zur Änderung der Reihenfolge in der Warteschlange genutzt werden kann. Für die Strategie *Shortest Processing Time First* (*SPTF*) werden die Jobs entsprechend ihrer exakten Ausführungszeit steigend sortiert (siehe [Feit95]). Sollten mehrere Jobs die gleiche Laufzeit haben, werden sie entsprechend ihrer Übermittlung an das System sortiert. Die Zahl der geforderten Knoten eines Jobs bleibt bei dieser Strategie zur Sortierung der Warteschlange unberücksichtigt. Jobs mit einer niedrigen Zahl geforderter Knoten werden jedoch durch *Back-Filling* bevorzugt. Da die Zahl der geforderten Knoten eines Jobs keinen Einfluß auf die Sortierung der Warteschlange hat, können Jobs mit einer hohen Zahl an geforderten Knoten über einen langen Zeitraum durch mehrere Jobs mit niedriger Zahl an Knoten und kürzerer Laufzeit an der Ausführung gehindert werden. Während dieser Zeit werden durch *Back-Filling* weitere Jobs mit geringer Knotenzahl gestartet, so daß zum Startzeitpunkt des Jobs mit hoher Knotenzahl nur wenige Jobs mit geringer Knotenzahl zum *Back-Filling* zur Verfügung stehen. Dadurch wird die Auslastung sinken.

Eine Abschätzung der Job-Laufzeit anhand der maximalen Laufzeit der Warteschlange des *NQS*, wie es beim Back-Filling eingeführt wurde, ist für die Auswahl der Jobs durch die Strategien ungeeignet, da die Warteschlangen nur wenige unterschiedliche maximale Laufzeiten haben. Die Realiserung dieser Strategie wird deshalb einige Schwierigkeiten bereiten.

Die Job-Scheduling-Strategie *Largest Processing Time First* (*LPTF*) sortiert die Warteschlange anhand der Laufzeiten der Jobs in fallender Reihenfolge. Dabei besteht die Gefahr, daß durch Jobs mit langen Ausführungszeiten und kleiner Knotenanzahl das massiv-parallele System für einen langen Zeitraum nicht voll ausgelastet wird. Dies führt zu einer niedrigen Auslastung (siehe [Feit95]).

Wie bei der *Shortest Processing Time First* Job-Scheduling-Strategie ist hier eine Abschätzung der Wartezeiten ebenfalls nicht sinnvoll, und die Realisierung der Strategie ist nur eingeschränkt möglich.

2.2.3 Smallest Job-Size First, Largest Job-Size First, Best Package

Die Strategie *Smallest Job-Size First* (*SJSF*) hat als Sortierungskriterium die Anzahl der vom Job geforderten Knoten. Die Warteschlange wird in steigender Reihenfolge sortiert. SJF wurde, wie auch die Strategie Shortest Processing Time, von der Strategie *Shortest Job First* für Einprozessorsysteme abgeleitet (siehe [Feit95]). Auf einem massiv-parallelen System wird diese Sortierung voraussichtlich zu starker Fragmentierung führen, da zunächst Partitionen für Jobs mit geringer Knotenzahl allokiert werden und anschließend zum Start der Jobs mit hoher Knotenzahl mehrere Jobs beendet werden müssen. Diese Jobs werden meistens nicht gleichzeitig beendet, so daß in dieser Zeit die Auslastung abnimmt, bis sie dann durch die Ausführung des Jobs mit hoher Knotenzahl wieder ansteigt.

Im Gegensatz zu allen anderen Strategien, die die Warteschlange sortieren, kann diese Strategie nicht durch Back-Filling optimiert werden, da alle Jobs der Warteschlange mehr oder genauso viele Knoten benötigen wie der erste Job der Warteschlange.

Wenn die Jobs nach der Anzahl geforderter Knoten in fallender Reihenfolge sortiert werden, kann die externe Fragmentierung gegenüber der FCFS Strategie verringert werden, da jeder beendete Job genügend Knoten für den nächsten Job frei gibt. Diese Idee liegt der *Largest Job-Size First* (*LJSF*) Strategie (siehe [SSG95], [Feit95]) zugrunde. Diese Strategie wird in Verbindung mit Back-Filling zu einer hohen Auslastung führen, denn Jobs mit geringer Knotenzahl, die durch Back-Filling gestartet werden, verhindern nicht, daß die Jobs mit hoher Knotenzahl ausgeführt werden können. Jeder Job, der nicht durch Back-Filling gestartet wurde, gibt nach seiner Fertigstellung genügend Knoten für die Ausführung des ersten Jobs der Warteschlange frei. Von diesen Knoten werden nur die durch Back-Filling genutzt, die nach dem Start des ersten Jobs der Warteschlange frei bleiben.

Die Strategie *Best Package* sortiert die Warteschlange ebenso wie die Strategie *Largest Job-Size First* fallend nach der geforderten Zahl der Knoten. Aus dieser sortierten Warteschlange werden dann sämtliche Kombinationen der Jobs gebildet. Die Kombination, die die meisten Knoten des massiv-parallelen Systems belegt, wird

gestartet. Sollten mehrere Kombinationen die gleiche Anzahl an Knoten belegen, wird die erste gefundene Kombination gestartet. Dadurch werden Jobs mit hoher Knotenzahl bevorzugt. Diese Job-Scheduling-Strategie läßt auf hohe Auslastungen hoffen, im Vergleich zur *Largest Job-Size First* Strategie mit Back-Filling besteht jedoch die Möglichkeit, daß eine Kombination die ausschließlich aus Jobs mit niedriger Knotenzahl besteht gestartet wird. Dadurch kann es vorkommen, daß anschließend die Jobs mit hoher Knotenzahl allein ausgeführt werden, da keine Jobs mit niedriger Knotenzahl zur Bildung einer Kombination zur Verfügung stehen. Back-Filling wird durch die Job-Scheduling-Strategie selbst ausgeführt. Sobald Knoten auf dem massiv-parallelen System frei werden, beginnt die Suche nach der Kombination, die die meisten der freien Knoten belegt.

2.2.4 Smallest Cumulative Demand First, Largest Cumulative Demand First

Eine weitere Abwandlung der Shortest Job First Strategie für Einprozessorsysteme ist die Strategie *Smallest Cumulative Demand First* (*SCDF*) (siehe [Feit95]), auch *Shortest Cumulative Demand First* (siehe [LV90]). Diese Job-Scheduling-Strategie sortiert die Jobs steigend nach dem Produkt $Knoten \times Ausführungszeit$. Ihr Vorteil besteht darin, daß Jobs mit geringer Knotenzahl und kurzen Laufzeiten bevorzugt werden. Für Jobs mit diesen Anforderungen werden häufig kurze Antwortzeiten erwartet. Der Nachteil dieser Strategie ist, daß durch die Bevorzugung der Jobs mit geringer Knotenzahl eine Fragmentierung entsteht, die auch mit Back-Filling nicht verringert werden kann.

Das Sortierkriterium der Strategie *Largest Cumulative Demand First* ist wie bei der Strategie *Smallest Cumulative Demand First* das Produkt $Knoten \times Ausführungszeit$. Hier werden die Jobs in fallender Reihenfolge sortiert. Auch diese Strategie wird zu einer hohen Auslastung des massiv-parallelen Systems führen, da Jobs mit hoher Knotenanzahl bevorzugt werden. Ihr Nachteil gegenüber der *Largest Job-Size First* Strategie ist, daß nicht ausschließlich Jobs mit hoher Knotenanzahl bevorzugt werden, sondern auch Jobs mit langen Laufzeiten. Dadurch können Jobs mit geringer Knotenzahl und langer Laufzeit gestartet werden, die dann wiederum Jobs mit hoher Knotenzahl an der Ausführung hindern. In Verbindung mit Back-Filling ohne Fairneßbetrachtung besteht die Gefahr, daß weitere Jobs mit geringer Knotenzahl gestartet werden, und das massiv-parallele System für die Ausführung von Jobs mit hoher Knotenzahl blockiert wird.

2.2.5 Co-Scheduling, Gang Scheduling

Co-Scheduling, in der Literatur auch als *Gang-Scheduling* bezeichnet, nutzt zur Erstellung des Zeitplans (*Schedule*) Informationen über die Abhängigkeit der Prozesse eines Jobs (siehe [Oust82]). Die Prozesse eines Jobs werden derart in Gruppen

(*Gangs*, *Taskforces*) eingeteilt, daß nur Prozesse innerhalb einer Gruppe miteinander kommunizieren. Die Gruppen aller Jobs werden vom Scheduler im *Timeslicing*-Verfahren auf die Knoten verteilt. J.K. Ousterhout stellt in seiner Arbeit [Oust82] drei Algorithmen zur Auswahl der gleichzeitig auszuführenden Gruppen vor. Dazu wird ein MPP mit P Knoten betrachtet. Die Zeit wird in Q Zeitscheiben (*Timeslot*, *Timeslice*) eingeteilt.

1. Die **Matrix Methode** teilt den MPP in ein Feld mit P Spalten und Q Zeilen auf. Jede Zeile entspricht einer *Zeitscheibe*, jede Spalte einem Knoten. Für jede Gruppe wird zeilenweise nach einem Platz gesucht.

2. Für den **kontinuierlichen Algorithmus** werden die Gruppen aller Jobs aneinandergereit. Die auszuführenden Gruppen werden mit einem Fenster der Länge P, das über die Liste der Gruppen gelegt wird, bestimmt. Wenn eine Zeitscheibe abgelaufen ist, wird das Fenster soweit nach rechts geschoben, bis die erste noch nicht ausgeführte Gruppe am linken Rand des Fensters liegt.

3. Der **nichtverteilende Algorithmus** entspricht weitgehend dem kontinuierlichen Algorithmus. Der Unterschied besteht darin, daß alle Gruppen eines Jobs aufeinanderfolgen müssen, um die Fragmentierung der Reihe nach der Beendigung eines Jobs kleinzuhalten.

Für die Matrix-Methode gibt es verschiedene Verfahren, um die Gruppen auf die Zeitscheiben zu verteilen (siehe [Feit96]). Diese Verfahren, auch *Packing* genannt, betreffen zum einen die Auswahl der Knoten, zum anderen die Auswahl der Zeitscheibe:

First Fit: Die Zeitscheiben werden vom Beginn an nach einem Platz für einen neuen Job durchsucht. Die erste Zeitscheibe, die genügend freie Knoten enthält wird benutzt.

Best Fit: Die Zeitscheiben werden entsprechend der Anzahl der freien Knoten steigend sortiert und anschließend durchsucht. Der Job wird der ersten Zeitscheibe, die genügend freie Knoten hat, zugeordnet. Dies ist gleichzeitig die Zeitscheibe die nach der Zuordnung die wenigsten frei bleibenden Knoten aller möglichen Zeitscheiben hat. Durch die Sortierung wird die Fragmentierung gegenüber dem First-Fit-Verfahren verringert.

Buddy Based Algorithm: Dieser Algorithmus teilt die Knoten nach dem Buddy-Prinzip logisch in Gruppen ein. Das Buddy-Prinzip wurde von der Speicherorganisation (siehe [HB84]) auf die Knotenzuteilung übertragen. Es lautet folgendermaßen: Für einen Job der Größe n wird ein Buddy der Größe $N = 2^{\lceil log n \rceil}$ gesucht. Falls kein Buddy der Größe N frei ist, wird ein Buddy der Größe $2 \times N$ in zwei Buddies der Größe N zerlegt. Umgekehrt werden zwei freie Buddies der Größe N zu einem Buddy der Größe $2 \times N$ zusammengefaßt. Für den *Buddy Based Algorithm* wird das gesamte massiv-parallele System rekursiv in Gruppen

der Größe 2^i eingeteilt. Jede Gruppe hat einen Controller, jeder dieser Controller hat zwei Untergruppen, so daß eine Hierarchie der Controller entsteht. Für jeden Job wird ein seiner Größe entsprechender Buddy gesucht. Sollten mehrere Buddies in Frage kommen, wird der Buddy mit der niedrigsten Last ausgewählt. Die Last eines Buddy ist die Summe aus drei Werten: die Anzahl der Jobs, die ihm zugeordnet sind, die maximale Last seiner Untergruppen, und die Summe aller Jobs der ihm übergeordneten Controller.

Der Overhead des Kontextwechsels ist schwer vorherzusagen, er fällt mit wachsender Länge der Zeitscheiben (siehe [FJ97]). Durch die Verwendung von Gang-Scheduling besteht die Möglichkeit Jobs auszuführen, deren geforderte Knotenzahl die Anzahl der Knoten des massiv-parallelen Systems übersteigt, soweit die Gruppen, in die der Job eingeteilt wird, nicht mehr Knoten benötigen als das System bietet.

2.2.6 Microeconomic Scheduler

Der *Microeconomic Scheduler* hat in gewisser Weise die freie Marktwirtschaft zum Vorbild (siehe [SAP95]). Sobald Knoten auf dem massiv-parallelen System frei werden, bieten die Jobs dem Scheduler *Geld* für die Knoten, und dieser entscheidet welcher Job zur Ausführung kommt. Der Begriff *Geld* steht hier vereinfachend für die Anteile die die Benutzer des massiv-parallelen Systems zur Verfügung gestellt bekommen. Das Wettbewerbssystem ist folgendermaßen gegliedert:

- Jeder Benutzer hat ein Sparkonto, auf das *Geld* mit einer zeitlich konstanten Rate fließt. Diese *Geld* kann er bis zu einem oberen Limit sparen. Das Verhältnis der monatlichen Geldzuflüsse an die einzelnen Benutzer entspricht der Aufteilung des $(Zeit \times Knoten)$-Feldes des massiv-parallelen Systems unter den Benutzern.

- Jeder Job, den ein Benutzer an den Scheduler weiterleitet, hat ein Spesenkonto, auf das *Geld* vom Sparkonto des Benutzers überwiesen wird. Neben einem Anfangsbetrag wird *Geld* mit einer zeitlich variablen Rate auf das Spesenkonto überwiesen. Diese variable Rate darf eine bestimmte Grenze nicht unterschreiten, damit der Job nicht endlos wartet.

- Sobald Knoten freigegeben werden, berechnet der Scheduler anhand der *Geldbeträge*, der geforderten Kontenzahl und der Laufzeit aller Jobs, welche Jobs ausgeführt werden.

Das einzige Ziel des Schedulers ist den höchstmöglichen *Geldbetrag* zu erwirtschaften. Dabei werden folgende Fälle berücksichtigt:

Fall 1: Durch die Auswahl des Jobs, der den höchsten *Betrag* pro Prozessorminute bietet, bleiben einige Knoten unbenutzt, weil alle anderen wartenden Jobs mehr Knoten benötigen als frei bleiben. Der *Microeconomic Scheduler*

prüft, ob er durch die Auswahl anderer Jobs, die einen niedrigeren Betrag pro Prozessorminute bieten, höhere Einnahmen erwirtschaften kann, indem er die Summe über die *Geldbeträge* aller Knoten des MPP bildet.

Fall 2: Ein großer Job bietet einen hohen *Betrag*, aber es werden nicht genügend Knoten auf einmal frei, um ihn auszuführen. In diesem Fall bietet der Scheduler dem Job an, bereits freie Knoten solange frei zu halten, bis weitere Jobs enden und genügend Knoten frei sind. Die freigehaltenen Knoten müssen allerdings auch für die reservierte Zeit bezahlt werden.

Fall 3: Ein Job gibt eine zu kurze Laufzeit an. Dieser Job wird entweder von den Knoten verdrängt oder er muß für die zusätzliche Zeit nachzahlen, wobei der *Betrag* anhand der neuen Last berechnet wird und direkt vom Sparkonto des Benutzers abgebucht wird, da das Spesenkonto des Jobs bereits aufgebraucht wurde.

Fall 4: Ein Job gibt eine zu lange Laufzeit an. In diesem Fall werden die Knoten nach der Fertigstellung des Jobs an die nächsten Jobs vergeben. Der gesamte *Geldbetrag* des Spesenkontos verfällt, es gibt keinen Rückfluß an den Benutzer.

Der *Microeconomic Scheduler* bietet viele Einstellungsmöglichkeiten, beispielsweise kann die Ressourcenverteilung über den Zufluß auf die Sparkonten der einzelnen Benutzer gesteuert werden, und jeder Benutzer kann über den *Geldzufluß* auf die Spesenkonten seiner Jobs die Prioritäten innerhalb des ihm zugeteilten Rahmens[3] vergeben.

Um den Preis, den ein Job pro Prozessorminute zahlen kann, zu berechnen kann auch der zukünftige, während der Ausführungszeit des Jobs zufliessende Geldbetrag miteinbezogen werden.

2.2.7 Lotterie Scheduling

Eine Weiterentwicklung des *Microeconomic Scheduler* ist das *Lotterie Scheduling* (siehe [Wal95]). In diesem System erhalten die Benutzer eine bestimmte Anzahl an Lotterielosen statt Geld. Dadurch können hier ebenfalls Prioritäten für die Benutzung des massiv-parallelen Systems gesetzt werden. Die Verlosung der freiwerdenden Knoten sorgt dafür, daß nicht ein einzelner Benutzer das gesamte System belegt. Über einen langen Zeitraum werden die Knoten jedoch entsprechend der Zahl der zugeteilten Lose aufgeteilt.

Für dieses Verfahren muß die Möglickeit bestehen Jobs zu unterbrechen, da jedes Los den gleichen Anteil am $Knoten \times Zeit$-Feld erhält, aber die Jobs unterschiedliche Knotenanzahlen fordern können. Aus Sicht der Auslastung des massiv-parallelen

[3] Dieser Rahmen wird durch die Verteilung der Geldbeträge auf die Sparkonten der einzelnen Benutzer bestimmt.

Systems ergeben sich gegenüber dem *Microeconomic Scheduler* keine Vorteile. Die Ergebnisse einer Simulation sind durch die Verlosungen, die mit Zufallszahlen durchgeführt werden, nicht reproduzierbar, deshalb wurde diese Strategie in der Untersuchung nicht betrachtet.

2.2.8 Prozeßorientierte Strategien

Neben den bisher vorgestellten Verfahren gibt es eine Vielzahl von Scheduling-Strategien, die die Zuordnung der einzelnen Prozesse oder Threads eines Jobs auf Knoten steuern. Für diese Strategien werden dem Scheduler alle Threads eines Jobs mit Informationen über ihre Laufzeit und die Abhängigkeiten untereinander übergeben und dieser entscheidet dann, wann ein Prozeß auf welchem Knoten ausgeführt wird.

Die Scheduling-Strategien, die in dieser Arbeit untersucht werden erhalten keine Informationen über die einzelnen Prozesse der Jobs, deshalb muß den Jobs für ihre gesamte Ausführungszeit die geforderte Zahl an Knoten als Partition zur Verfügung gestellt werden. Innerhalb dieser Partitionen werden dann die Prozesse nach einem internen Ablaufplan, der entweder vom Benutzer gesteuert oder durch den Compiler erstellt wird, auf die Knoten verteilt. Diese internen Ablaufpläne werden nach einer prozeßorientierten Strategie erstellt, die die Kommunikationsanforderungen der Prozesse berücksichtigt. Eine ausführliche Übersicht über diese Strategien, die aus verschiedenen Bereichen, wie zum Beispiel Produktionsplanung, Datenbanksysteme, Simulated Annealing oder VLSI-Design übertragen wurden, gibt F. Przybylski in seiner Arbeit [Przy92]. In diesem Bereich gibt es eine Reihe weiterer Arbeiten, die sowohl die Auslastung als auch die Fairneß der verschiedenen Verfahren untersuchen ([CV96], [MEB88]). Das Verfahren der *dynamischen Partitionierung* braucht ebenfalls Informationen über die Laufzeit der einzelnen Threads und deren Abhängigkeiten.

3. Die Simulation

In diesem Kapitel werden zunächst die Randbedingungen für die Simulation der Belegung der beiden massiv-parallelen Systeme Intel Paragon und Cray T3E vorgestellt. Anschließend folgt die Beschreibung der Implementierung des Simulators. In Abschnitt 3.4 wird die Darstellung der simulierten Belegung durch das Tool *VAMPIR* beschrieben.

Zur Analyse der Auslastung der massiv-parallelen Systeme und der Wartezeiten der Jobs können Daten der bereits betriebenen Systeme besser benutzt werden als zufällige künstliche Daten. Durch die Nutzung der Daten realer Lasten wird die Simulation auf die Scheduling-Strategien beschränkt, die Erstellung von Job-Profilen ist nicht erforderlich und beeinflußt somit nicht die Ergebnisse der Simulation.

Für das System Intel Paragon stehen die Daten des Abrechnungssystems von 21 Monaten (September 1995 bis Mai 1997) zur Verfügung. Auf dem massiv-parallelen System Cray T3E wurde erst Ende Februar 1997 der Produktionsbetrieb aufgenommen. Deshlab stehen von diesem System nur die Daten von 3 Monaten (März 1997 bis Mai 1997) zur Verfügung. Die Job-Informationen für das System Cray T3E werden aus den Daten extrahiert die das Tools *TREND* (Torus Resources and Node Display) (siehe [Moh96]) aufzeichnet.

3.1 Randbedingungen der Simulation für das System Intel Paragon

Auf dem System Intel Paragon XP/S 10 im Forschungszentrum Jülich werden alle Jobs an das *NQS* zur Ausführung übermittelt. Die Implementierung des *NQS* auf Intel Paragon weist einige Besonderheiten auf, die bei der Simulation der Scheduling-Strategien berücksichigt werden müssen. Diese Besonderheiten werden im Anschluß an die Übersicht der vom Abrechnungssytem aufgezeichneten Informationen erläutert.

3.1.1 Job-Informationen des Abrechnungssystems von Intel Paragon

Das Abrechnungssytem des massiv-parallelen Systems Intel Paragon im Forschungszentrum Jülich zeichnet von jedem ausgeführten Job folgende Informationen auf:

jobclass:	NQS-Warteschlange (*NQS-Queue*)
userid:	Zeichenkette, die zur Identifikation des Benutzers dient
aborted:	`1`, falls der Job abgeschlossen wurde, oder `abo`, falls er abgebrochen wurde
Nodes:	Anzahl der vom Job benötigten Knoten
submitted:	Datum und Uhrzeit bei der Übermittlung des Jobs an das *NQS*
start:	Datum und Uhrzeit bei Ausführungsbeginn
end:	Datum und Uhrzeit bei Ausführungsende
wait:	Wartezeit, Zeit zwischen Übermittlung (*Submitted*) und Ausführung (*Start*) des Jobs
run:	Laufzeit des Jobs

Die Uhrzeit wird in Stunden, Minuten und Sekunden angegeben, so daß die Simulation ebenfalls auf die Sekunde genau durchgeführt wird. Die Informationen *Jobclass*, *Userid*, *Nodes* und *Submitted* stehen bereits bei der Übermittlung des Jobs an das *NQS* zur Verfügung. Die Informationen *Start* und *Wait* werden zum Startzeitpunkt verfügbar, und die Informationen *End* und *Run* sind erst ab dem Ausführungsende eines Jobs bekannt. Die aufgezeichneten Daten werden in Dateien, die jeweils einen Kalendermonat umfassen, abgespeichert.

Jeder Job, der im Batch-Betrieb abgearbeitet werden soll, wird vom *NQS* je nach voraussichtlicher Laufzeit und Zahl angeforderter Knoten in eine Warteschlange (*Queue*) einsortiert, deren Bezeichnung (siehe Abschnitt 3.1.2) im Bereich `jobclass` abgelegt wird. Bei interaktiven Jobs erscheint hier: *interact*.

Interaktive Jobs werden sofort nach ihrer Übermittlung gestartet, falls genügend Knoten verfügbar sind. Andernfalls werden sie abgelehnt. Für die Simulation werden die interaktiven Jobs herausgefiltert, weil ihre Ausführung von der Scheduling-Strategie nicht beeinflußt wird.

3.1.2 NQS-Warteschlangen

Die Warteschlangen des *NQS* auf dem massiv-parallelen System Intel Paragon (siehe [DFK96]) sind in zwei Gruppen unterteilt:

1. Die *NQS Regular Queues*, sie haben das Format:
   p*nnn***h***tt*
   wobei **p***nnn* bedeutet, daß ein Job dieser Warteschlange maximal *nnn* Knoten benötigt. **h***tt* gibt an, daß die maximale Laufzeit eines einzelnen Jobs *tt* Stunden beträgt. Die Warteschlange **p***032***h***4* beispielsweise besagt, daß jeder dort einsortierte Job höchstens *32* Knoten benötigt und maximal *4* Stunden läuft.

   Eine Ausnahme unter den *Regular Queues* bildet die Warteschlange **p***004***h***1*. Diese Warteschlange ist für Test-Jobs vorgesehen. Jobs aus dieser Warteschlange dürfen jederzeit, also auch während der interaktiven Zeit (siehe Abschnitt 3.1.5), ausgeführt werden.

2. Die *NQS Sequential Queues*, deren Format
   p*nnn***h***tt***s** oder **p***nnn***h***tt***s2**
   lautet. Die Bezeichnung der Warteschlangen ist analog zu denen der *NQS Regular Queues*, mit dem Unterschied des Appendix **s** für *Sequential* und **s2** für *Sequential2*. Alle Jobs, die an eine dieser Warteschlangen übermittelt wurden, werden nacheinander ausgeführt. Es dürfen zu keiner Zeit zwei oder mehr Jobs aus einer *NQS Regular Queue* ausgeführt werden. Dies wird zum Beispiel benötigt, falls ein Job auf den Ergebnissen eines vorhergehenden Jobs aufbaut. Es gibt jeweils zwei Warteschlangen, damit zwei Benutzer unabhängig voneinander ihre Jobs ausführen lassen können.

3.1.3 Self-Submitting Jobs

Neben den *NQS Regular Queues* des Formates **p***nnn***h***tt* gibt es noch *NQS Regular Queues*, die das Format **p***nnn***e***tt* haben. Die Jobs in diesen Queues haben die Möglichkeit, bei ihrer Fertigstellung auf dem MPP einen neuen Job an die Queue, aus der sie gestartet wurden zu übermitteln. In den aufgezeichneten Daten des Abrechnungssystems sind die *Vorgänger* der Jobs nicht explizit genannt.

3.1.4 NQS-Beschränkungen

Ein einzelner, an das *NQS* übermittelter Job kann maximal 136 der 138 vorhandenen Knoten anfordern und bis zu 4 Stunden laufen. Um zu verhindern, daß ein Benutzer den gesamten MPP über einen längeren Zeitraum allein nutzt, gibt es eine obere Grenze von Prozessorstunden, die die übermittelten Jobs eines Benutzers nicht überschreiten dürfen. Diese Grenze beträgt

$$2 \times 136\ Knoten \times 4\ Stunden.$$

Falls ein Benutzer durch die Übermittlung seiner Jobs diese Grenze überschreitet, werden die nach der Überschreitung der Grenze verbleibenden Jobs auf die niedrigste Priorität gestuft und erst nach Ausführung eines Jobs eines anderen Benutzers auf ihre ursprüngliche Priorität hochgestuft. Daraus folgt, das ein Benutzer den MPP maximal 8 Stunden allein benutzen kann.

3.1.5 Batch-Betriebszeit

Jobs die ein Benutzer zur Ausführung im Batch-Betrieb an das *NQS* übermittelt hat, werden in der für den Batch-Betrieb reservierten Zeit ausgeführt. Während der Batch-Betriebszeit sind alle 138 Knoten für den Batch-Betrieb reserviert. Die Batch-Betriebszeiten des Systems Intel Paragon sind in Jülich wie folgt festgelegt:

werktags von 20:00 bis 8:00
samstags, sonntags, an Feiertagen und dienstfreien Werktagen ganztägig

Alle Jobs, die vom *NQS* gestartet wurden, werden bis zu ihrem Ende ausgeführt, dies gilt auch für Jobs, die an Werktagen noch kurz vor 8:00 Uhr gestartet wurden. Um zu verhindern, daß ein Job mit hoher Knotenanzahl den interaktiven Betrieb verhindert, darf ab 4:30 Uhr kein Job mit mehr als 64 Knoten ausgeführt werden. Die Anzahl der Jobs mit genau 64 Knoten ist auf eins begrenzt.

3.1.6 Ausfallzeiten des Systems Intel Paragon

Auf dem massiv-parallelen System Intel Paragon kommt es gelegentlich zu Systemunterbrechungen. Diese Systemunterbrechungen haben verschiedene Ursachen. Neben den planmäßigen Unterbrechungen für Wartungsarbeiten der Hardware und Software kommt es zu Unterbrechungen durch defekte Hardware und auf Grund von fehlerhaften Programmen beziehungsweise Fehlern im Betriebssystem. Die Ausfallzeiten des Systems Intel Paragon sind für den Zeitraum September 1995 bis Mai 1997 in einer gesonderten Datei aufgezeichnet und werden bei der Simulation der Belegung ebenfalls berücksichtigt. Die Jobs, die durch eine Systemunterbrechung abgebrochen werden, werden in der Simulation als nicht gestartet betrachtet und erneut in die Warteschlange einsortiert, damit sie zu einem späteren Zeitpunkt zur Ausführung gebracht werden können. Die Knoten, die die Jobs auf dem massiv-parallelen System bereits belegt haben, werden als unbenutzt betrachtet. Dadurch führen die Systemunterbrechungen zur Verringerung der Auslastung des Systems. Die Jobs, die in der Originalbelegung durch Systemunterbrechungen abgebrochen wurden, werden durch das Abrechnungssystem gekennzeichnet und für die Auslastung des massiv-parallelen Systems ebenfalls nicht berücksichtigt. Diese Jobs werden zur erneuten Ausführung an das *NQS* übermittelt.

3.2 Randbedingungen der Simulation für das System Cray T3E

Der Batch-Betrieb auf Cray T3E wird ebenso wie auf dem System Intel Paragon vom *NQS* verwaltet. Hier werden zunächst die für die Simulation benötigten Ereignisse, die das Tool *TREND* (siehe [Moh96]) aufzeichnet, erläutert. Anschließend folgen die Besonderheiten der Implementierung des *NQS* auf dem System Cray T3E.

3.2.1 Ereignisse des Tools TREND

Das Tool *TREND* (Torus Resources and Node Display) zeichnet in chronologischer Reihenfolge Ereignisspuren für Zustandswechsel in der Programmbelegung auf dem massiv-parallelen System Cray T3E auf. Für die Simulation wichtig sind die Ereignisse:

qenter: Ankunft eines Jobs am *NQS* mit den Informationen Zeitpunkt, Knotenanzahl, *NQS*-Queue, Job-Nummer und Benutzer

qstart: Start eines Jobs durch das *NQS*. Hierbei werden der Zeitpunkt, die Knotenanzahl, die Job-Nummer, der Benutzer und die Basis (der erste Knoten des Jobs auf dem System) aufgezeichnet

exit: Ausführungsende eines Jobs mit Zeitpunkt und Job-Nummer

down: Beginn einer Ausfallzeit

up: Ende einer Ausfallzeit

Die Zeitangaben der Ereignisse bestehen aus Stunden und Minuten. Die Belegung wird deshalb mit der Genauigkeit von $\pm$ 1 Minute simuliert.

Die Ereignisse *qenter*, *qstart* und *exit* müssen für jeden Job zusammengefaßt werden, um die für die Simulation notwendigen Job-Informationen zu erhalten.

Die Ereignisse *down* und *up* werden ebenfalls zusammengefaßt um sie bei der Simulation zu berücksichtigen.

Die aufgezeichneten Ereignisse jeweils eines Kalendermonats werden in einer Datei gespeichert.

3.2.2 NQS-Warteschlangen

Die Warteschlangen des *NQS* des massiv-parallelen Systems Cray T3E haben das Format

$$\mathbf{pe}nnn\mathbf{p}tt.$$

Hierbei entspricht *nnn* der maximalen Knotenzahl, die ein Job aus dieser Warteschlange benötigt. *tt* entspricht der maximalen Laufzeit der Jobs in Stunden. Neben diesen Warteschlangen gibt es noch sogenannte Express-Queues (Format: **pe***nnn***ex**), deren Jobs eine maximale Laufzeit von 10 Minuten haben.

3.2.3 Self-Submitting Jobs

Die Jobs aller *NQS-Warteschlangen* haben, ebenso wie auf dem System Intel Paragon, die Möglichkeit bei ihrer Fertigstellung auf dem MPP einen neuen Job an die Queue, aus der sie gestartet wurden zu übermitteln. In den Daten des Tools *TREND* wird der *Vorgänger* eines Jobs nicht bezeichnet.

3.2.4 NQS-Beschränkungen

Die maximale Anzahl an Knoten, die ein Job anfordern darf beträgt 512, die maximale Laufzeit 4 Stunden. Wie auf dem massiv-parallelen System Intel Paragon, gibt es auch auf Cray T3E ein Limit, das verhindern soll, daß ein Benutzer den gesamten MPP über einen längeren Zeitraum allein nutzt. Dieses Limit beträgt

$$2 \times 512 Knoten \times 4 Stunden.$$

3.2.5 Batch-Betriebszeit

Auf dem System Cray T3E gibt es neben dem reinen Batch-Betrieb während dem nur Batch-Jobs ausgeführt werden dürfen und dem interaktiven Betrieb noch den gemischten Betrieb während dem sowohl interaktive Jobs als auch Batch-Jobs ausgeführt werden. Während des gemischten Betriebs stehen den Batch-Jobs 383 der 518 Knoten zur Verfügung, damit gleichzeitig auch interaktive Jobs ausgeführt werden können ohne daß der Benutzer warten muß bis ein Batch-Job beendet wurde. In der Simulation wird die Anzahl der für den Batch-Betrieb reservierten Knoten entsprechend der folgenden Einteilung berücksichtigt:

werktags von	20:00 Uhr bis	7:00 Uhr:	518 Knoten für Batch-Betrieb
werktags von	7:00 Uhr bis	12:00 Uhr:	383 Knoten für Batch-Betrieb
werktags von	12:00 Uhr bis	20:00 Uhr:	4 Knoten für Batch-Betrieb
samstags, sonntags, an Feiertagen und dienstfreien Werktagen		ganztägig:	518 Knoten für Batch-Betrieb

Auf diesem System werden ebenfalls alle Jobs bis zu ihrem Ende ausgeführt. Esgibt also keine Job-Unterbrechungen beim Übergang vom Batch-Betrieb zum gemischten Betrieb. Um zu verhindern, daß ein Job mit 512 Knoten den interaktiven Betrieb bis 11 Uhr blockiert (7 Uhr + 4 Stunden maximale Job-Laufzeit) wird morgens die maximale Job-Größe von 512 Knoten über 256 Knoten auf 128 Knoten gesenkt wird. Die zeitlichen Beschränkungen sehen im einzelnen wie folgt aus:

werktags von	20:00 Uhr bis	4:00 Uhr:	512 Knoten maximale Job-Größe
werktags von	4:00 Uhr bis	9:00 Uhr:	256 Knoten maximale Job-Größe
werktags von	9:00 Uhr bis	12:00 Uhr:	128 Knoten maximale Job-Größe
werktags von	12:00 Uhr bis	20:00 Uhr:	4 Knoten maximale Job-Größe
samstags, sonntags, an Feiertagen und dienstfreien Werktagen		ganztägig:	518 Knoten maximale Job-Größe

3.2.6 Ausfallzeiten des Systems Cray T3E

Aus den Ereignissen *down* und *up* können die Ausfallzeiten des Systems Cray T3E berechnet werden. Diese Ausfallzeiten werden in der Simulation berücksichtigt, indem alle Jobs die zum Zeitpunkt des Ereignisses *down* ausgeführt werden, als nicht gestartet markiert und in die Warteschlange eingereiht werden. Alle Jobs, die laut *TREND* durch eine Sytemunterbrechung beendet wurden, werden bei der Simulation nicht beachtet.

3.2.7 Migration

Auf dem massiv-parallelen System Cray T3E müssen die Knoten, die ein Job zur Ausführung zugewiesen bekommt, fortlaufend numeriert sein. Dies bedeutet zwar nicht, daß die Knoten zusammenhängen müssen, aber die Auswirkungen dieser Einschränkung sind exakt die gleichen wie bei der kontinuierlichen Prozessor Allokation (*Continous Processor Allocation*, siehe Abschnitt 2.1.5), es kann also externe Fragmentierung entstehen. Um zu verhindern, daß durch externe Fragmentierung die Auslastung der Machine absinkt, besteht auf dem System Cray T3E die Möglichkeit, Jobs zu migrieren, daß heißt Jobs von einem Knoten-Bereich auf einen anderen Bereich zu verschieben. In der Untersuchung werden die Ergebnisse der Simulation sowohl mit als auch ohne Migration betrachtet. Dabei muß beachtet werden, daß die Zeit, die eine Migration in Anspruch nimmt, in den Werten für die Auslastung des Systems und in den Wartezeiten der Jobs nicht berücksichtigt wird. Die Zeit die ein Kontext-Wechsel auf einem Konten in Anspruch nimmt, ist nicht genau bekannt. Dieser Wert ist abhängig von der Größe des benutzten Speichers des Knotens und der Ein- und Ausgabeaktivität aller Jobs im Moment des Kontext-Wechsels. Erste Messungen haben einen durchschnittlichen Wert von ca. $\frac{1}{10}$ Sekunde pro Knoten ergeben. Dieser Wert muß jedoch durch weitere Messungen bestätigt werden.

3.3 Implementierung des Simulators

Zur Simulation der Belegung der massiv-parallelen Systeme Intel Paragon und Cray T3E wurde ein Simulator in der Programmiersprache C geschrieben. Die Simulation wurde auf dem System IBM SP2 des Forschungszentrums Jülich durchgeführt. Die Kommandozeile zum Aufruf des Simulators hat folgendes Format:

```
schedule {-paragon|-t3e|-t3emig} {-STRATEGY} {-BACKFILLING}
         [f:XX] [t:YY] {inputfile} {outputfile}
```

Die Option `STRATEGY` erwartet die Abkürzung der Job-Scheduling-Strategie (siehe Abschnitt 3.3.3) und die Option `BACKFILLING` die Abkürzung des Back-Filling-Verfahrens (siehe Abschnitt 3.3.2). Die Optionen `[f:XX]` und `[t:YY]` werden von manchen Back-Filling-Verfahren benötigt. Ihre Erläuterung folgt in Abschnitt 3.3.2. Die Eingabedatei (`inputfile`) ist entweder eine Datei des Abrechnungssystems des massiv-parallelen Systems Intel Paragon (der Schalter `-paragon` muß zusätzlich gewählt werden) oder eine Datei des Tools *TREND* (Schalter `-t3e` oder `-t3emig` wählen). Durch die Wahl des Schalters `-t3emig` wird die Simulation der Belegung des Systems Cray T3E mit Migrationen gewählt. In die Ausgabedatei (`outputfile`) werden die Informationen für die detailierte Darstellung durch das Tool *VAMPIR* geschrieben.

Der Simulator ist in mehrere Module unterteilt. Er besteht aus einem Grundpaket, welches das Einlesen und Ausgeben der Daten und die Arbeit auf den Datenstrukturen sowie deren Verwaltung übernimmt. Ein weiteres Modul übernimmt das Back-Filling aller Strategien. Jede Scheduling-Strategie ist in einem eigenen Modul implementiert.

3.3.1 Grundpaket

Das Grundpaket kann in drei Teile gegliedert werden:

1. Die aufgezeichneten Daten werden eingelesen, für die Simulation aufbereitet und in passende Datenstrukturen geschrieben.
2. Sämtliche von den Strategien für die Simulation benötigten Funktionen werden bereitgestellt.
3. Die durch die Simulation erzeugten Daten werden in dem von *VAMPIR* lesbaren Format als Tracefile abgespeichert und die statistische Auswertung der Daten wird in die beiden Textdateien geschrieben.

Neben diesen Aufgaben werden im Grundpaket einige Definitionen vorgenommen, die zum einen Werte der realen Umgebung wiederspiegeln, wie zum Beispiel `NUMBER_OF_CPUS`, `QUEUEFORMAT`, `SELFSUBMIT` oder `SINGLEQUEUE`, zum anderen erleichtern sie die Lesbarkeit der Programm-Codes, zum Beispiel werden die Integer-Werte `YES` und `NO` definiert, die als Ergebnisse von Funktionen geliefert werden, und die Bezeichnungen der Zustände eines Jobs, `NOTSUBMITTED`, `SUBMITTED`, `RUNNING` oder `READY` werden ebenfalls als Integer-Werte definiert, so müssen sie werder in einer Zeichenkette gespeichert werden, noch muß eine Zuordnungstabelle zwischen Integern und Zuständen geführt zu werden. Jede Job-Scheduling-Strategie und jeder Back-Filling-Algorithmus erhalten einen Integer-Wert. Der Wert der gewählten Job-Scheduling-Strategie und des gewählten Back-Filling-Algorithmus werden addiert und in einer Variablen gespeichert. Mit Hilfe dieser Variablen wird bei der Auswahl der Job-Scheduling-Strategie und des Back-Filling-Algorithmus in das entsprechende Modul verzweigt.

Die Eingabedateien des Simulators für die Belegung der Systeme Intel Paragon und Cray T3E enthalten einige Informationen, die für die Simulation nicht benötigt werden. Wenn eine Eingabedatei zum ersten Mal an den Simulator übergeben wird, schreibt der Simulator die für die Simulation notwendigen Informationen in eine neue Datei. Sollte der Simulator erneut mit dieser Eingabedatei aufgerufen werden (auch zur Simulation der Belegung mit einer anderen Job-Scheduling-Strategie), wird nur noch die neue Datei mit den notwendigen Informationen eingelesen. Die Informationen werden für die Simulation in ein Feld des Typs `JOB` eingelesen. Diese Feld, im weiteren Job-Feld genannt, enthält folgende Informationen über jeden einzelnen Job:

`submit_clock`:	Zeitpunkt der Übermittlung des Jobs an das *NQS*
`run_clock`:	Laufzeit des Jobs
`queue_run_clock`:	maximale Laufzeit eines Jobs in der Warteschlange
`end_clock`:	Zeitpunkt des Ausführungsendes
`start_clock`:	Startzeitpunkt
`org_start_clock`:	Zeitpunkt des Ausführungsbeginns in der Originalbelegung
`base`:	Erster durch den Simulator zugewiesene Knoten
`size`:	Zahl der geforderten Knoten
`jobid`:	Job-Nummer
`symbol_no`:	Nummer des Symbols zur Darstellung in *VAMPIR*
`userid`:	Benutzerkennung
`queue`:	Name der Warteschlange in die der Job vom *NQS* einsortiert wurde

`servant`: Job-Nummer des Nachfolgers, falls ein Nachfolger vorhanden ist, ansonsten der Wert `NO`

`status`: Integer-Wert für den Zustand des Jobs

Alle Zeitwerte werden in *Clocks* für die Darstellung mit *VAMPIR* umgerechnet, der Wert *0 Clocks* entspricht 0:00:00 Uhr am Montag vor der ersten Job-Übermittlung.

Zusätzlich zur Komponente `run_clock`, die die Laufzeit des Jobs enthält, wird in der Komponente `queue_run_clock` die maximale Laufzeit, die durch die Warteschlange in die der Job einsortiert wurde bestimmt wird, gespeichert. Dieser Wert kann von den Scheduling-Strategien zur Sortierung der Warteschlange benutzt werden, da er bereits bei der Einteilung des Jobs in eine Warteschlange durch das *NQS* bekannt ist.

In dem Job-Feld werden neben den Jobs auch die Systemunterbrechungen zur Erkennung der Ausfallzeiten gespeichert. Dazu wird der Beginn der Ausfallzeit als *Submitted*-Wert gespeichert. Das Ende der Ausfallzeit ergibt sich aus dem Wert `run_clock` der Systemunterbrechung.

Das Job-Feld wird nach den *Submitted*-Werten steigend sortiert, anschließend erhält jeder Job seinen Feld-Index als Job-Nummer. Um auf das Job-Feld zuzugreifen, wird eine doppelt-verkettete Liste erstellt. Die Strategien suchen in der sortierten Warteschlange der Jobs immer vom ersten Element bis zum letzten Element. Die Back-Filling-Verfahren suchen jedoch in beide Richtungen nach einem Job. Deshalb ist die doppelte Verkettung der Liste notwendig.

Die Komponente `org_start_clock` enthält den Zeitpunkt des Ausführungsbeginns eines Jobs in der Originalbelegung. Anhand dieser Werte werden Feiertage und dienstfreie Werktage, an denen kein interaktiver Betrieb stattfand, erkannt. Diese Tage werden in einem Integer-Feld namens `batchday` gespeichert und bei der Simulation berücksichtigt.

Um Jobs, die von einem anderen Job übermittelt wurden zu identifizieren wird zu jedem Job, der an eine *Self-Submitting Queue* übermittelt wurde, der Vorgänger gesucht. Ein Job, der von einem Vorgänger an das *NQS* übermittelt wurde, wird in der Simulation erst mit dem Ausführungsende des Vorgängers in die Warteschlange einsortiert. Dieser Zeitpunkt kann durch die Anwendung der verschiedenen Strategien wesentlich später sein als in der Originalbelegung. Würde die Reihenfolge dieser Jobs nicht beachtet könnten einige Strategien durch das gleichzeitige Starten der Jobs eine höhere Auslastung erzielen.

In der Komponente `status` wird der Integer-Wert des momentanen Zustands des Jobs gespeichert. Falls der Job in der Simulation noch nicht an das *NQS* übermittelt wurde steht hier `NOTSUBMITTED`, wenn er in der Warteschlange steht `SUBMITTED`, während der Ausführung `RUNNING` und nach der Fertigstellung `READY`.

Neben dem Job-Feld wird ein weiteres Feld vom Typ `SYMBOL` erstellt. Dieses Feld enthält die Informationen, die *VAMPIR* für die Darstellung der Symbole aller Jobs benötigt.

Nachdem alle Daten in die Felder eingelesen wurden, wird die in der Komandozeile angegebene Strategie aufgerufen. Folgende Werte werden an die Strategie übergeben:

`anchor`: Ein Zeiger auf den Beginn der doppelt-verketteten Liste, das Listenelement zeigt auf keinen Job.

`final`: Ein Zeiger auf das Ende der doppelt-verketteten Liste; dieses letzte Listenelement zeigt ebenfalls auf keinen Job.

`eventarray`: Ein Zeiger auf das Feld der Ereignisse, die von *VAMPIR* dargestellt werden sollen. Die Größe des Feldes ist zunächst der dreifache Wert der Job-Anzahl[1]. Bevor eine der Funktionen des Grundpakets ein Ereignis speichert, wird die Größe des `eventarray` überprüft und gegebenenfalls um den Wert `EVENTS_ADD` erweitert. Dies ist notwendig, da durch Systemunterbrechungen die Anzahl der Ereignisse steigt.

`events_total`: Die Anzahl der gespeicherten Ereignisse. `events_total` wird mit `1` initialisiert und nach dem Speichern eines neuen Ereignisses erhöht.

`events_max`: Die aktuelle Größe der Feldes `eventarray`.

`jobarray`: Das Feld, das alle Informationen über die Jobs enthält.

`strategy`: Der Integer-Wert der Strategie und des Back-Filling-Verfahrens.

`first_job_start_clock`: Der Startzeitpunkt des ersten Jobs in den Original-Daten. Dieser Wert wird benötigt, um den ersten Job in der Simulation zum gleichen Zeitpunkt wie im Original zu starten, damit ein fairer Vergleich der Zeiten möglich ist. Würde dieser Wert nicht berücksichtigt, wird der erste Job bereits direkt nach seiner Übermittlung ausgeführt, dadurch kann zum einen die Auslastung geringer werden, da zu Beginn der Simulation kein Back-Filling möglich ist, falls nicht mehrere Jobs gleichzeitig übermittelt werden. Zum anderen kann die Gesamtzeit der Ausführung aller Jobs verkürzt werden.

[1] Es wurde der dreifache Wert gewählt, da jeder Job drei Ereignisse verursacht: Ankunft, Start und Ende.

`batchday`: Das Feld enthält für jeden Tag einen Integer-Wert. Der Wert `NO` gibt an, daß dieser Tag Batch- und interaktiven Betrieb hat, der Wert `YES` gibt an, daß an diesem Tag ausschließlich Batch-Betrieb stattfindet.

`max_fragmentation`: Die maximale Fragmentierung, die durch Back-Filling entstehen darf. Dieser Wert wird nur in der Scheduling-Strategie *Largest Job First* und im Back-Filling-Verfahren *Best Combination* benutzt.

Die eingelesenen Jobs werden, wie oben bereits beschrieben, über eine doppelt-verkettete Liste aus Zeigern auf die Speicherstellen des Job-Feldes verwaltet. Für die Simulation sind Zeiger auf die doppelt-verkettete Liste notwendig, um den augenblicklichen Fortgang der Simulation zu speichern. Diese Zeiger sind:

`queue_end`: Der nächste Job, den ein Benutzer an das *NQS* übermittelt. Dieser Zeiger zeigt zu Beginn auf den ersten Platz der doppelt-verketteten Liste.

`queue_start`: Der nächste Job, der auf dem MPP zur Ausführung kommen soll. Die momentan zur Ausführung anstehenden Jobs liegen in dem Bereich zwischen `queue_start` und `queue_end`, wobei `queue_start` der erste Job in der Warteschlange ist und der Vorgänger von `queue_end` der letzte. In diesem Bereich wird die Reihenfolge der Jobs in der doppelt-verketteten Liste durch die unterschiedlichen Scheduling-Strategien geändert, so daß der Job, der zuletzt übermittelt wurde, nicht zwangsläufig der letzte Job in der Reihe der auszuführenden Jobs ist.

`exec_start`: Der erste Job in der Liste, der noch auf dem MPP ausgeführt wird. Im Bereich von `exec_start` bis `queue_start` befinden sich alle Jobs, die zur Zeit auf dem MPP ausgeführt werden. Da die Jobs unterschiedliche Laufzeiten haben, können sich in diesem Bereich zusätlich bereits beendete Jobs befinden. Ob ein Job noch ausgeführt wird oder bereits beendet wurde, wird durch die Komponente `status` im Job-Feld angezeigt.

Neben diesen drei Zeigern werden noch drei Zeitwerte für die zeitliche Einordnung der Ereignisse benötigt. Diese Zeitwerte sind:

`submit_clock`: Der Zeitpunkt, zu dem der nächste Job an das *NQS* übermittelt wird.

`start_clock`: Der Zeitpunkt, zu dem der nächste Job gestartet werden soll. Dieser Wert liegt nicht notwendiger Weise in der Batch-Betriebszeit, so daß dies vor dem Start eines Jobs überprüft werden muß.

`end_clock`: Der Zeitpunkt, zu dem der nächste der augenblicklich laufenden Jobs beendet wird.

Zuletzt wird noch ein Feld des Typs `NODE` zur Aufnahme der Zustände aller Knoten des massiv-parallelen Systems, kurz MPP-Feld genannt, eine Variable für die Anzahl der augenblicklich freien Knoten `free_cpus` und eine Variable des Typs `LIMIT`, welche zur Überwachung des *NQS-Limits* (siehe Abschnitt 3.1.4 für Intel Paragon und Abschnitt 3.1.4 für Cray T3E) dient, eingerichtet. Der Typ `NODE` hat folgende Komponenten:

`jobid`: Die Nummer des Jobs, der diesen Knoten momentan belegt; sollte der Knoten von keinem Job belegt sein steht hier `NO`.

`end_clock`: Das Ausführungsende des Jobs, der den Knoten momentan belegt (Startzeitpunkt + Laufzeit des Jobs).

`queue_end_clock`: Der Zeitpunkt zu dem der Job beendet würde, wenn er die maximale Laufzeit seiner *NQS-Queue* benötigt.

`queue`: Der Name der *NQS-Warteschlange*.

Die Komponente `queue_end_clock` wird von einigen der Back-Filling-Verfahren für die Fairneßbetrachtung benötigt (siehe Abschnitt 3.3.2). Für die Überprüfung der Regelung für die *NQS-Sequential-Queues* (Abschnitt 3.1.2) werden die *NQS*-Warteschlangen aller laufenden Jobs benötigt, die in der Kompnente `queue` abgelegt werden.

Für die Simulation werden den Scheduling-Strategie-Modulen folgende Funktionen zur Verfügung gestellt:

`submit_job`: Diese Funktion speichert die für die Darstellung des Ereignisses „Job-Übermittlung an das *NQS*“ wichtigen Informationen des übergebenen Jobs im Ereignisfeld. Des weiteren wird der Zeiger `queue_end` auf den nächsten am *NQS* eintreffenden Job gesetzt.

Die Variable `start_clock` erhält den Wert der Variablen `submit_clock`[2], falls dieser größer als der Wert `first_job_start_clock` ist.

Anschließend wird die Variable `submit_clock` auf die Übermittlungszeit des nächsten Jobs gesetzt.

Der Status des Jobs wird von `NOTSUBMITTED` auf `SUBMITTED` gesetzt.

[2]Durch das Setzen der Variablen `start_clock` wird als nächstes versucht, einen Job aus der Warteliste zu starten. Falls der Zeitpunkt nicht in der Batch-Betriebszeit liegt, muß der Wert der Variablen `start_clock` auf den Beginn der nächsten Batch-Betriebszeit gesetzt werden.

Die Funktion `submit_job` ist zusätzlich für die Ausfallzeiten der massiv-parallelen Systeme verantwortlich. Falls der Zeiger `queue_end` auf eine Systemunterbrechung zeigt, werden die Starteinträge aller Jobs, die sich zu diesem Zeitpunkt in der Ausführung befinden, im Ereignisfeld `eventarray` als Kommentar gekennzeichnet. Anschließend werden die Jobs über Zeiger wieder in die doppelt-verkettete Liste eingegliedert.

`start_job`: Mit Hilfe dieser Funktion wird der Ausführungsbeginn eines Jobs auf dem MPP simuliert. Vor dem Start des Jobs wird überprüft, ob das *NQS-Limit* überschritten wurde. Für das massiv-parallele System Intel Paragon wird zusätzlich kontrolliert, ob der zu startende Job aus einer *NQS Sequential Queue* (siehe Abschnitt 3.1.2) stammt. Sollte ein Job aus der gleichen *NQS Sequential Queue* bereits ausgeführt werden, wird der Job nicht gestartet, die Funktion `start_job` liefert den Wert `NO`. Die Simulation für das System Cray T3E sucht nun nach dem kleinsten zusammenhängenden Bereich, der genügend Knoten zur Aufnahme des Jobs hat. Sollte kein solcher *Slot* gefunden werden, wird der Job nicht ausgeführt, die Funktion `start_job` liefert den Wert `NO`. Falls Migration zugelassen wird, packt an dieser Stelle die in Ausführung befindlichen Jobs zusammen, damit ein Job, der weniger als die freien Knoten des Systems Cray T3E benötigt, zur Ausführung gebracht werden kann. Falls die genannten Bedingungen erfüllt werden, wird der Job gestartet. Die Funktion erwartet, daß der Job zum augenblicklichen Zeitpunkt gestartet werden darf, es wird also nicht überprüft, ob durch die Ausführung des Jobs eine Randbedingung wie zum Beispiel die Batch-Betriebszeit, überschritten wird.

Ist der gestartete Job der erste Job der Warteschlange, wird der Zeiger `queue_start` auf den nachfolgenden Job-Eintrag gesetzt. Andernfalls wurde ein Job durch Back-Filling, wegen Überschreitung des NQS-Limits oder aufgrund der *Sequential-Queue-Regelung* (siehe Abschnitt 3.1.2) aus der Reihe der wartenden Jobs gestartet.

Weiterhin wird die Anzahl der freien Knoten um die Größe des gestarteten Jobs verringert, im MPP-Feld werden die Komponenten `end_clock`, `job_id` und `queue` der entsprechenden Knoten gesetzt, und der Status des Jobs wird von `SUBMITTED` auf `RUNNING` gestuft.

Zuletzt wird der Wert der Variablen `end_clock` neu bestimmt, da der zuletzt gestartete Job möglicherweise vor den bereits laufenden Jobs beendet wird.

`end_job`: Diese Funktion gibt alle Konten des MPP-Feldes frei, deren Komponente `end_clock` dem oben eingeführten Zeitwert

`end_clock` entsprechen. Für jeden beendeten Job wird das Ereignis „Ausführungsende“ gespeichert.

Die Komponente `servant` der beendeten Jobs wird geprüft; falls sie ungleich `NO` ist, wird ein neuer Zeiger auf den `servant` in die doppelt-verkettete Liste vor dem Zeiger `queue_end` eingefügt. Der Zeiger `queue_end` wird anschließend auf den neuen Job gesetzt, und der `submit`-Wert des neuen Jobs wie auch die Variable `submit_clock`[3] erhalten den Wert der Variablen `end_clock`.

Zum Abschluß dieser Funktion wird der Wert der Variablen `end_clock` neu bestimmt.

`isbatchtime`: Diese Funktion liefert den Wert `YES`, wenn die übergebene Zeit in der reinen Batch-Betriebszeit liegt, während der keine Beschränkung für die Anzahl der Knoten eines Jobs besteht und der gesamte MPP für den Batch-Betrieb reserviert ist.

`nextbatchtime`: Ausgehend von der über den Funktionsaufruf mitgegebenen Zeit wird der Beginn der nächsten Batch-Betriebs-Phase ermittelt und als Ergebnis geliefert. Dabei werden Samstage, Sonntage und die aus den aufgezeichneteten Daten ermittelten Feiertage sowie dienstfreie Werktage anhand des ebenfalls übergebenen Feldes `batchday` berücksichtigt.

`limit_exceeded`: Diese Funktion überprüft, ob durch den Start des im Aufruf übergebenen Jobs das *NQS-Limit* überschritten wird. Diese Überprüfung wird anhand der übergebene Variable des Typs `LIMIT` durchgeführt, welche gleichzeitig um den Betrag an $Knoten \times Queuelaufzeit$ erhöht wird. Sollte der Wert höher als das *NQS-Limit* liegen, wird zusätzlich überprüft, ob ein Job eines anderen Benutzers wartet. Ist dies der Fall, liefert die Funktion den Wert `YES`, gleichzusetzten mit der Aussage, daß das Limit überschritten wurde, andernfalls liefert sie den Wert `NO`.

`queue_used`: Diese Funktion wird aufgerufen bevor ein Job gestartet wird. Es wird überprüft, ob der zu startende Job aus einer *NQS Sequential Queue* stammt. Sollte dies der Fall sein wird weiterhin überprüft, ob ein Job aus dieser *NQS Sequential Queue* bereits ausgeführt wird. Die Funktion liefert den Wert `NO` falls kein Job aus der untersuchten *NQS Sequential Queue* ausgeführt wird, ansonsten liefert sie `YES`.

`show_queue`: Die Funktion gibt zum Zeitpunkt ihres Aufrufs eine Liste der augenblicklich wartenden Jobs aus. Sie wird nach jeder Job-Übermittlung und vor jedem Start eines Jobs aufgerufen, damit die

[3] Durch das Setzen der Variablen `submit_clock` wird das nächste Ereignis die Übermittlung des neuen Jobs sein, weil alle anderen Ereignisse später stattfinden.

Entscheidungen der Scheduling-Strategie später in der Visualisierung mit *VAMPIR* kontrolliert werden können. Über die Integer-Variable `caller` wird der Aufrufer der Funktion identifiziert, dies sind entweder die Funktion `submit_job`, `start_job` oder eine der Back-Filling-Strategien.

Zur Ausgabe der Simulationsergebnisse wird das Feld `eventarray` von Element `1` bis zu Element `event_no-1` durchlaufen. Jedes Ereignis wird in dem von *VAMPIR* lesbaren Format (siehe [Arn96]) in dem Tracefile, das in der Komandozeile angegeben wurde, abgespeichert[4].

Neben diesem Tracefile werden zwei Textdateien mit statistischen Werten der Simultion erzeugt. Eine Datei enthält Informationen über die Ausführungszeiten der Jobs und die Belegung des MPP, die andere Datei enthält Informationen über die Wartezeiten der Jobs. Diese Dateien werden für jeden Monat erstellt, daß heißt, das alle Strategien, die für einen Monat die Belegung des MPP simuliert haben, in dieselbe Datei schreiben. Dadurch wird eine vergleichende Auswertung der Strategien möglich.

Eine weitere Textdatei wird erzeugt, wenn als Job-Scheduling-Strategie in der Kommandozeile `-orig` angegeben wird. Dadurch wird die Belegung des MPP nicht simuliert, sondern es wird die Original-Belegung der aufgezeichneten Daten ausgewertet. Die dabei zusätzlich erzeugte Datei enthält Informationen über die Verteilung der Laufzeit und der Zahl der angeforderten Knoten der Jobs.

3.3.2 Back-Filling

Die Scheduling-Strategien rufen, falls noch Knoten frei sind und der erste Job der Warteschlange nicht gestartet werden konnte, die Prozedur `backfilling` auf. In dieser Prozedur wird zunächst festgestellt, warum der erste Job in der Warteschlange nicht ausgeführt werden kann. Falls dies durch Überschreiten des *NQS-Limits* oder durch Blockierung eines Jobs einer *NQS Sequential Queue* geschieht, wird nach dem nächsten Job gesucht, der gestartet werden darf[5]. Sollte dieser Job genügend freie Knoten vorfinden wird er gestartet. Anschließend wird versucht, beginnend mit `queue_start`[6], weitere Jobs zu starten. Sollten danach noch Knoten frei sein, wird die in der Kommandozeile angegebene *Back-Filling-Strategie* gestartet. Die *Back-Filling-Strategien* arbeiten auf der von der Scheduling-Strategie bereits sortierten Warteschlange. Daß heißt, das die *First Fit Back-Filling-Strategie* beispielsweise nicht den ersten übermittelten Job findet, sondern den ersten Job in der bereits sortierten Warteschlange, bei der Scheduling-Strategie *Largest Job-Size First* ist dies

[4] Hierfür wird das zu *VAMPIR* gehörende *BINLIB*-Paket benutzt.

[5] Diese Suche entspricht dem Herunterstufen der Priorität der Jobs, die das *NQS-Limit* eines Benutzers überschreiten und dem Entblocken der *NQS Sequential Queues*.

[6] Dadurch wird berücksichtigt, daß durch den Start eines Jobs das *NQS-Limit* für heruntergestufte Jobs wieder ausreicht, um sie zu starten.

gleichzeitig der größte ausführbare Job. Sollte ein Job durch *Back-Filling* gestartet werden, so wird dieser Job aus der Reihe der wartenden Jobs in der doppeltverketteten Liste herausgenommen, und vor dem Job auf den `queue_start` zeigt, eingefügt, wodurch er der letzte der momentan laufenden Jobs ist.

Folgende Back-Filling-Verfahren sind implementiert:

First Fit: Die *First Fit Back-Filling-Strategie* (Abkürzung für die Kommandozeile: `ff0`) sucht in der Reihe der wartenden Jobs von `queue_start` bis `queue_end` nach dem ersten Job, der nicht mehr als die augenblicklichen freien Knoten benötigt und der sowohl das *NQS-Limit* einhält als auch die *Sequential-Queue-Regelung* beachtet. Sollten nach dem Start dieses Jobs noch Knoten auf dem MPP frei sein, wird nach dem nächsten Job gesucht, der wiederum mit der nun geringeren Anzahl an Knoten auskommt. Diese Suche wird solange weitergeführt, bis das Ende der Warteschlange erreicht ist oder kein Knoten des MPP frei ist.

Die erste Erweiterung dieser Strategie um eine Fairneßbetrachtung (Abkürzung: `ff1`) überprüft, nachdem mittels *First Fit Back-Filling* ein Job ermittelt wurde, ob der ermittelte Job die Ausführung eines der Jobs, die vor ihm in der Warteschlange stehen, auf einen späteren Zeitpunkt verschiebt. Dazu wird eine Vorschau auf das Scheduling mit der momentanen Systembelegung[7]und der vorhandenen Warteschlange bis zu dem ermittelten Job erstellt, ohne den ermittelten Job selbst zu starten. Sollten während dieser Vorschau weniger Knoten frei bleiben als der ermittelte Job benötigt, so wird dieser Job nicht als fair betrachtet und somit nicht gestartet. Um die Vorschau auf das Scheduling zu erstellen werden die tatsächlichen Laufzeiten der Jobs benutzt. Fas *Back-Filling* mit dieser Fairneßbetrachtung ist dadurch in der Realität nicht möglich ist.

Die zweite Erweiterung des *First-Fit-Back-Filling* um eine Fairneßbetrachtung (`ff2`) orientiert sich an der maximalen Laufzeit der Warteschlange eines Jobs. Diese Betrachtung hat den Vorteil, daß sie in der Realität implementierbar ist. Der Nachteil ist, daß die maximale Laufzeit einer Queue häufig keinen Rückschluß auf die tatsächliche Laufzeit eines Jobs zuläßt und somit kein faires *Back-Filling* entsteht. Die Vorschau wird ebenso wie bei der

[7]Die für die Erstellung der Vorschau benötigten Werte, die Endzeiten der laufenden Jobs, werden beim Start eines Jobs in das MPP-Feld geschrieben.

Erweiterung `ff1` erstellt, mit dem Unterschied, daß statt der tatsächlichen Laufzeiten die maximalen Laufzeiten der Warteschlangen den Endzeitpunkt der laufenden und wartenden Jobs festlegen[8].

First Come: Für die *First-Come-Back-Filling-Strategie* (Abkürzung: `fc0`) wird die Warteschlange nach dem *Submitted*-Wert steigend sortiert. Anschließend wird die Warteschlange, wie bei der *First-Fit-Back-Filling-Strategie*, nach einem Job durchsucht, der höchstens soviel Knoten anfordert wie frei sind. Das weitere Vorgehen ist analog zu der *First-Fit-Back-Filling-Strategie*. Für diese Strategie sind ebenfalls beide Fairneßbetrachtungen (`fc1` und `fc2`) implementiert.

Best Size: Die Suche nach dem Job, der die meisten der freien Knoten nutzt, wird als *Best Size Back-Filling* (Abkürzung: `bs0`) bezeichnet. Für diese Back-Filling-Strategie wird die Warteschlange entsprechend dem Wert der geforderten Knoten fallend sortiert. Wiederum sind beide Fairneßbetrachtungen (`bs1` und `bs2`) implementiert.

Best Demand: Diese Back-Filling-Strategie (Abkürzung: `bd0`) sucht den Job mit dem größten Produkt $Knoten \times Zeit$. Die Warteschlange wird entsprechned dem Produkt aus der Zahl angeforderter Knoten multipliziert mit der Laufzeit sortiert. Zur Bestimmung des Produktes $Knoten \times Zeit$ wird die tatsächliche Laufzeit des Jobs benutzt, wodurch diese *Back-Filling-Strategie* nur eingeschränkt realisierbar ist.

Als Erweiterung wird nur die Fairneßbetrachtung `bd1`, die sich an den tatsächlichen Laufzeiten orientiert, angeboten da eine Fairneßbetrachtung unter Zuhilfenahme der maximalen Laufzeit der Warteschlangen in Verbindung mit der hier verwendeten Bestimmung des Produktes $Knoten \times Zeit$ nicht sinnvoll erscheint.

Best Queue Demand: Die *Best-Queue-Demand-Back-Filling-Strategie* (Abkürzung: `bqd0`) sucht ebenfalls den Job mit dem größten Produkt $Knoten \times Zeit$. Im Unterschied zur *Best Demand Back-Filling-Strategie* benutzt sie die maximale Laufzeit der Warteschlange, um das Produkt $Knoten \times Zeit$ eines Jobs zu bestimmen, wodurch diese Strategie uneingeschränkt realisierbar ist.

[8] Beim Start eines Jobs wird die maximale Laufzeit der Warteschlange zu der Startzeit addiert und in der Komponente `queue_end_clock` des MPP-Feldes gespeichert.

Für diese Strategie wurde nur die Erweiterung `bqd2` implementiert, die die Fairneß anhand der maximalen Laufzeiten der Warteschlangen beurteilt.

Best Combiantion: Diese Back-Filling-Strategie (Abkürzung: `bc`) sucht aus allen Jobs der Warteschlange die Kombination von Jobs, die die meisten der freien Knoten belegt. Für diese Strategie wurde keine Fairneßbetrachtung implementiert, aber es besteht die Möglichkeit, durch Angabe der maximalen Fragmentierung in der Kommandozeile festzulegen, welche Fragmentierung nach dem Back-Filling noch bestehen bleiben darf. Sollte das Back-Filling mehr Knoten frei lassen, als die maximale Fragmentierung zuläßt, wird die gefundene beste Kombination von Jobs nicht gestartet. Dieses Vorgehen hat den Vorteil, daß ein großer Job, der entsprechend der Scheduling-Strategie als nächster zur Ausführung kommen soll, nicht durch langes ineffektives Back-Filling an der Ausführung gehindert wird.

Worst Size: *Worst-Size-Back-Filling* (Abkürzung: `ws0`) bedeutet, daß der Job mit der niedrigsten Zahl geforderter Knoten als erster zum *Back-Filling* benutzt wird. Diese Strategie wurde lediglich zu Abschätzungszwecken implementiert. Für diese *Back-Filling-Strategie* wurden keine Fairneßbetrachtungen erstellt.

3.3.3 Scheduling-Strategien

Die Scheduling-Strategien sind alle in gleicher Art und Weise implementiert. Zu Beginn werden die Zeiger `queue_start`, `queue_end` und `exec_start` auf den ersten Eintrag in der doppelt-verketteten Liste gesetzt, den Nachfolger von `anchor`. Die Variable `submit_clock` erhält die Übermittlungszeit des ersten Jobs, `end_clock` wird auf 50 Wochen gesetzt, und `start_clock` erhält den Wert `first_start_clock`, damit die Ausführung des ersten Jobs zur gleichen Zeit beginnt wie in den aufgezeichneten Daten.

Nach der Initialisierung wird eine Schleife solange durchlaufen, bis die Variable `queue_start` den Wert `final`[9] enthält und alle Knoten frei sind[10]. Innerhalb der Schleife wird als erstes die Variable `submit_clock` mit den beiden anderen Zeitwerten verglichen. Sollte `submit_clock` der kleinste der drei Werte sein wird der Job, auf den `queue_end` zeigt, durch Aufruf der Prozedur `submit_job` aus dem Grundpaket gestartet.

[9] Das bedeutet, daß alle Jobs auf dem MPP gestartet wurden.

[10] Diese Bedingung muß erfüllt sein, damit nach dem Start des letzten Jobs die Aufzeichnung der Siimulation solange fortgeführt wird bis alle Jobs ihre Ausführung beendet haben.

Als nächstes wird der Wert der Variablen `end_clock` überprüft. Falls dieser der kleinste der drei Zeitwerte ist wird die Prozedur `end_jobs` aufgerufen.

Abschließend wird auch der Wert der Variablen `start_clock` mit den beiden anderen Zeitwerten verglichen. Sollte dies der kleinste Wert sein, so wird mit seiner Hilfe die augenblickliche Größe der Batch-Partition (Cray T3E) und die maximale Job-Größe (Intel Paragon und Cray T3E) bestimmt. Dann wird die Warteschlange von `queue_start` bis zum Vorgänger von `queue_end` entsprechend der gewählten Strategie sortiert. Anschließend werden beginnend mit `queue_start` Jobs gestartet. Falls ein Job der Warteschlange nicht gestartet werden, kann wird die Prozedur `backfilling` aufgerufen. Nach Abschluß der Prozedur wird der Wert der Variablen `start_clock` auf den niedrigsten der drei Werte `end_clock`, `submit_clock` oder nächster Beginn der Batch-Betriebszeit gesetzt. Somit ist gewährleistet, das nach jeder Übermittlung und nach jedem Ausführungsende eines Jobs versucht wird Jobs der Warteschlange zu starten.

Aufdem massiv-parallelen System Intel Paragon besteht nun noch die Möglichkeit, daß der Wert der Variablen `start_clock` kleiner als die anderen beiden Zeitwerte ist, aber nicht in der Batch-Betriebszeit liegt. In diesem Fall wird die Warteschlange nach Jobs durchsucht, die am Tag gestartet werden können. Falls möglich werden diese gestartet und ebenso wie beim *Back-Filling* in der doppelt-verketteten Liste vor dem `queue_start` plaziert.

Folgende Scheduling-Strategien wurden implementiert:

FCFS: *First Come First Serve* soritert die Warteschlange anhand der Übermittlungszeiten der Jobs an das *NQS*.

SPTF: *Shortest Processing Time First* sortiert die Warteschlange aufsteigend nach der tatsächlichen Laufzeit der Jobs.

LPTF: *Largest Processing Time First* sortiert die Warteschlange absteigend nach der tatsächlichen Laufzeit der Jobs.

SJSF: *Smallest Job-Size First* sortiert die Warteschlange aufsteigend nach der Zahl angeforderter Knoten.

LJSF: *Largest Job-Size First* sortiert die Warteschlange absteigend nach der Zahl angeforderter Knoten. Für diese Strategie können zusätzlich die Argumente `f:XX` und `t:YY` in der Kommandozeile für das Back-Filling angegeben werden. Der Wert `f:XX` gibt an, daß die Fragmentierung durch Backfilling maximal `XX` Prozent des massiv-parallelen Systems betragen darf. Durch die Wahl der Option `t:YY` wird das Back-Filling beendet, falls seit über `YY` Stunden Back-Filling durchgeführt wird und der erste Job der Warteschlange nicht genügend freie Knoten vorfindet. Nach jedem Back-Filling wird kontrolliert, ob die Fragmentierung höher ist als die maximale Fragmentierung zuläßt, oder ob die Zeit von `YY` Stunden abgelaufen ist. Sollte eines der beiden Kriterien zutreffen, so wird das Back-Filling eingestellt,

bis der erste Job der Warteschlange ausgeführt wurde. Falls diese Werte nicht gesetzt werden, werden die Kriterien nicht überprüft.

BP: *Best Package* wählt die Kombination von Jobs aus, die die meisten Knoten belegt. Diese Strategie wurde ohne Back-Filling implementiert, da nach dem Start der besten Kombination kein Job mehr vorhanden sein kann, der weniger als die verbleibenden Knoten anfordert und gestartet werden darf. Die Suche nach der besten Kombination erfordert die Berechnung der belegten Knoten für maximal $n!$ Kombinationen, wenn n Jobs in der Warteschlange sind. Die Zahl der Kombinationen die überprüft werden müssen ist jedoch meistens geringer. Wenn eine Kombination a von Jobs weniger Knoten belegt als die bislang beste Kombination b wird der letzte Job aus der Kombination a entfernt. Anschließend wird, beginnend mit dem nächst kleineren Job, die Suche nach der besten Kombination mit der Kombination a weitergeführt bis das Ende der Warteschlange erreicht wurde.

SCDF: *Smallest Cummulative Demand First* sortiert die Warteschlange aufsteigend nach dem Produkt $Knoten \times Laufzeit$ der Jobs.

LCDF: *Largest Cummulative Demand First* sortiert die Warteschlange absteigend nach dem Produkt $Knoten \times Laufzeit$ der Jobs.

3.4 Visualisierung der simulierten Belegung

Der Simulator speichert für jede simulierte Belegung der massiv-parallelen Systeme durch die verschiedenen Job-Scheduling-Strategien detailierte Informationen in einem Tracefile für das Visualisierungs-Tool *VAMPIR*. Dieses Tool wurde zur Darstellung zeitlicher Abläufe innerhalb paralleler Programme auf Multiprozessorsystemen entwickelt. Damit kann die Belegung eines massiv-parallelen Systems mit 512 Prozessoren für einen Monat vollständig dargestellt werden. Die Darstellungsmöglichkeiten und die Bedienung des Tools sind ausführlich in [Arn93] und [AR96] erläutert. In diesem Abschnitt wird beschrieben, wie die Entscheidungen des Job-Schedulers mit Hilfe des Tools *VAMPIR* überprüft und die Belegungen der massiv-parallelen Systeme dargestellt werden können.

Die für die Untersuchung der Strategien wichtigen Ansichten des Tools *VAMPIR* sind:

- Summaric Activity Chart
- Parallelism View
- Global Timeline

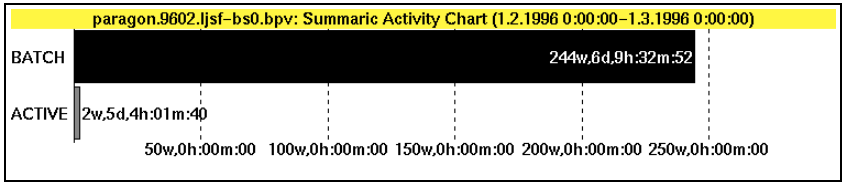


Abbildung 3.1: VAMPIR: Summaric Activity Chart

Die Ansicht *Summaric Activity Chart* (Abblidung 3.1) zeigt die Summen der einzelnen Betriebsarten. Die Werte können entweder als Zeiten oder als Prozentsätze angezeigt werden.

In der Simulation werden die Betriebsart *BATCH* und die aktive Zeit *ACTIVE* für die Darstellung aufbereitet. Der Wert der Betriebsart *BATCH* in Abbildung 3.1 hat die Einheit $Prozessoren \times Zeit$. Die aktive Zeit *ACTIVE* wird als absolute Zeit angegeben. Mit Hilfe dieser Werte kann die Auslastung des massiv-parallelen Systems bestimmt werden. Die Auslastung während der aktiven Zeit beispielsweise, kann mit den Werten *BATCH* und *ACTIVE* für das System Intel Paragon folgendermaßen bestimmt werden:

$$l_{act} = \frac{BATCH}{138 \times ACTIVE}$$

Für das massiv-parallele System Cray T3E muß im Nenner statt der 138 die Prozessorzahl 518 eingesetzt werden.

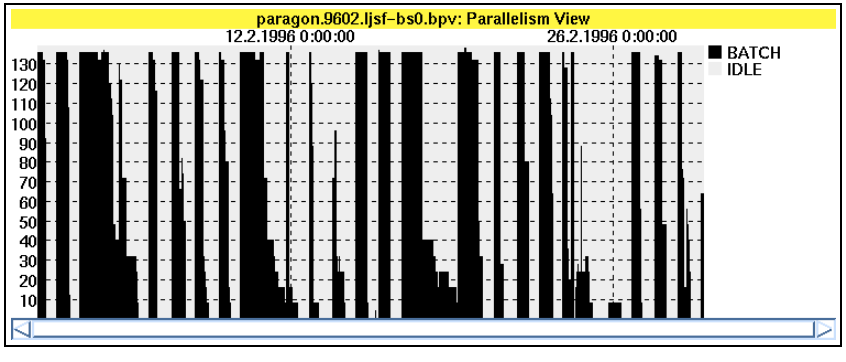


Abbildung 3.2: VAMPIR: Parallelism View

In der Ansicht *Parallelism View* (Abbildung 3.2) wird die Auslastung des massiv-parallelen Systems über die Zeit dargestellt. Auf der Ordinate ist die Prozessorzahl aufgetragen, auf der Abszisse die Zeit. Die Auslastung wird durch den Wert *BATCH* angegeben. In Abbildung 3.2 sind die Wochenenden (breite dunkle Bereiche) und die Nächte (schmale dunkle Bereiche) zu erkennen.

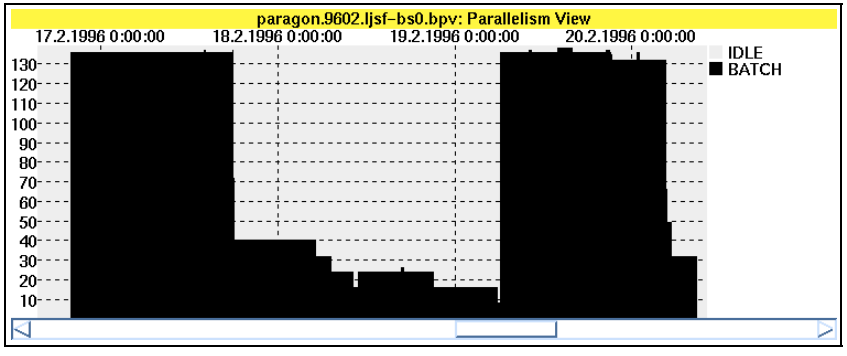


Abbildung 3.3: VAMPIR: Parallelism View, kurzer Zeitraum

In der Ansicht *Parallelism View* kann die Zeitachse gedehnt werden. Dadurch ist es möglich, die Auslastung auch in einem kürzeren Zeitraum zu analysieren und Zeiten niedriger Auslastung aufzuspüren. Abbildung 3.3 zeigt die Auslastung des Systems Intel Paragon in der Zeit vom Abend des 16.2.1996 bis zum Morgen des 20.2.1996 durch die Strategie *Largest Job-Size First* mit *Best-Size*-Back-Filling. In diesem Zeitraum lief das massiv-parallele System $3\frac{1}{2}$ Tage ununterbrochen im Batch-Betrieb (Wochenende 17.2./18.2. und Rosenmontag 19.2.1996). In der Zeit von Samstagabend (17.2.1996) bis Montagmorgen (19.2.1996) belegen die durch den Scheduler ausgewählten Jobs maximal 40 Knoten des Systems, obwohl einige wartende Jobs das System in dieser Zeit wesentlich höher auslasten könnten. Diese niedrige Auslastung wird durch das Back-Filling verursacht. Abbildung 3.7 zeigt die Entscheidung des Schedulers, die zu der niedrigen Auslastung des Systems führt, und die Warteschlange der Jobs in diesem Moment. Zum Zeitpunkt der Entscheidung wird lediglich ein Job mit einer Größe von 8 Knoten ausgeführt (130 Knoten sind frei). Der Scheduler startet einen Job mit 64 Knoten, obwohl Jobs mit 136 Knoten in der Warteschlange stehen. Nachdem der Job mit 64 Knoten gestartet wurde, bringt der Scheduler mittels Back-Filling weitere Jobs mit niedriger Knotenzahl zur Ausführung. Einige dieser Jobs übermitteln ihrerseits neue Jobs (siehe Abbschnitt 3.1.3), die wiederum durch das Back-Filling-Verfahren gestartet werden. Dadurch werden die Jobs mit 136 Knoten für längere Zeit an der Ausführung gehindert.

In Abbildung 3.4 ist die Ansicht *Global Timeline* dargestellt. Auf der Ordinate sind die Prozessoren aufgetragen (Prozessor Nr. 1 oben, Prozessor Nr. 138 unten) und auf der Abszisse ist die Zeit aufgetragen. Die Abbildung zeigt einen für die Strategie *Largest Job-Size First* typischen Wochentag. Ab 20:00 werden zunächst die Jobs mit hoher Knotenzahl gestartet. Später kommen die Jobs mit den niedrigeren Knotenzahlen zur Ausführung.

Die Linien in Abbildung 3.4 repräsentieren die Wartezeiten der Jobs zwischen Übermittlung an den Scheduler und dem Beginn der Ausführung auf dem massiv-parallelen System. Der Anfangspunkt (links unten) jeder Linie gibt den Zeitpunkt

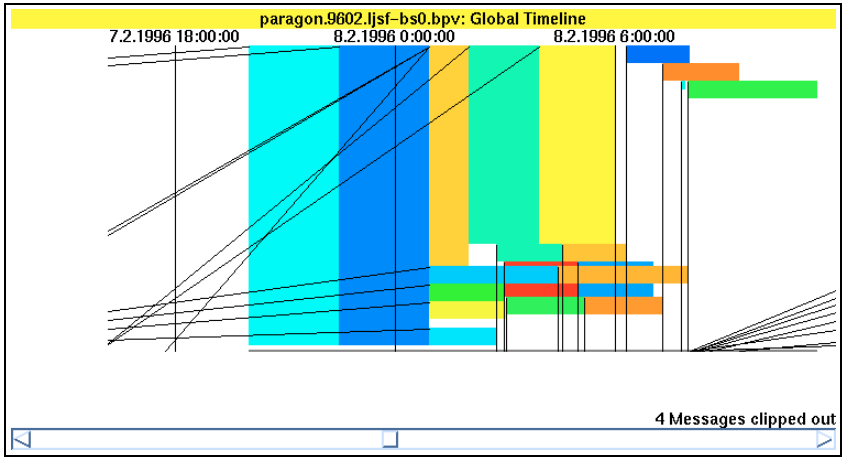


Abbildung 3.4: VAMPIR: Global Timeline

der Übermittlung des Jobs an den Scheduler an, der Endpunkt (rechts oben) gibt den Startzeitpunkt des Jobs an. Die Linien beginnen jeweils auf dem Prozessor Nr. 139, der auch die aktive Zeit *ACTIVE* anzeigt. Das Ende einer Linie ist der erste dem Job zugewiesene Knoten.

Einige Linien in Abbildung 3.4 verlaufen senkrecht von unten nach oben. Diese Linien entsprechen einer Wartezeit von 0 Sekunden. Das bedeutet, daß diese Jobs unmittelbar nach ihrer Übermittlung gestartet werden. Dies geschieht häufig mit Jobs, die von anderen Jobs übermittelt wurden (siehe Abbschnitt 3.1.3).

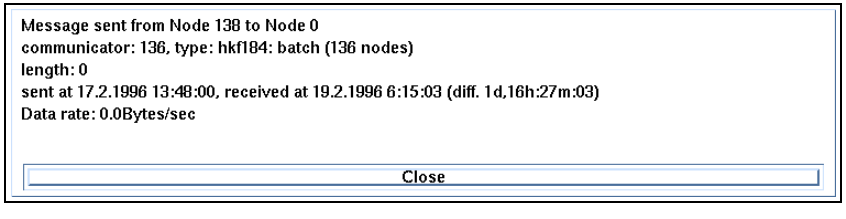


Abbildung 3.5: VAMPIR: Informationen über die Wartezeit eines Jobs

Die Linien in der Ansicht *Global Timeline* können ausgewählt werden, um Informationen über die Wartezeit eines Jobs zu erhalten. Abbildung 3.5 zeigt die Informationen, die durch die Auswahl einer Linie angezeigt werden. Dies sind der Übermittlungszeitpunkt des Jobs an den Scheduler (`sent at`), der Startzeitpunkt des Jobs (`received at`) sowie die Größe des Jobs (`communicator`) und die *Userid* des Benutzers (`type`).

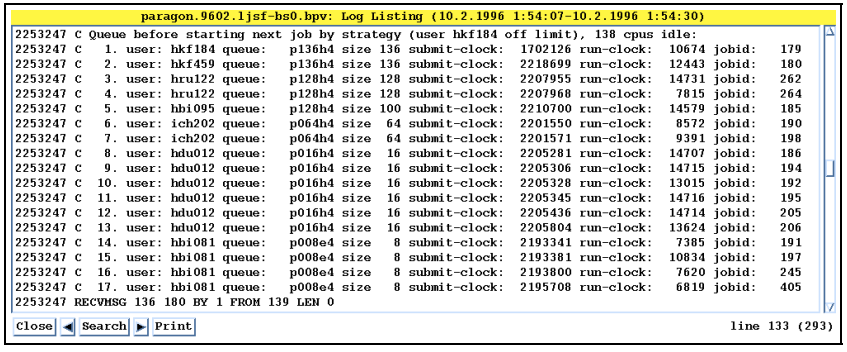


Abbildung 3.6: VAMPIR: Log Listing des Starts eines Jobs durch die Strategie

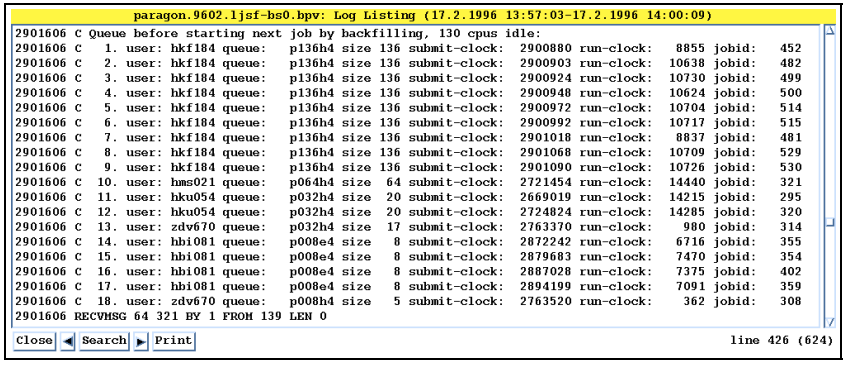


Abbildung 3.7: VAMPIR: Log Listing des Starts eines Jobs durch Back-Filling

Die Abbildungen 3.6 und 3.7 zeigen Ausschnitte des durch den Simulator erzeugten Tracefiles. Diese Anzeige wird durch die Funktion *Show Log* in den Ansichten *Show Parallelism* und *Global Timeline* erzeugt. Mit Hilfe dieser Anzeige kann jede Entscheidung des Job-Schedulers überprüft werden. In der obersten Zeile der beiden Listings steht jeweils, wie die Entscheidung getroffen wurde.

In Abbildung 3.6 wurde die Entscheidung einen Job zu starten durch die Strategie *Largest Job-Size First* getroffen. Allerdings wird nicht der erste Job in der Warteschlange gestartet, weil das *NQS-Limit* (siehe Abschnitt 3.1.4) des Benutzers überschritten wurde. In der letzten Zeile dieses Listings gibt die erste Zahl (`136`) hinter dem Wort `RECVMSG` die Größe des gestarteten Jobs an, die zweite Zahl (`180`) gibt die `jobid` des Jobs an. In dem vorliegenden Fall wurde also der zweite Job der Warteschlange gestartet.

Das in Abbildung 3.7 aufgeführte Listing zeigt, daß hier der Job mit der `jobid 321` (10. in der Warteschlange) durch Back-Filling gestartet wurde, weil für die neun

Jobs, die vor diesem Job in der Warteschlange plaziert sind, nicht genügend freie Knoten auf dem System vorhanden sind.

Die hier vorgestellten Verfahren zur Überprüfung der Entscheidungen des Job-Schedulers helfen auch bei der Suche nach Optimierungsmöglichkeiten der verschiedenen Job-Scheduling-Strategien.

4. Die Ergebnisse der Untersuchung

In der Untersuchung wurde die Belegung der massiv-parallelen Systeme Intel Paragon und Cray T3E in Abschnitten mit jeweils einem Monat Laufzeit simuliert. Die Zusammenfassung der Ergebnisse aller Monate (Intel: 21 Monate, Cray: 3 Monate) ist die Grundlage der nun folgenden Analyse.

Zunächst wird ein Überblick über die Bewertungskriterien gegeben. Dann werden für beide massiv-parallele Systeme die nach dem Sortierkriterium der Job-Scheduling-Strategie zusammengefaßten Ergebnisse präsentiert. Abschließend werden die Strategien anhand der Ergebnisse für Intel Paragon und Cray T3E beurteilt.

4.1 Bewertungskriterien

Zur Bewertung der Job-Scheduling-Strategien wurden Daten über die Belegung der massiv-parallelen Systeme und statistische Werte der Job-Wartezeiten ausgewertet.

Viele der Daten, die zur Beschreibung der Auslastung genutzt werden, sind Zeiten. Um diese Zeiten in Auslastungswerte umzurechnen, wird die Last benötigt, die in diesen Zeiten verarbeitet wird. Für alle Job-Scheduling-Strategien ist die Gesamtlast L_{ges}, die Summe der Produkte aus Laufzeit und Knotenzahl aller Jobs, gleich.

$$L_{ges} = \sum_{i=1}^{n} t_i \times p_i \quad \text{mit}$$

L_{ges}	Gesamtlast
t_i	Laufzeit des $i-$ten Jobs
p_i	Anzahl geforderter Knoten des $i-$ten Jobs
n	Anzahl der Jobs

Die Gesamtlast läßt sich in die Last im Batch-Betrieb, im gemischten Betrieb und im interaktiven Betrieb aufteilen:

$$L_{ges} = L_{bat} + L_{mix} + L_{ia} \quad \text{mit}$$

L_{ges}	Gesamtlast
L_{bat}	Last im Batch-Betrieb
L_{mix}	Last im gemischten Betrieb
L_{ia}	Last im interaktiven Betrieb

Folgende Werte beschreiben die Belegung der massiv-parallelen Systeme:

- Gesamtlaufzeit T_{ges}:

  Dieser Wert gibt die Zeitdifferenz zwischen dem Start des ersten Jobs und dem Ende des letzten Jobs an. Mit Hilfe dieses Wertes wird auch die Gesamtauslastung berechnet:

$$l_{ges} = \frac{L_{ges}}{T_{ges} \times P_{MPP}} \quad \text{mit}$$

l_{ges}	Gesamtauslastung des Systems
L_{ges}	Gesamtlast
T_{ges}	Gesamtlaufzeit
P_{MPP}	Anzahl der Knoten im System

- aktive Zeit $T_{act,ges}$:

  Dies ist die Zeit, während der mindestens ein Knoten des Systems aktiv ist. Die Auslastung in der aktiven Zeit ergibt sich zu:

$$l_{act,ges} = \frac{L_{ges}}{T_{act,ges} \times P_{MPP}} \quad \text{mit}$$

$l_{act,ges}$	Auslastung in der aktiven Zeit
L_{ges}	Gesamtlast
$T_{act,ges}$	aktive Zeit
P_{MPP}	Anzahl der Knoten des Systems

- aktive Batch-Betriebszeit $T_{act,bat}$:

  Die Zeit, während der das massiv-parallele System für den Batch-Betrieb reserviert war und durch Jobs genutzt wurde. Die Auslastung der aktiven Batch-Betriebszeit ist:

$$l_{act,bat} = \frac{L_{bat}}{T_{act,bat} \times P_{MPP}} \quad \text{mit}$$

$l_{act,bat}$	Auslastung in der aktiven Batch-Betriebszeit
L_{bat}	Last im Batch-Betrieb
$T_{act,bat}$	aktive Batch-Betriebszeit
P_{MPP}	Anzahl der Knoten des Systems

- freie Batch-Betriebszeit $T_{idle,bat}$:

  Für den Batch-Betrieb reservierte Zeit, während der kein Job ausgeführt wird. Der Anteil der freien Batch-Betriebszeit an der gesamten Batch-Betriebszeit ist wie folgt definiert:

$$t_{idle,bat} = \frac{T_{idle,bat}}{T_{bat}} \quad \text{mit}$$

$T_{bat} =$	$T_{act,bat} + T_{idle,bat}$ gesamte Batch-Betriebszeit
$t_{idle,bat}$	freier Anteil an der gesamten Batch-Betriebszeit
$T_{idle,bat}$	freie Batch-Betriebszeit
$T_{act,bat}$	aktive Batch-Betriebszeit

- aktive Zeit im gemischten Betrieb $T_{act,mix}$:

  Den gemischten Betrieb gibt es nur auf dem massiv-parallelen System Cray T3E. Während dieses Betriebs dürfen sowohl Jobs im Batch-Betrieb als auch interaktive Jobs ausgeführt werden (vormittags von 7:00 Uhr bis 12:00 Uhr)

- freie Zeit im gemischten Betrieb $T_{idle,mix}$:
  analog zur freien Batch-Betriebszeit
- Last im gemischten Betrieb L_{mix}:
  Last der im gemischten Betrieb ausgeführten Jobs (Einheit: Prozessorstunden)
- aktive Zeit im interaktiven Betrieb $T_{act,ia}$
- Last im interaktiven Betrieb L_{ia}:
  Last der im interaktiven Betrieb ausgeführten Jobs (Einheit: Prozessorstunden)

Das Kriterium Gesamtauslastung l_{ges} hat nur eine geringe Aussagekraft, da die Gesamtlaufzeit T_{ges} stark von den Übermittlungszeiten der Jobs beeinflußt wird[1] und somit eine höhere Gesamtauslastung als in der Originalbelegung unmöglich ist. Dieses Kriterium kann jedoch genutzt werden, um Strategien zu erkennen, die zu einer wesentlich längeren Laufzeit führen als dies durch die Übermittlungszeiten der Jobs erforderlich wäre. Die anderen Auslastungswerte sind nur mittelbar von den Übermittlungszeiten der Jobs abhängig. Da keine der betrachteten Job-Scheduling-Strategien das massiv-parallele System frei läßt, wenn Jobs warten und ausgeführt werden dürfen, kann es, falls die Warteschlange leer ist und ein Job eine geringe Knotenanzahl fordert, kommt es zu einer niedrigen Auslastung. Die Werte für die Auslastung in der aktiven Zeit l_{act} und während der Batch-Betriebszeit $l_{act,bat}$ sollen möglichst hoch sein. Als Vergleich dienen bei allen Betrachtungen die Werte der Originalbelegung. Durch manuelles Eingreifen der Systemadministratoren in die Zeitplanerstellung der auf den massiv-parallelen Systemen Intel Paragon und Cray T3E eingesetzten Scheduling-Systeme konnte die Auslastung der Systeme auf einen sehr hohen Wert gebracht werden, der nur durch wenige automatisierte Job-Scheduling-Strategien erreicht werden kann. Dabei nutzen die Systemadministratoren zum Beispiel Informationen über die Job-Laufzeiten die sie durch Rückfragen bei den Benutzern oder durch Beobachtung der bisherigen Laufzeiten der Programme erhalten. Diese Informationen stehen den untersuchten Job-Scheduling-Strategien natürlich nicht zur Verfügung.

Das Kriterium freie Batch-Betriebszeit $T_{idle,bat}$ dient dazu freie Bereiche aufzufinden, die durch eine gute Zeitplanerstellung entstehen, wenn alle Jobs der Warteschlange abgearbeitet wurden und keine weiteren Jobs übermittelt werden. Diese Situation entsteht beispielsweise, wenn alle Jobs die die Benutzer der massiv-parallelen Systeme vor einem Wochende übermitteln, berits Samstags abends ausgeführt wurden.

Die Auslastung hat für den interaktiven Betrieb und den gemischten Betrieb keinen hohen Informationswert. In diesen Betriebsarten sollten grundsätzlich so wenig Batchjobs wie möglich ausgeführt werden, damit der interaktive Betrieb nicht beeinträchtigt wird. Aus diesem Grund werden nicht die Auslastungen, sondern die Lasten (L_{ia} , L_{mix}) und die aktiven Zeiten im interaktiven Betrieb ($T_{act,ia}$) und im gemischten Betrieb ($T_{act,mix}$) betrachtet.

[1] Der letzte Job kann frühestens unmittelbar nach seiner Übermittlung ausgeführt werden.

In der Untersuchung werden alle Auslastungen beziehungsweise Lasten für jede einzelne Strategie betrachtet. In der Zusammenfassung der Ergebnisse wird vor allem die Auslastung in der aktiven Zeit präsentiert, da sich anhand dieses Wertes die unterschiedlichen Auslastungen der massiv-parallelen Systeme durch die einzelnen Strategien am deutlichsten zeigen. Wenn dieser Wert zu falschen beziehungsweise zu nicht eindeutigen Schlüssen führt, werden weitere Auslastungswerte betrachtet.

Viele der Jobs, die an die massiv-parallelen Systeme Intel Paragon und Cray T3E übermittelt werden, können nicht sofort ausgeführt werden, weil sie eine der in den Abschnitten 3.1.5 und 3.2.5 beschriebenen Beschränkungen überschreiten oder nicht genügend freie Knoten zur Verfügung stehen. In diesen Fällen werden die Jobs an Warteschlangen zur späteren Ausführung übermittelt. Die Wartezeit ist für den Anwender eine wichtige Größe. Zur Bewertung der Wartezeiten werden die Jobs in fünf Klassen eingeteilt:

- alle Jobs C_{all}
- Jobs, deren Laufzeit maximal 10 Minuten beträgt $C_{\leq 10min}$
- Jobs, deren Laufzeit länger als 10 Minuten ist $C_{>10min}$
- Jobs, die maximal 32 Knoten benötigen $C_{\leq 32p}$
- Jobs, die mehr als 32 Knoten benötigen $C_{>32p}$

Von den fünf Klassen sind die Klassen der Jobs mit kleiner Job-Größe $C_{\leq 32p}$ sowie der Jobs mit kurzer Laufzeit $C_{\leq 10min}$ von besonderem Interesse, da die Jobs dieser beiden Klassen unmittelbar oder von einem Tag auf den nächsten ausgeführt werden sollten.

Für jede dieser fünf Klassen werden die folgenden statistischen Werte bestimmt:

- arithmetischer Mittelwert
- Maximum
- .50-Quantil Q_{50}, .75-Quantil Q_{75}, .90-Quantil Q_{90} und .95-Quantil Q_{95}

Das .50-Quantil wird auch als Median bezeichnet. Dieser Wert gibt an, wie lange 50% der Jobs warten, bis sie ausgeführt werden. Im Unterschied zum arithmetischen Mittelwert bleiben hierbei Extremwerte unberücksichtigt. Bei der Auswertung der vorliegenden Daten hat sich gezeigt, daß für viele Strategien das .50-Quantil und oft auch das .75-Quantil in den Gruppen C_{all}, $C_{\leq 32p}$ und $C_{\leq 10min}$ den Wert 0 hat. Dieser Effekt tritt auf, da viele der Jobs, die bei der Fertigstellung ihres Vorgängers übermittelt wurden (siehe Kapitel 3, Abschnitte 3.1.3 und 3.2.3), direkt starten, da ihr Vorgänger die benötigten Knoten freigibt und die Jobs über Back-Filling ausgewählt werden. Dies gilt für die Klassen C_{all} und $C_{\leq 32p}$. In der Klasse $C_{\leq 10min}$ kommen die

Nullwerte für diese Quantile aufgrund der Sonderregelung für Test-Jobs zustande. Diese Sonderegelung erlaubt es, Jobs mit kurzer Laufzeit (maximal eine Stunde) und geringer Knotenanzahl (maximal 4 Knoten) im interaktiven Betrieb zu Testzwecken auszuführen. Diese Sonderregelung betrifft zwar auch die beiden anderen Klassen, aber auf Grund der geringen Zahl an Jobs, die im interaktiven Betrieb zu Testzwecken gestartet werden, fällt diese Regelung dort nicht ins Gewicht. In der Klasse $C_{\leq 10min}$ wird die Mehrzahl der Jobs durch diese Sonderregelung gestartet. Die zweite große Gruppe in dieser Klasse bilden Jobs, die aufgrund eines Fehlers im Programmablauf frühzeitig abbrechen.

Neben dem arithmetischen Mittelwert und dem Maximum der Wartezeiten werden in dieser Untersuchung die oben aufgeführten Quantile der Wartezeiten betrachtet, da diese Werte die Verteilung der Wartezeiten besser beschreiben als der Mittelwert und die Standardabweichung. Die Quantile werden durch den Simulator jeweils für einen Monat bestimmt. Für das System Intel Paragon werden 21 Monate betrachtet, so daß die Analyse der Quantile sehr umfangreich ist. Die Quantile der Wartezeiten werden zwar in der Untersuchung der einzelnen Strategien ausgewertet, aber in der nun folgenden Zusammenfassung der Ergebnisse werden hauptsächlich die Mittelwerte und Maxima der Wartezeiten präsentiert. Wenn diese Werte zu keiner eindeutigen Beurteilung der Strategien führen oder die Quantile der Wartezeiten eine andere Beurteilung zulassen, werden die Quantile exemplarisch für einzelne Monate betrachtet, um eine genaue Beurteilung der Strategien zu erreichen.

Zur Beurteilung der Strategien müssen die Bewertungskriterien gewichtet werden. Die folgende Liste gibt die Einstufung der Kriterien nach ihrer Wichtigkeit an:

1. Gesamtauslastung
2. Auslastung in der aktiven Zeit und in der Batch-Betriebszeit
3. Wartezeiten aller Jobs
4. Wartezeiten der Jobs in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$
5. Last im gemischten und im interaktiven Betrieb
5. freie Batch-Betriebszeit

4.2 Analyse der Belegung von Intel Paragon

Die Originalbelegung des Systems Intel Paragon, die in der nun folgenden Analyse als Maßstab dient, hat folgende charakteristische Werte:

- Gesamtlaufzeit: ca. 633 Tage
- Gesamtlast: ca. 303 Tage auf 138 Knoten
- Gesamtanzahl der Jobs, Jobs der Klasse C_{all}: 14353
- Anzahl der Jobs in der Klasse $C_{\leq 32p}$: 10693
- Anzahl der Jobs in der Klasse $C_{>32p}$: 3660
- Anzahl der Jobs in der Klasse $C_{\leq 10min}$: 5180
- Anzahl der Jobs in der Klasse $C_{>10min}$: 9173

Alle weiteren Werte der Originalbelegung werden bei den Vergleichen der Ergebnisse der Strategien mit den Werten der Originalbelegung in der nun folgenden Zusammenfassung der Ergebnisse vorgestellt.

Die Belegung des massiv-parallelen Systems Intel Paragon wird mit einer Genauigkeit von $\pm$ 1 Sekunde simuliert.

4.2.1 First Come First Serve

Die Strategie *First come First Serve* (*FCFS*) nutzt die Ankunftzeit der Jobs am System als Sortierkriterium. Die hier betrachtete Auswahl der Back-Filling-Verfahren in Verbindung mit der Job-Scheduling-Strategie *First Come First Serve* sind:

- First Come First Serve ohne Back-Filling (*fcfs*)
- First Come First Serve mit First-Come-Back-Filling und Fairneßbetrachtung anhand der Queue-Laufzeiten (*fcfs fc2*)
- First Come First Serve mit Best-Size-Back-Filling ohne Fairneßbetrachtung (*fcfs bs0*)
- First Come First Serve mit Best-Demand-Back-Filling ohne Fairneßbetrachtung (*fcfs bd0*)
- First Come First Serve mit Best-Demand-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*fcfs bd1*)
- First Come First Serve mit Best-Combination-Back-Filling ohne Fairneßbetrachtung (*fcfs bc*)

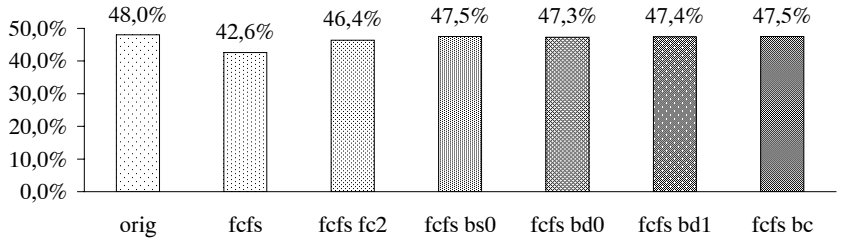


Abbildung 4.1: Intel Paragon FCFS: Gesamtauslastung l_{ges}

Als Vergleichswert dient, wie auch bei der Betrachtung aller weiteren Strategien, der Wert der Originalbelegung (*orig*).

Abbildung 4.1 zeigt die Gesamtauslastung l_{ges} des Systems. Der Wert *orig* stellt den Anteil der im Batch-Betrieb gestarteten Jobs an der maximalen Geamtlast, also der 100%-igen Auslastung der Maschine dar. Die verbleibenden 52% wurden entweder für interaktiven Betrieb genutzt, waren Ausfallzeiten oder ungenutzte Resourcen.

Die Differenz von 1,1% in der Gesamtauslastung zwischen der besten (*fcfs bc*) und der schlechtesten Variante (*fcfs fc2*) mit Back-Filling entspricht einer um 16 Tage längeren Gesamtlaufzeit. Die Gesamtlaufzeit der besten Variant (*fcfs bc*) ist wiederum ca. 6 Tage länger (0,5% niedrigere Gesamtauslastung) als die der Originalbelegung, bei einer Gesamtlaufzeit der Originalbelegung von 633 Tagen.

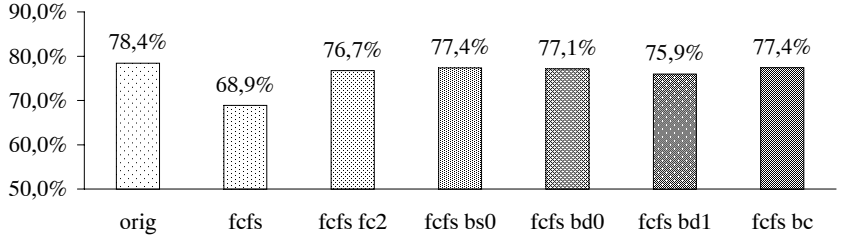


Abbildung 4.2: Intel Paragon FCFS: Auslastung in der aktiven Zeit l_{act}

Die Auslastung in der aktiven Zeit l_{act} (Abbildung 4.2) zeigt ähnliche Verhältnisse zwischen den einzelnen Varianten. Ohne Back-Filling ergibt sich, wie auch für die Gesamtauslastung l_{ges} (Abbildung 4.1), eine sehr niedrige Auslastung. Diese wird durch Jobs mit geringer Knotenanzahl hervorgerufen, die die Ausführung des nächsten Jobs blockieren, falls dieser mehr als die verbleibenden Knoten fordert. Der Unterschied zwischen der besten Variante mit Back-Filling (*fcfs bc*) und der schlechtesten (*fcfs bd1*) beträgt 1,5%, dies entspricht einer um ca. 8 Tage längeren aktiven Zeit.

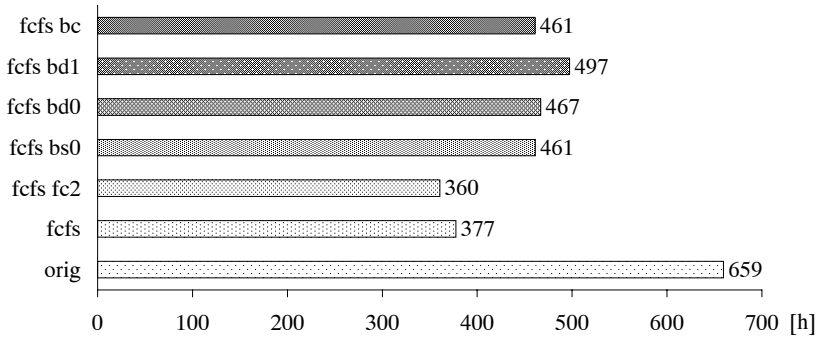


Abbildung 4.3: Intel Paragon FCFS: aktive Zeit im interaktiven Betrieb $t_{act,ia}$

Ein Vergleich der Varianten *fcfs fc2* und *fcfs bd1* zeigt, daß die Strategie *fcfs fc2* trotz niedriger Gesamtauslastung eine höhere Auslastung in der aktiven Zeit erreicht. Dies wird bedingt durch die kurze Zeit, die diese Variante im interaktiven Betrieb beansprucht. Eine Strategie, die wie *fcfs bd1* mehr Zeit im interaktiven Betrieb verwendet, hat eine niedrigere Auslastung in der aktiven Zeit. Das folgende Beispiel verdeutlicht diesen Zusammenhang: Ein Job J_1 mit geringer Knotenanzahl und langer Laufzeit wird kurz vor Beginn des interaktiven Betriebs gestartet. Nach kurzer Zeit werden alle anderen laufenden Jobs beendet. Nun zählt die Zeit, die Job J_1 bis zur Fertigstellung benötigt als aktive Zeit im interaktiven Betrieb $T_{act,ia}$, und somit geht sie auch in die aktive Zeit T_{act} mit ein. Da aufgrund der geringen Knotenanzahl von Job J_1 nur eine niedrige Auslastung erreicht wird und im interaktiven Betrieb keine weiteren Jobs gestartet werden dürfen, wird die Auslastung in der aktiven Zeit l_{act} ebenfalls niedriger. Abbildung 4.3 zeigt die Zeiten, die die einzelnen Varianten im interaktiven Betrieb belegen.

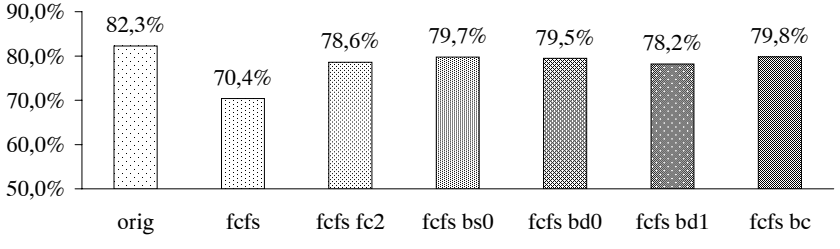


Abbildung 4.4: Intel Paragon FCFS: Auslastung in der aktiven Batch-Betriebszeit $l_{act,bat}$

Der Einfluß der Einschränkungen durch den interaktiven Betrieb auf die Auslastung wird durch Betrachtung der Batch-Betriebszeit T_{bat} ausgeschaltet. Abbildung 4.4

zeigt die Auslastung in der aktiven Batch-Betriebszeit. Dieses Bewertungskriterium kann zusammen mit dem freien Anteil der Batch-Betriebszeit $t_{idle,bat}$ (Abbildung 4.5) zur Beurteilung der Qualität der Job-Scheduling-Strategien benutzt werden, da die Strategien in der Batch-Betriebszeit nur wenige Einschränkungen erfahren[2].

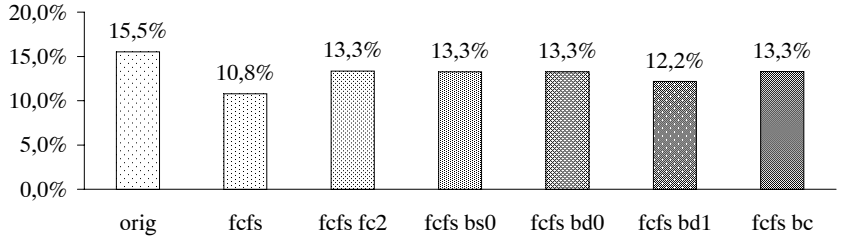


Abbildung 4.5: Intel Paragon FCFS: Anteil der freien Zeit an der gesamten Batch-Betriebszeit $t_{idle,bat}$

Mit den zuletzt vorgestellten Informationen über die Batch-Betriebszeit ($l_{act,bat}$, $t_{idle,bat}$) lassen sich Rückschlüsse über die Gesamtauslastung ziehen. Eine Betrachtung der beiden Strategien *fcfs fc2* und *fcfs bd1* zeigt, daß *fcfs fc2* eine niedrigere Auslastung im Batch-Betrieb hat, jedoch der Anteil der freien Zeit an der gesamten Batch-Betriebszeit genauso hoch ist wie bei den anderen Job-Scheduling-Strategien. Berücksichtigt man den Wert dieser Strategie für die Last im interaktiven Betrieb $L_{act,ia}$ (Abbildung 4.6), folgt, daß die Gesamtlaufzeit T_{ges} länger sein muß als bei den anderen Strategien, da die Gesamtlast L_{ges} für alle Strategien konstant ist. Für die Strategie *fcfs bd1* zeigt sich, daß die geringe Auslastung in der aktiven Batch-Betriebszeit teilweise durch eine längere Nutzung der Batch-Betriebszeit – kleinerer Anteil der freien Zeit an der Batch-Betriebszeit – aufgehoben wird. Wird nun noch die Last im interaktiven Betrieb L_{ia} betrachtet, so ergibt sich, daß die Gesamtlaufzeit nicht länger als bei den anderen Strategien ist.

Neben der Auslastung der massiv-parallelen Systeme durch die Belegung der Job-Scheduling-Strategien wird in dieser Arbeit auch die Beeinträchtigung des interaktiven Betriebs durch die Job-Scheduling-Strategien betrachtet, die so gering wie möglich sein sollte.

Um die tatsächliche Beeinträchtigung des interaktiven Betriebs zu beurteilen reicht eine Betrachtung der Zeit, die durch die Ausführung von Jobs im interaktiven Betrieb beansprucht wird ($T_{act,ia}$), nicht aus. Abhilfe schafft hier die Meßgröße Batch-Last im interaktiven Betrieb L_{ia}, die in Abbildung 4.6 dargestellt ist. Die Werte sind in Prozessorstunden aufgetragen. Die Originalbelegung (*orig*) beispielsweise nutzt 25858 Prozessorstunden im interaktiven Betrieb, das entspricht bei 138 Knoten einer

[2]Diese Einschränkungen sind das NQS-Limit, das verhindert, das ein Benutzer das massiv-parallel System zu lange allein benutzt, die maximale Job-Größe, die den interaktiven Betrieb vor übermäßiger Last aus dem Batch-Betrieb schützt und die Ausfallzeiten des Systems.

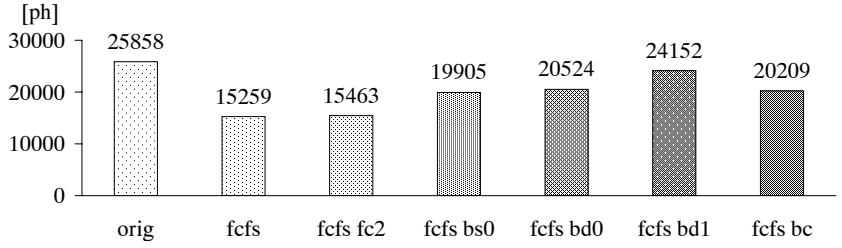


Abbildung 4.6: Intel Paragon FCFS: Last im interaktiven Betrieb L_{ia}

100%-igen Belegung für ca. 187 Stunden. Bei 633 Tagen Gesamtlaufzeit und interaktivem Betrieb an maximal 5 von 7 Tagen ergibt sich eine tägliche Beeinträchtigung von durchschnittlich ca. 25 Minuten. Dieser Wert macht deutlich, daß diese Betrachtung bei der Beurteilung der Job-Scheduling-Strategien eine untergeordnete Rolle spielt, zumal die meisten der betrachteten Strategien den Wert der Originalbelegung unterbieten, also der interaktive Betrieb nicht stärker beeinträchtigt wird, als dies bisher der Fall ist.

Die Analyse der Wartezeiten beginnt mit der mittleren und maximalen Wartezeit aller Jobs (Klasse C_{all}) in den Abbildungen 4.7 und 4.8.

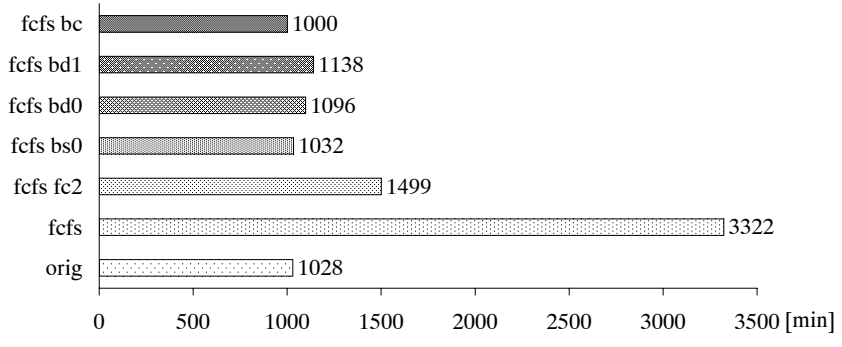


Abbildung 4.7: Intel Paragon FCFS: mittlere Wartezeit aller Jobs

Die Variante *fcfs* hat die längste mittlere Wartezeit aller Varianten der Job-Scheduling-Strategie *First Come First Serve*. Der Wert liegt mit 3322 Minuten etwa bei dem 3-fachen des Wertes der Originalbelegung. Die maximlae Wartezeit hingegen ist mit 346 Stunden 48 Stunden kürzer als in der Originalbelegung. Dieser scheinbare Gegensatz wird durch die Betrachtung der Quantile der Wartezeiten deutlich.

Es werden hier zwei Monate exemplarisch betrachtet. Diese Monate sind der Dezember 1995, der Monat mit der höchsten Last, und der Oktober 1996, ein Monat in

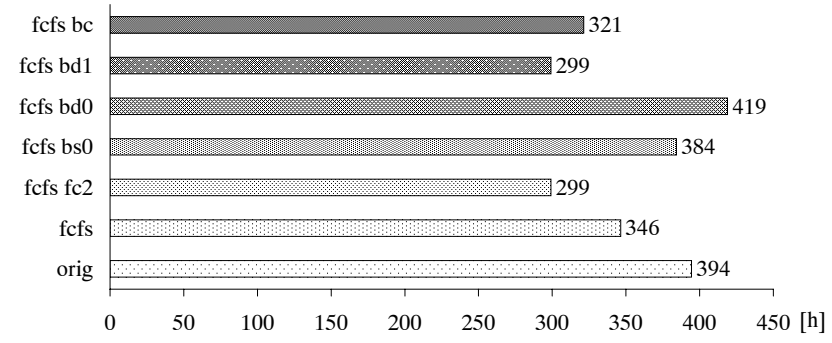


Abbildung 4.8: Intel Paragon FCFS: maximale Wartezeit aller Jobs

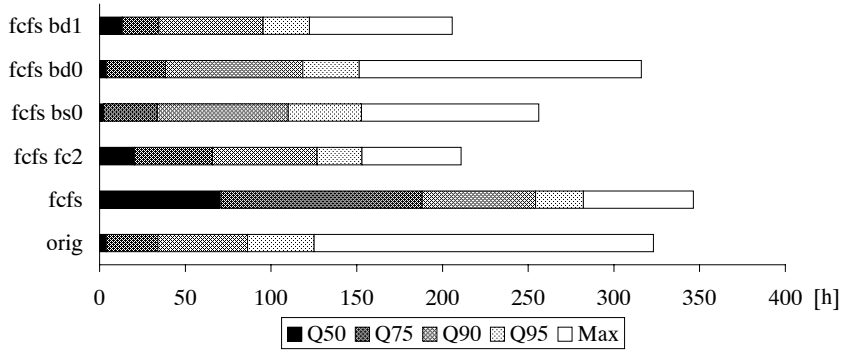


Abbildung 4.9: Intel Paragon FCFS: Quantile der Wartezeiten aller Jobs im Dezember 1995

dem die Auslastung des massiv-parallelen Systems durch die Originalbelegung dem Durchschnittswert der 21 Monate entsprechen. Die Quantile der Wartezeiten für die Monate Dezember 1995 (Abbildung 4.9) und Oktober 1996 (Abbildung 4.10) sind in Tabelle 4.1 zusammengestellt. Im Oktober 1996 werden durch die Originalbelegung 75% aller Jobs bereits nach 14 Stunden ausgeführt. Bei Verwendung der *fcfs* Variante werden 75% der Jobs nach spätestens 101 Stunden gestartet. Ein Drittel dieser Jobs wird jedoch frühestens nach 75 Stunden gestartet. Anhand dieser Werte wird der oben beschriebene scheinbare Gegensatz zwischen der sehr langen mittleren Wartezeit und der kurzen maximalen Wartezeit für die Variante *fcfs* verständlich.

Die mittleren Wartezeiten der Jobs in der Klassen $C_{\leq 32p}$ (Abbildung 4.11) und $C_{\leq 10min}$ (Abbildung 4.12) liefern ein ähnliches Bild wie die mittleren Wartezeiten der Jobs in der Klasse C_{all}. In der Klasse $C_{\leq 10min}$ ist jedoch zu erkennen, daß keine der betrachteten Varianten den Wert der Originalbelegung erreicht.

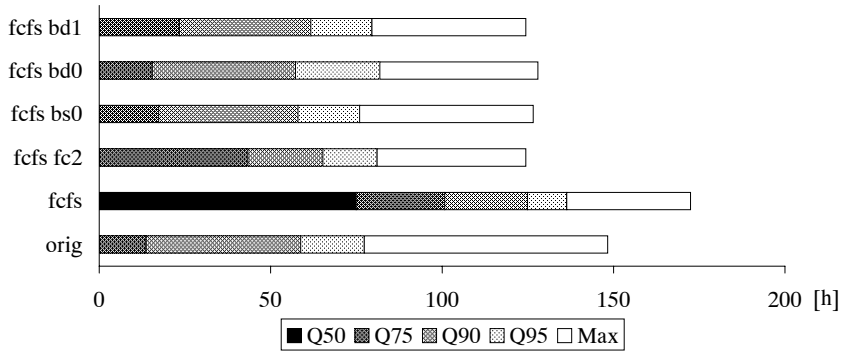


Abbildung 4.10: Intel Paragon FCFS: Quantile der Wartezeiten aller Jobs im Oktober 1996

Quantile der Wartezeiten in der Klasse C_{all} [h]										
	Dezember 1995					Oktober 1996				
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	4	34	86	125	323	0	14	59	77	148
fcfs	71	188	254	282	346	75	101	125	136	172
fcfs fc2	20	66	127	153	211	0	43	65	81	124
fcfs bs0	2	34	110	153	256	0	17	58	76	127
fcfs bd0	4	38	119	151	316	0	15	57	82	128
fcfs bd1	13	34	95	122	206	0	23	62	80	124
fcfs bc	2	35	114	158	223	0	14	57	74	127

Tabelle 4.1: Intel Paragon FCFS: Quantile der Wartezeiten aller Jobs Dezember 1995, Oktober 1996

Abschließend lassen sich die Varianten der Strategie *First Come First Serve* wie folgt zusammenfassen:

Die Varianten *fcfs bs0*, *fcfs bd0*, *fcfs bd1* und *fcfs bc* führen zu einer schlechteren Auslastung als die Originalbelegung, die Varianten *fcfs* und *fcfs fc2* lasten das massivparallele System deutlich schlechter aus. Die Betrachtung der Wartezeiten zeigt, daß nur die Varianten *fcfs bs0*, *fcfs bd0* und *fcfs bc* mit den Wartezeiten der Originalbelegung konkurrieren können. Unter diesen Varianten erzielt die Variante *fcfs bs0* die besten Ergebnisse.

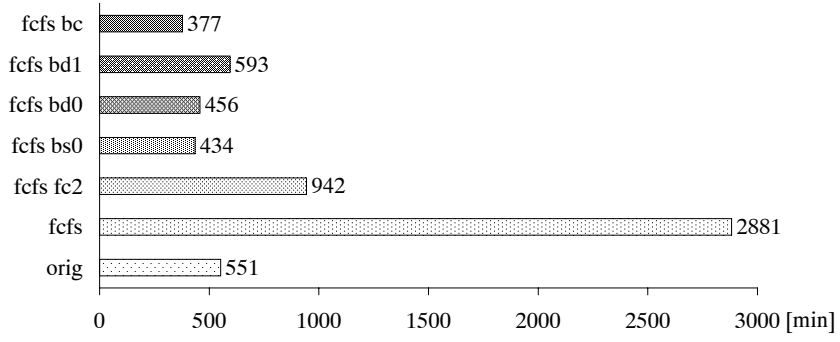


Abbildung 4.11: Intel Paragon FCFS: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

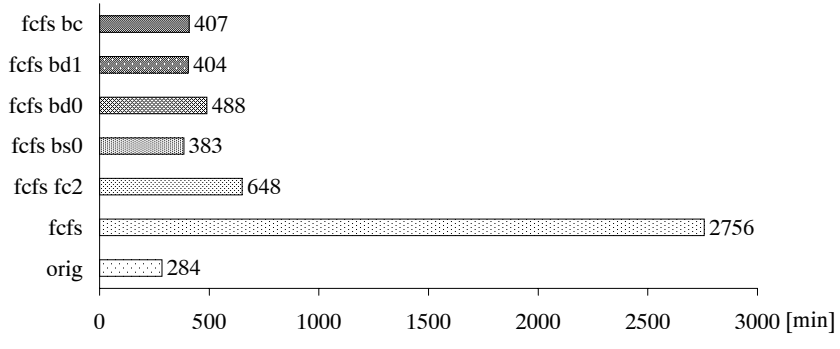


Abbildung 4.12: Intel Paragon FCFS: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

4.2.2 Shortest Processing Time First, Largest Processing Time First

Die Job-Scheduling-Strategien *Shortest Processing Time First* (*SPTF*) und *Largest Processing Time First* (*LPTF*) verwenden die tatsächliche Laufzeit der Jobs zur Sortierung der Warteschlange. Die exakte Laufzeit eines Jobs steht erst nach dessen Beendigung fest, so daß diese Strategien nur eingeschränkt realisierbar sind. Sie werden hier betrachtet, um zu ermitteln, ob sich der Aufwand für eine gute Abschätzung der Job-Laufzeit lohnt.

Die Varianten, die hier betrachtet werden, sind zum einen die Strategien ohne Back-Filling, zum anderen sind es die Varianten, die von allen Back-Filling-Verfahren

in Verbindung mit den beiden Strategien die besten Ergebnisse in der simulierten Belegung erzielten. Dies sind:

- Shortest Processing Time First ohne Back-Filling (*sptf*)
- Shortest Processing Time First mit First-Come-Back-Filling ohne Fairneßbetrachtung, beim Back-Filling werden Jobs der Sequential-Queues bevorzugt (*sptf fc3*)
- Shortest Processing Time First mit Best-Size-Back-Filling ohne Fairneßbetrachtung (*sptf bs0*)
- Largest Processing Time First ohne Back-Filling (*lptf*)
- Largest Processing Time mit First-Come-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lptf fc1*)
- Largest Processing Time mit Best-Demand-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lptf bd1*)

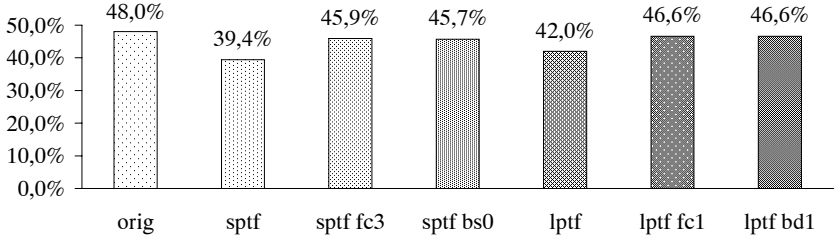


Abbildung 4.13: Intel Paragon SPTF, LPTF: Gesamtauslastung l_{ges}

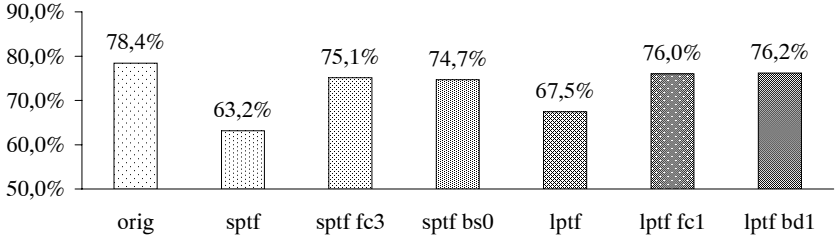


Abbildung 4.14: Intel Paragon SPTF, LPTF: Auslastung in der aktiven Zeit l_{act}

Die beste Gesamtauslastung l_{ges} (Abbildung 4.13 aller Varianten der Job-Scheduling-Strategie *Shortest Processing Time First* beträgt 45,9% (*sptf fc3*). Dies entspricht

einer um 28 Tage längeren Gesamtlaufzeit t_{ges} der simulierten Belegung gegenüber der Originalbelegung. Auch die Auslastung in der aktiven Zeit l_{act} (Abbildung 4.14) bringt ähnlich schlechte Ergebnisse: 75,1% für die Variante *sptf fc3* statt 78,4% durch die Originalbelegung. Die Job-Scheduling-Strategie *Largest Processing Time First* führt zu etwas besseren Ergebnissen, 46,6% Gesamtauslastung (*lptf bd1*) und 76,2% Auslastung in der aktiven Zeit. Diese Werte liegen jedoch ebenfalls deutlich unter denen der Originalbelegung und auch unter den Werten, die durch die besten Varianten der Job-Scheduling-Strategie *First Come First Serve* erzielt werden. Der Wert der besten Variante der Strategie *First Come First Serve* liegt zum Vergleich bei 77,4% (*fcfs bc*), ist also ebenfalls deutlich besser als die Varianten der Strategien *Shortest Processing Time First* und *Largest Processing Time First*.

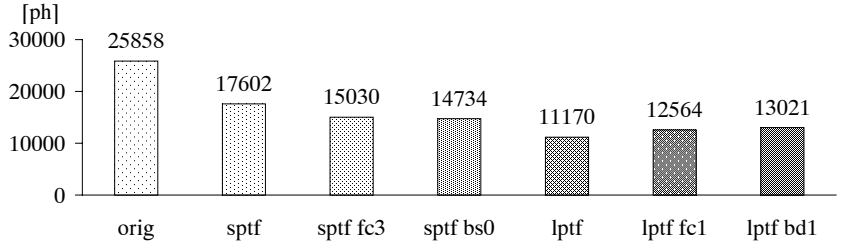


Abbildung 4.15: Intel Paragon SPTF, LPTF: Last im interaktiven Betrieb L_{ia}

Die Last im interaktiven Betrieb l_{ia} (Abbildung 4.15) wird durch alle Varianten gegenüber der Originalbelegung verringert. Die Varianten *sptf* und *sptf fc3* führen im Vergleich zu den Varianten der Job-Scheduling-Strategie *Largest Processing Time First* zu höheren Lasten im interaktiven Betrieb. Dies ist darauf zurückzuführen, daß der größte Teil dieser Last durch Jobs entsteht, die im Batch-Betrieb gestartet und im interaktiven Betrieb beendet werden. Da die Strategie *Shortest Processing Time First* zuerst Jobs mit kurzen Laufzeiten startet, werden am Ende der Batch-Betriebszeit Jobs mit längeren Laufzeiten gestartet. Diese Jobs werden für einen längeren Zeitraum im interaktiven Betrieb ausgeführt als die Jobs, die durch die Varianten der Strategie *Largest Processing Time First* kurz vor dem Ende der Batch-Betriebszeit gestartet werden. Der Vorteil der geringeren Last im interaktiven Betrieb kann jedoch den Nachteil der schlechten Auslastung aller Varianten der Job-Scheduling-Strategien *Shortest Processing Time First* und *Largest Processing Time First* nicht aufheben.

Die mittleren Wartezeiten der Jobs in der Klasse C_{all}, also der Gesamtheit aller Jobs, sind, wie in Abbildung 4.16 zu erkennen ist, nur für die Varianten der Strategie *Shortest Processing Time First* mit Back-Filling akzeptabel. Die maximalen Wartezeiten sind sowohl für die Varianten der Strategie *Shortest Processing Time First* als auch für die der Strategie *Largest Processing Time First* mehr als doppelt so lang wie die maximale Wartezeit der Originalbelegung.

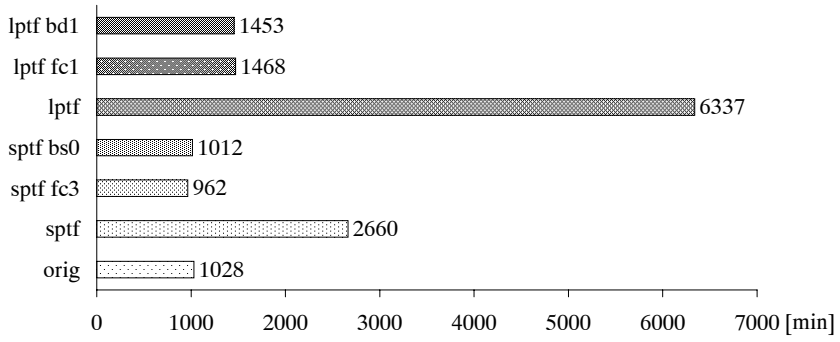


Abbildung 4.16: Intel Paragon SPTF, LPTF: mittlere Wartezeit aller Jobs

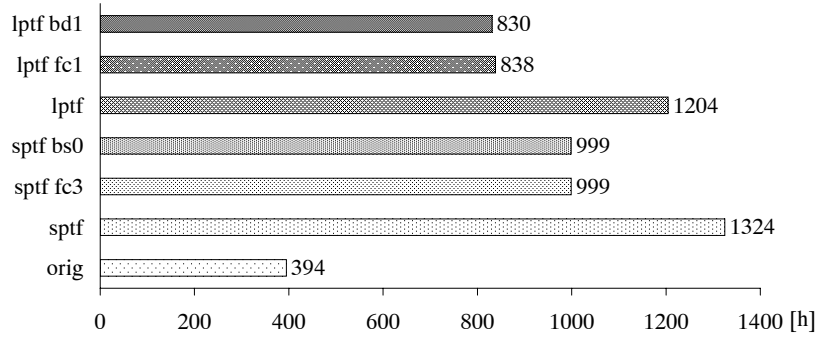


Abbildung 4.17: Intel Paragon SPTF, LPTF: maximale Wartezeit aller Jobs

Deutlich kürzere mittlere Wartezeiten als die Originalbelegung erreichen die Varianten *sptf fc3*, *sptf bs0* der Job-Scheduling-Strategie *Shortest Processing Time First* und die Varianten *lptf fc1* und *lptf bd1* der Strategie *Largest Processing Time First* (Abbildung 4.18, logarithmischer Einteilung der Zeitachse!) in der Klasse der Jobs mit kleiner Job-Größe $C_{\leq 32p}$. In dieser Job-Klasse sind die maximalen Wartezeiten der Varianten *sptf fc3* (182 Stunden), *sptf bs0* (178 Stunden) und *lptf fc1* (198 Stunden) deutlich kürzer als der Vergleichswert der maximalen Wartezeit (305 Stunden).

Abbildung 4.19 (logarithmische Einteilung der Zeitachse!) zeigt, daß die Varianten *sptf fc3* und *sptf bs0* für die Jobs mit kurzer Laufzeit – Klasse $C_{\leq 10min}$ – kürzere mittlere Wartezeiten liefern als die Originalbelegung. Die maximalen Wartezeiten dieser beiden Varianten sind für diese Job-Klasse jedoch höher als die maximale Wartezeit in der Originalbelegung, 219 Stunden für *sptf fc3* und 195 Stunden für *sptf bs0* gegenüber 178 Stunden in der Originalbelegung.

Die Strategien *Shortest Processing Time First* und *Largest Processing Time First*

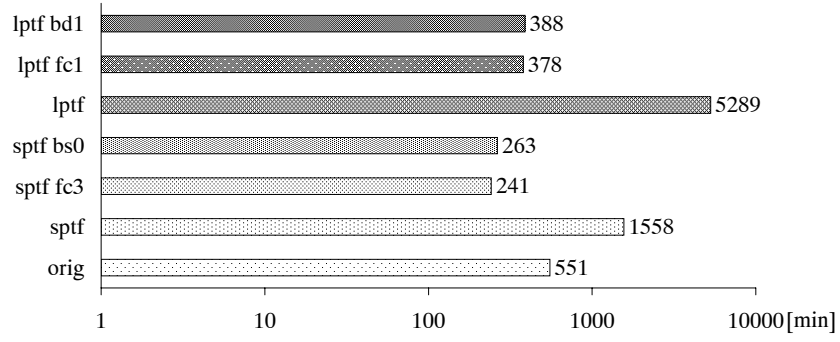


Abbildung 4.18: Intel Paragon SPTF, LPTF: mittlere Wartezeit der Jobs in Klasse $C_{\leq 32p}$

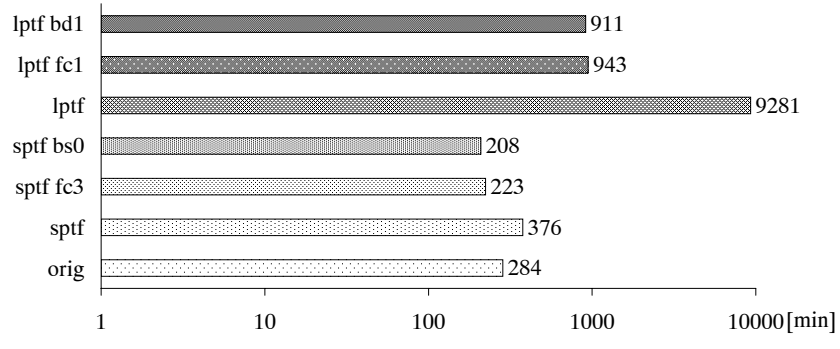


Abbildung 4.19: Intel Paragon SPTF, LPTF: mittlere Wartezeit der Jobs in Klasse $C_{\leq 10min}$

lassen sich wie folgt zusammenfassen:

Alle Varianten der *Shortest Processing Time First* und der Job-Scheduling-Strategie *Largest Processing Time First* sind den Varianten der Strategie *First Come First Serve* und der Originalbelegung in den Bewertungskriterien zur Auslastung unterlegen. Die Back-Filling Varianten der Strategie *Shortest Processing Time First* liefern für die Jobs mit kurzen Laufzeiten und mit geringer Job-Größe kurze mittlere Wartezeiten. Für die Gesamtzahl der Jobs ergibt sich bei diesen Varianten eine akzeptable mittlere Wartezeit, jedoch eine extrem hohe maximale Wartezeit. Alle anderen Varianten führen in der einen oder anderen Job-Klasse zu Wartezeiten, die für die Benutzer nicht zumutbar sind.

4.2.3 Smallest Job-Size First, Largest Job-Size First, Best Package

Die Strategien *Smallest Job-Size First*, *Largest Job-Size First* und *Best Package* nutzen die Job-Größe zur Sortierung der Warteschlange. Diese Strategien können ohne Einschränkung realisiert werden, es sei denn, eines der Back-Filling Verfahren nutzt die exakte Laufzeit der Jobs als Auswahlkriterium. Für die Job-Scheduling-Strategie *Smallest Job-Size First* gibt es kein Back-Filling-Verfahren, da der Job mit der geringsten Anzahl geforderter Knoten als erster in der Warteschlange steht. Falls dieser Job nicht gestartet werden kann, weil nicht genügend Knoten frei sind (und nur in diesem Fall tritt das Back-Filling in Aktion) kann auch kein anderer Job aus der Warteschlange gestartet werden, weil alle anderen Jobs gleich viele oder mehr Knoten fordern. In diesem Abschnitt werden zunächst die folgenden Varianten der einzelnen Job-Scheduling-Strategien betrachtet:

- Smallest Job-Size First ohne Back-Filling (*sjsf*)
- Largest Job-Size First mit Best-Size-Back-Filling ohne Fairneßbetrachtung (*ljsf bs0*)
- Largest Job-Size First mit Best-Size-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*ljsf bs1*)
- Largest Job-Size First mit Best-Size-Back-Filling und Fairneßbetrachtung anhand der Queue-Laufzeiten(*ljsf bs2*)
- Largest Job-Size First mit Best-Combination-Back-Filling ohne Fairneßbetrachtung (*ljsf bc*)
- Best Package (*bp*)

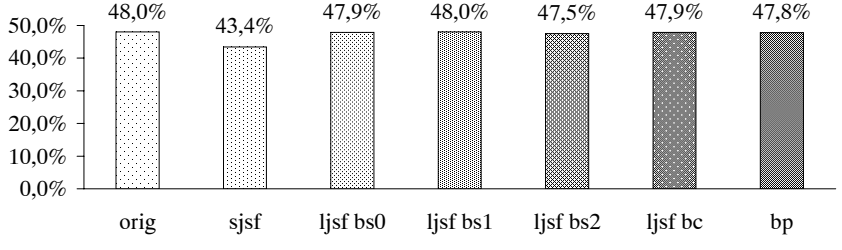


Abbildung 4.20: Intel Paragon SJSF, LJSF, BP: Gesamtauslastung l_{ges}

Die Gesamtauslastung l_{ges} (Abbilddung 4.20) des Systems durch die Belegung der Job-Scheduling-Strategie *Best Package* und der Varianten der Strategie *Largest Job-Size First* ist maximal 0,5% (*ljsf bs2*) schlechter als die der Originalbelegung. Dies entspricht einer um ca. 6 Tage längeren Gesamtlaufzeit. Die Gesamtauslastung der besten Variante ist geringfügig besser (*ljsf bs1* als durch die Originalbelegung. Die Strategie *Smallest Job-Size First* hat mit 43,4% eine um 4,5% schlechtere Gesamtauslastung als die Originalbelegung.

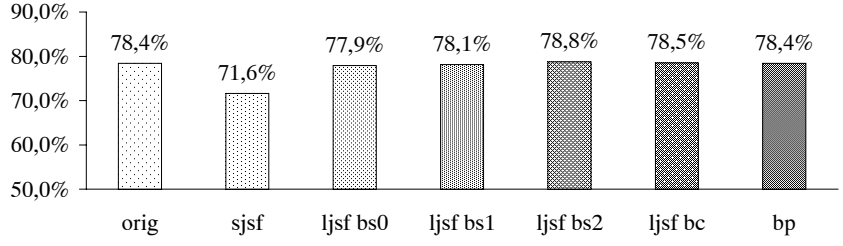


Abbildung 4.21: Intel Paragon SJSF, LJSF, BP: Auslastung in der aktiven Zeit l_{act}

Die in Abbildung 4.21 dargestellte Auslastung in der aktiven Zeit l_{act} liegt für die Varianten der Strategie *Largest Job-Size First* und der Strategie *Best Package* auf einem ähnlich hohen Wert wie der Vergleichswert. Die Auslastung in der aktiven Batch-Betriebszeit liegt für diese Strategien etwa einen Prozentpunkt unter der Auslastung, die durch die Originalbelegung erzielt wird. Auf Grund der hohen Auslastungen dieser Varianten, insbesondere der *ljsf bs1* Variante, wurden die Back-Filling Verfahren für die Strategie *Largest Job-Size First* genauer untersucht und verbessert. Da die Variante *ljsf bs1* nicht ohne weiteres zu realisieren ist – sie nutzt die exakte Ausführungszeit eines Jobs zur Fairneßbetrachtung des Back-Filling und somit zur Auswahl des zu startenden Jobs – wurden andere Möglichkeiten gesucht, um Back-Filling Verfahren ohne Fairneßbetrachtung an die Fairneß durch Betrachtung der Job-Laufzeit anzupassen. Als Ergebnis ergaben sich folgende Einschränkungen der Back-Filling Verfahren:

1. Die zeitliche Begrenzung des Back-Fillings. Das bedeutet, daß der erste Job in der Warteschlange durch Back-Filling nur für eine bestimmte Zeit blockiert werden darf. Läuft diese Zeit ab, so darf Back-Filling erst dann wieder ausgeführt werden, wenn der erste Job der Warteschlange durch die Strategie gestartet wurde. Im Anschluß daran startet die Back-Filling Phase erneut. Hier wird nun die Strategie mit zeitlicher Begrenzung des Back-Filling betrachtet, die die besten Ergebnisse bezüglich der Auslastung des massiv-parallelen Systems Intel Paragon erzielt. Dies ist die Strategie *ljsf bs3 t: 8h*, mit einer zeitlichen Begrenzung von 8 Stunden. Das *bs3* Back-Filling startet zunächst, falls möglich, alle Jobs der verschiedenen *Sequential-Queues*, anschließend den größten passenden Job. Diese Variante wird im weiteren als *ljsf bs3t* bezeichnet.

2. Die maximale Fragmentierung durch Back-Filling. Dieser Fragmentierungsgrad ist ein Wert, der angibt wieviel Knoten trotz Back-Filling unbenutzt bleiben dürfen. Bleiben mehr Knoten frei als die maximale Fragmentierung zuläßt, wird das Back-Filling eingestellt, bis der erste Job der Warteschlange durch die Job-Scheduling-Strategie gestartet wird. Die besten Werte liefert die Variante mit *Best Combination* Back-Filling und einer maximalen Fragmentierung von 40% (55 Knoten auf Intel Paragon), die im weiteren als *ljsf bcf* bezeichnet wird.

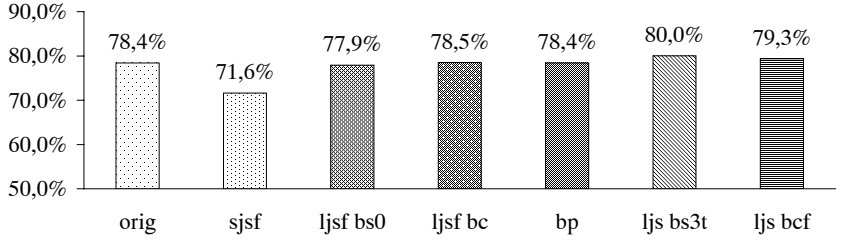


Abbildung 4.22: Intel Paragon SJSF, LJSF, BP: Auslastung in der aktiven Zeit l_{act}

Die mit diesen Verbesserungen erzielte Auslastung in der aktiven Zeit l_{act} ist in Abbildung 4.22 dargestellt. Die Differenz von 1,6% zwischen *ljsf bs3t* und *orig* entspricht einer um 7 Tage und 14 Stunden kürzeren aktiven Zeit. Die Auslastung in der aktiven Batch-Betriebszeit ist für die beiden optimierten Varianten minimal geringer als der Vergleichswert der Originalbelegung. Gleiches gilt auch für den Anteil der freien Batch-Betriebszeit an der Batch-Betriebszeit $T_{idle,bat}$. Dieses Kriterium ist für beide Varianten ebenfalls geringfügig schlechter als der Vergleichswert.

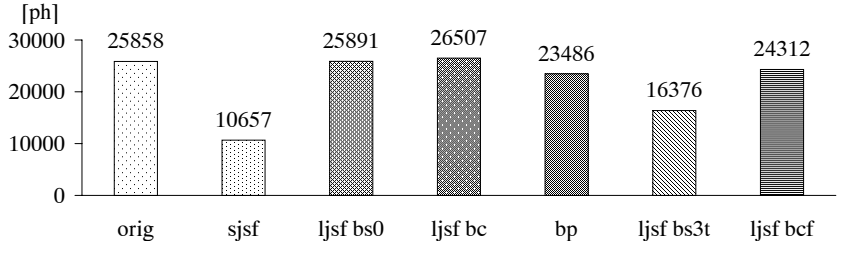


Abbildung 4.23: Intel Paragon SJSF, LJSF, BP: Last im interaktiven Betrieb L_{ia}

Die Last im interaktiven Betrieb L_{ia} ist nur für die nicht-optimierten Varianten *ljsf bs0* und *ljsf bc* der Strategie *Largest Job-Size First* geringfügig höher als in der Originalbelegung. Die optimierten Varianten der Strategie *Largest Job-Size First* sowie die Varianten der Strategien *Smallest Job-Size First* und *Best Package* führen zu

keiner stärkeren Beeinträchtigung des interaktiven Betriebs als die Originalbelegung (siehe Abbildung 4.23).

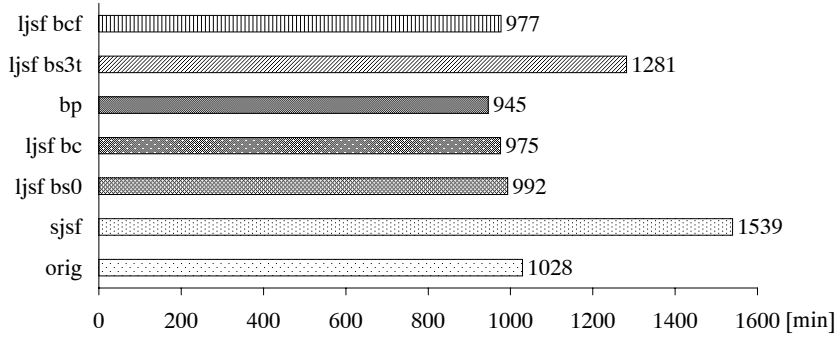


Abbildung 4.24: Intel Paragon SJSF, LJSF, BP: mittlere Wartezeit aller Jobs

Die mittlere Wartezeit aller Jobs liegt für die Strategie *Smallest Job-Sie First* und die Variante *ljsf bs3t* der Strategie *Largest Job-Size First* deutlich über dem Vergleichswert der Originalbelegung. Die anderen Varianten der Job-Scheduling-Strategie *Largest Job-Size First* sowie die Strategie *Best Package* liefern geringfügig niedrigere Werte. Hier ist eine Betrachtung der Job-Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ notwendig.

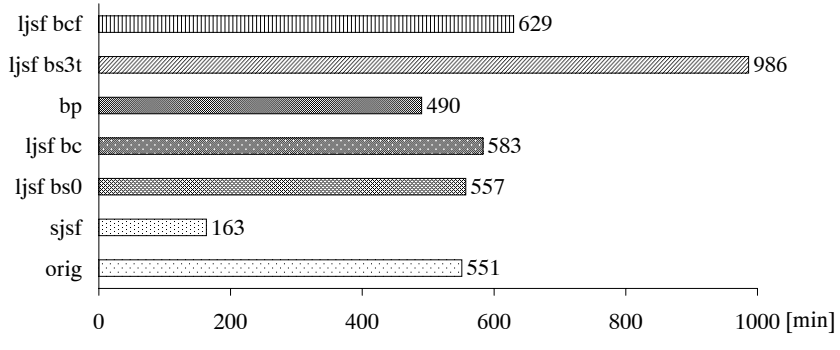


Abbildung 4.25: Intel Paragon SJSF, LJSF, BP: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

Die in Abbildung 4.25 dargestellte mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$ zeigt, daß die Strategie *Smallest Job-Size First*, wie zu erwarten war, einen deutlich niedrigeren Wert als die Originalbelegung liefert. Ebenfalls unter dem Vergleichswert liegt der Wert der Strategie *Best Package*. Alle Varianten der Strategie *Largest Job-*

Size First haben eine längere mittlere Wartezeit als die Originalbelegung. Der Wert der Variante *ljsf bs3t* ist fast doppelt so groß wie der Vergleichswert.

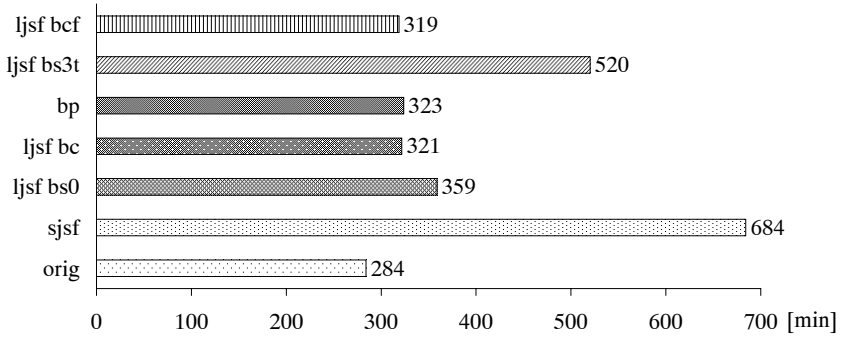


Abbildung 4.26: Intel Paragon SJSF, LJSF, BP: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

Die Klasse $C_{\leq 10min}$ ist ebenso wie die Klasse $C_{\leq 32p}$ von besonderer Bedeutung (siehe Abschnitt 4.1). Abbildung 4.26 zeigt die mittleren Wartezeiten dieser Klasse. Alle vorgestellten Varianten haben in der Klasse $C_{\leq 10min}$ längere mittlere Wartezeiten als die Originalbelegung.

Die mittleren Wartezeiten der Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ sind für die meisten der in diesem Abschnitt betrachteten Varianten geringfügig höher als die mittleren Wartezeiten der Originalbelegung. Aus diesem Grund wird eine Analyse der Quantile der Wartezeiten dieser Varianten in den Klassen C_{all}, $C_{\leq 32p}$ und $C_{\leq 10min}$ exemplarisch an den Monaten Dezember 1995, dem Monat mit der höchsten Last, und Oktober 1996, einem Monat mit durchschnittlicher Last, durchgeführt.

Die Gesamtauslastung der Originalbelegung l_{ges} erreichte im Dezember 1995 70,5% und im Oktober 1996 49,0%. Aufgrund der hohen Auslastung im Dezember 1995 sind für diesen Monat höhere Werte der Q_{90} und Q_{95} Quantile der Wartezeiten sowie der maximalen Wartezeit zu erwarten.

Die Abbildungen 4.27 und 4.28 zeigen die Quantile der Wartezeiten aller Jobs für die beiden ausgewählten Monate. In den Balken der einzelnen Strategien gibt der rechte Rand eines Bereiches den Wert des entsprechenden Quantils an. Die exakten Werte der Quantile sind zusätzlich in Tabelle 4.2 aufgelistet. Im Dezember 1995 wurden insgesamt 878 Jobs im Batch-Betrieb ausgeführt. Der Durchschnittswert beträgt 684 Jobs pro Monat (14353 Jobs in 21 Monaten). Die Zahl der ausgeführten Jobs im Oktober liegt mit 674 knapp unter dem Durchschnittswert.

Im Dezember 1995 führt die Variante *ljsf bs3t* zu deutlich höheren Werten der einzelnen Quantile. Die Strategie *Best Package* führt 95% aller Jobs im Dezember 1995 nach einer Wartezeit von maximal 130 Stunden aus. Dies sind 5 Stunden mehr als

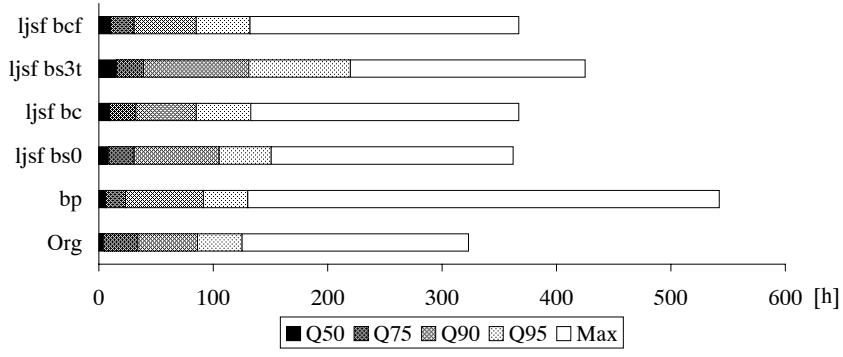


Abbildung 4.27: Intel Paragon LJSF, BP: Quantile der Wartezeiten aller Jobs im Dezember 1995

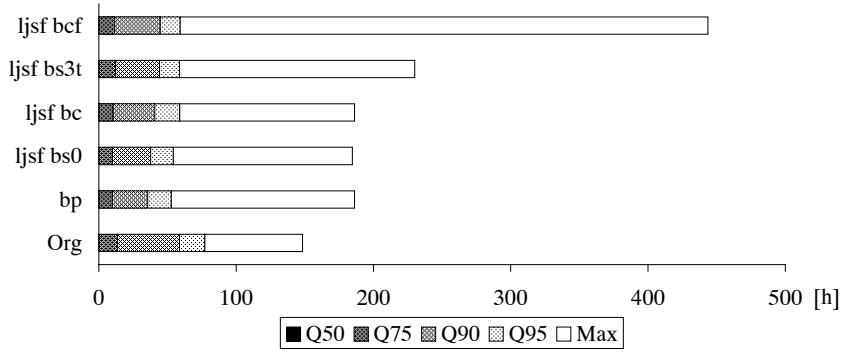


Abbildung 4.28: Intel Paragon LJSF, BP: Quantile der Wartezeiten aller Jobs im Oktober 1996

in der Originalbelegung. Die letzten 5% aller Jobs warten bei Anwendung dieser Strategie bis zu 542 Stunden (22 Tage). Das sind 219 Stunden (9 Tage) länger als in der Originalbelegung.

Im Oktober 1996 werden 50% aller Jobs durch alle Varianten und auch in der Originalbelegung direkt nach ihrer Ankunft im System gestartet. Die Q_{50} Quantile der Wartezeiten haben den Wert Null. 95% aller Jobs werden von den hier betrachteten Strategien spätestens nach 59 Stunden zur Ausführung gebracht. In der Originalbelegung liegt das Q_{95} Quantil bei 77 Stunden. Die maximale Wartezeit ist für alle Varianten höher als in der Originalbelegung.

Die Quantile der Wartezeiten der Jobs, die maximal 32 Knoten anfordern (Klasse $C_{\leq 32p}$), sind in den Abbildungen 4.30 (Dezember 1995) und 4.31 (Oktober 1996)

Quantile der Wartezeiten in der Klasse C_{all} [h]										
	Dezember 1995					Oktober 1996				
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	4	34	86	125	323	0	14	59	77	148
bp	6	23	91	130	542	0	10	36	53	186
ljsf bs0	9	31	105	151	362	0	10	38	54	185
ljsf bc	9	32	85	133	367	0	10	41	59	186
ljsf bs3t	15	39	131	220	425	0	12	44	59	230
ljsf bcf	10	31	85	132	367	0	12	45	59	444

Tabelle 4.2: Intel Paragon LJSF, BP: Quantile der Wartezeiten aller Jobs Dezember 1995, Oktober 1996

Quantile der Wartezeiten in der Klasse $C_{\leq 32p}$ [h]										
	Dezember 1995					Oktober 1996				
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	2	16	45	90	132	0	0	14	36	85
bp	3	15	36	92	162	0	0	12	28	53
ljsf bs0	4	20	52	106	168	0	0	13	25	51
ljsf bc	5	22	60	113	158	0	0	12	28	59
ljsf bs3t	14	29	115	220	425	0	0	18	35	85
ljsf bcf	6	21	58	113	158	0	0	16	33	85

Tabelle 4.3: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 32p}$ Dezember 1995, Oktober 1996

dargestellt. Die einzelnen Werte können der Tabelle 4.3 entnommen werden. In dieser Klasse wurden im Dezember 1995 711 der 878 Jobs ausgeführt, im Oktober 1996 514 der 674 Jobs. Durchschnittlich wurden jeden Monat 509 Jobs in der Klasse $C_{\leq 32p}$ ausgeführt.

Die Strategie *Best Package* führt als einzige der hier betrachteten Varianten für die Klasse $C_{\leq 32p}$ im Dezember 1995 zu Quantilen der Wartezeiten, die denen der Originalbelegung nahekommen. Der Maximalwert der Wartezeiten ist allerdings auch für diese Strategie wesentlich höher als der Vergleichswert.

Im Oktober 1996 werden 75% aller Jobs der Klasse $C_{\leq 32p}$ unmittelbar nach ihrem Eintreffen im System ausgeführt. Dies gilt für alle Varianten der Strategie *Largest Job-Size First* sowie für die Strategie *Best Package* und die Originalbelegung. Der Vergleichswert des Q_{95} Quantils und des Maximums der Wartezeiten dieser Gruppe wird im Oktober 1996 von den nichtoptimierten Varianten der Strategie *Largest Job-Size First* (*ljsf bs0* und *ljsf bc*) sowie von der Strategie *Best Package* deutlich unterboten. Die optimierten Varianten *ljsf bs3t* und *ljsf bcf* führen zu ähnlichen Werten wie die Originalbelegung.

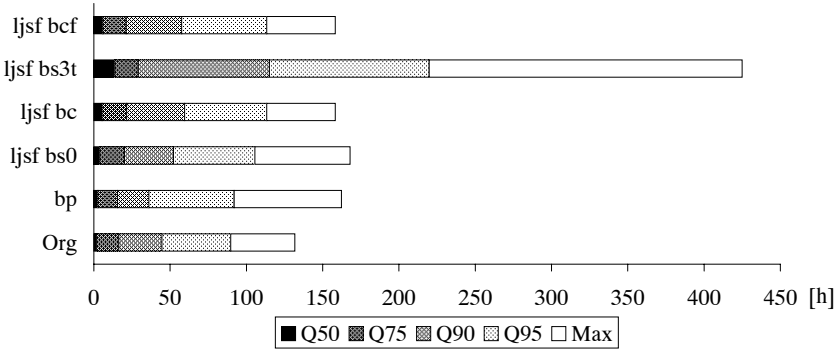


Abbildung 4.29: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 32p}$ Dezember 1995

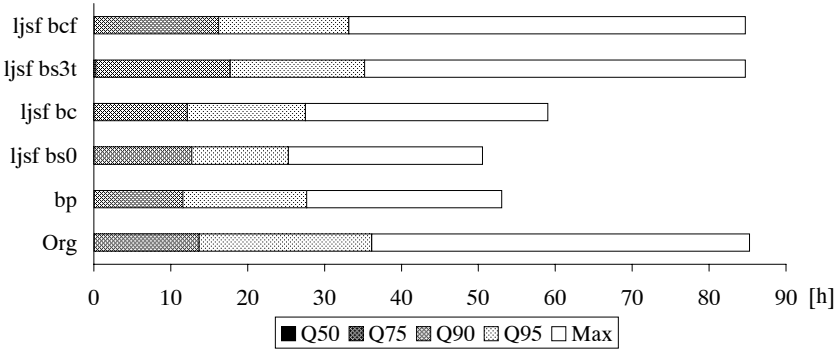


Abbildung 4.30: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 32p}$ Oktober 1996

Die letzte der drei Klassen ist die Klasse $C_{\leq 10min}$. Im Dezember 1995 wurden 86 von 878 Jobs in dieser Klasse ausgeführt, im Oktober 1996 473 von 674 Jobs. Der Durchschnitt beträgt 247 Jobs in der Klasse $C_{\leq 10min}$ von insgesamt 684 Jobs im Monat. In dieser Klasse befinden sich viele der Jobs, die durch die Sonderregelung für Testzwecke (siehe Abbschnitt 3.1.2) im interaktiven Betrieb ausgeführt wurden. Da diese Jobs unmittelbar nach ihrem Eintreffen im System gestartet werden, sind in dieser Klasse für die Quantile Q_{50} und Q_{75} Nullwerte zu erwarten.

Im Dezember 1995 treten die erwarteten Nullwerte für die Q_{75} Quantile nicht auf. Dies ist darauf zurückzuführen, daß in diesem Monat nur wenige Jobs zu Testzwecken im interaktiven Betrieb gestartet wurden, was auch an der geringen Zahl der Jobs in dieser Klasse (86 von 878) zu erkennen ist. Die Maximalwerte und die Q_{95} Quantile der Wartezeiten sind für alle betrachteten Varianten deutlich schlechter als die Werte

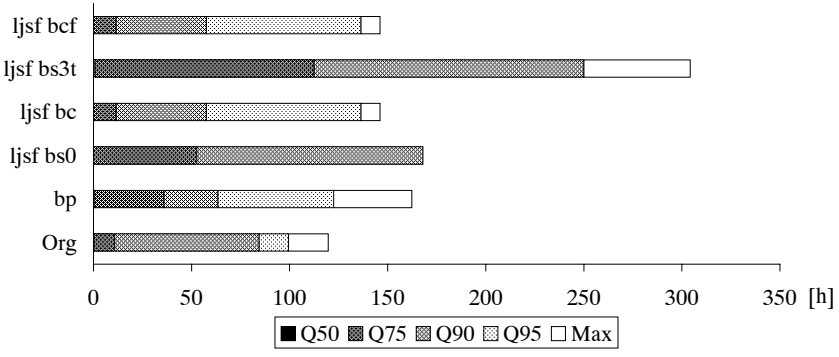


Abbildung 4.31: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 10min}$ Dezember 1995

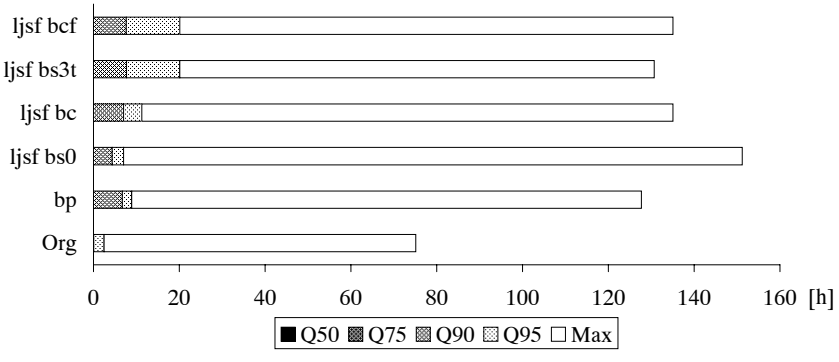


Abbildung 4.32: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 10min}$ Oktober 1996

der Originalbelegung.

Die Quantile Q_{90} und Q_{95} der Strategie *Best Package* (*bp*) und der *ljsf bs0* und *ljsf bc* Varianten sind im Oktober 1996 zwar deutlich höher als die Werte der Originalbelegung, dies ist aber insofern vertretbar, als diese Jobs innerhalb einer Batch-Betriebsphase von 20:00 Uhr bis zum nächsten morgen um 8:00 Uhr ausgeführt wurden. Die deutlich höheren maximalen Wartezeiten hingegen stellen einen klaren Nachteil aller Varianten dar.

Die Strategien *Smallest Job-Size First*, *Largest Job-Size First* und *Best Package* lassen sich folgendermaßen zusammenfassen:

Smallest Job-Size First führt zu einer niedrigen Auslastung und hohen Wartezeiten in allen Klassen mit Ausnahme der Klasse der Jobs, die maximal 32 Knoten

Quantile der Wartezeiten in der Klasse $C_{\leq 10min}$ [h]										
	Dezember 1995					Oktober 1996				
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	0	11	84	99	120	0	0	0	2	75
bp	0	36	64	122	162	0	0	7	9	128
ljsf bs0	0	53	168	168	168	0	0	4	7	151
ljsf bc	0	12	58	136	146	0	0	7	11	135
ljsf bs3t	1	112	250	250	304	0	0	8	20	131
ljsf bcf	0	12	58	136	146	0	0	8	20	135

Tabelle 4.4: Intel Paragon LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 10min}$ Dezember 1995, Oktober 1996

anfordern. Die Job-Scheduling-Strategie *Largest Job-Size First* führt zu einer hohen Auslastung des Systems. Die Wartezeiten der Jobs liegen in allen Klassen nur geringfügig über denen der Originalbelegung. Einzige Ausnahme bildet die Klasse der Jobs mit einer Laufzeit von höchstens 10 Minuten. In dieser Klasse werden 5 bis 10% der Jobs deutlich später ausgeführt als in der Originalbelegung. Die Auslastung einiger Varianten dieser Strategie läßt sich durch Einschränkung des Back-Filling verbessern, so daß die Auslastung durch die Originalbelegung übertroffen werden kann. Diese Verbesserung führt allerdings zu längeren Wartezeiten insbesondere in den Klassen $c_{\leq 32p}$ und $C_{\leq 10min}$. Die Belegung durch die Strategie *Best Package* erreicht eine ebenso hohe Auslastung wie die Originalbelegung. Für die Wartezeiten der Jobs liefert diese Strategie ebenfalls gute Resultate. Eine Ausnahme bildet auch hier die Klasse $C_{\leq 10min}$, in der, wie bei den Varianten der Strategie *Largest Job-Size First*, ca. 5 bis 10% der Jobs deutlich später ausgeführt werden.

4.2.4 Smallest Cumulative Demand First, Largest Cumulative Demand First

Der letzte Abschnitt der Ergebnisse, die mit der Belegung des Systems Intel Paragon erzielt wurden, befaßt sich mit den Job-Scheduling-Strategien, die das Produkt aus $Knoten \times Laufzeit$ der Jobs als Sortierkriterium nutzen. Dies sind die Strategien *Smallest Cumulative Demand First* (*SCDF*) und *Largest Cumulative Demand First* (*LCDF*). Folgende Varianten werden hier betrachtet:

- Smallest Cumulative Demand First ohne Back-Filling (*scdf*)
- Smallest Cumulative Demand First mit First-Come-Back-Filling ohne Fairneßbetrachtung (*scdf fc0*)
- Smallest Cumulative Demand First mit First-Come-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*scdf fc1*)

- Largest Cumulative Demand First ohne Back-Filling (*lcdf*)
- Largest Cumulative Demand First mit Best-Demand-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lcdf bd1*)
- Largest Cumulative Demand First mit Best-Combination-Back-Filling ohne Fairneßbetrachtung (*lcdf bc*)

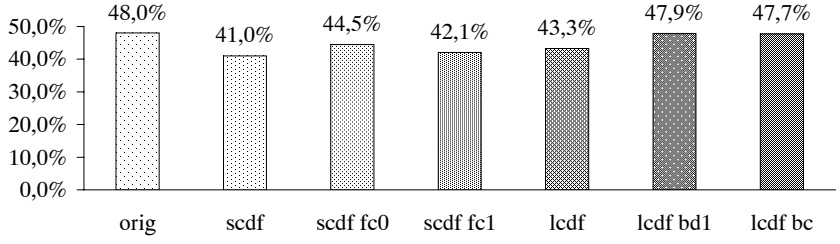


Abbildung 4.33: Intel Paragon SCDF, LCDF: Gesamtauslastung l_{ges}

Die Gesamtauslastung l_{ges} (Abbildung 4.33) der Varianten der Strategie *Smallest Cumulative Demand First* und der Strategie *Largest Cumulative Demand First* ohne Back-Filling liegen zwischen 41,0% und 44,5%. Das entspricht einer zwischen 107 und 48 Tagen längeren Gesamtlaufzeit T_{ges} als die Originalbelegung. Die beiden Varianten der Strategie *Largest Cumulative Demand First* mit Back-Filling haben eine um 3 Tage (*lcdf bd1*) beziehungsweise 1 Tag (*lcdf bc*) längere Gesamtlaufzeit.

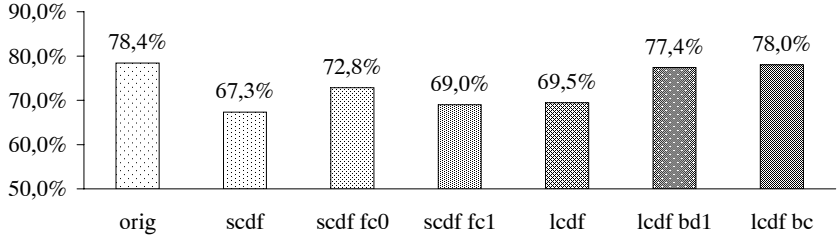


Abbildung 4.34: Intel Paragon SCDF, LCDF: Auslastung in der aktiven Zeit l_{act}

Ähnliche Verhältnisse wie die Gesamtauslastung l_{ges} liefert auch die Auslastung in der aktiven Zeit l_{act} (Abbildung 4.34) und die Auslastung während der Batch-Betriebszeit $l_{act,bat}$, die hier nicht dargestellt wird.

Die Last im interaktiven Betrieb L_{ia} liefert keine überraschenden Ergebnisse und wird deshalb ebenfalls nicht dargestellt. Alle Varianten bleiben unter dem Wert der

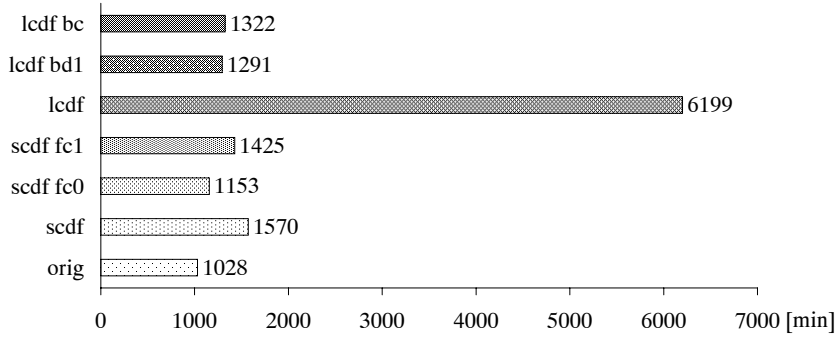


Abbildung 4.35: Intel Paragon SCDF, LCDF: mittlere Wartezeit aller Jobs

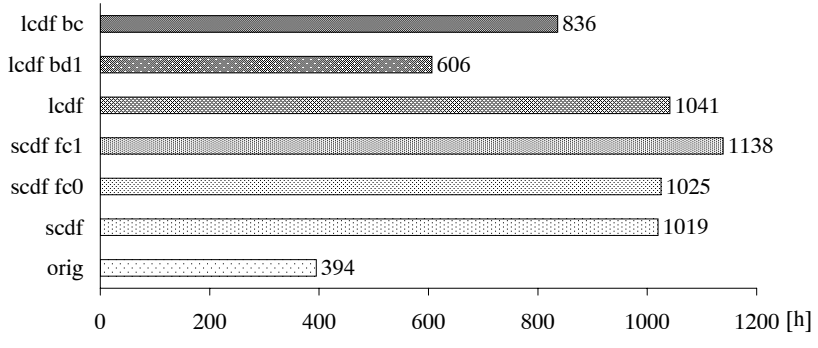


Abbildung 4.36: Intel Paragon SCDF, LCDF: maximale Wartezeit aller Jobs

Originalbelegung. Die beiden Varianten *lcdf bd1* und *lcdf bc* kommen dem Wert der Originalbelegung am nächsten.

Die mittlere Wartezeit aller Jobs (Abbildung 4.35) liegt für alle Varianten der Strategien *Smallest Cumulative Demand First* und *Largest Cumulative Demand First* deutlich über dem Vergleichswert der Originalbelegung.

In den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ erreichen die Varianten der Strategie *Smallest Cumulative Demand First* eine kürzere mittlere Wartezeite als die Originalbelegung. Die Varianten der Strategie *Largest Cumulative Demand First* führen zu längeren mittleren Wartezeiten in diesen beiden Klassen.

Zusammenfassend stellen sich die Strategien *Smallest Cumulative Demand First* und *Largest Cumulative Demand First* folgendermaßen dar:

Die Belegung des Systems Intel Paragon unter Anwendung der Strategie *Smallest Cumulative Demand First* führt zu einer deutlich schlechteren Auslastung als die

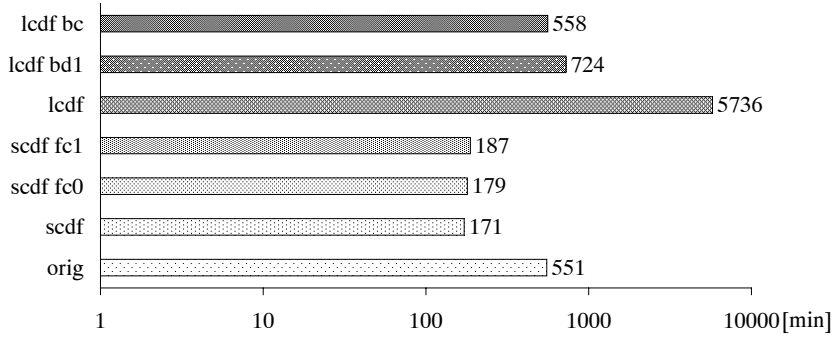


Abbildung 4.37: Intel Paragon SCDF, LCDF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

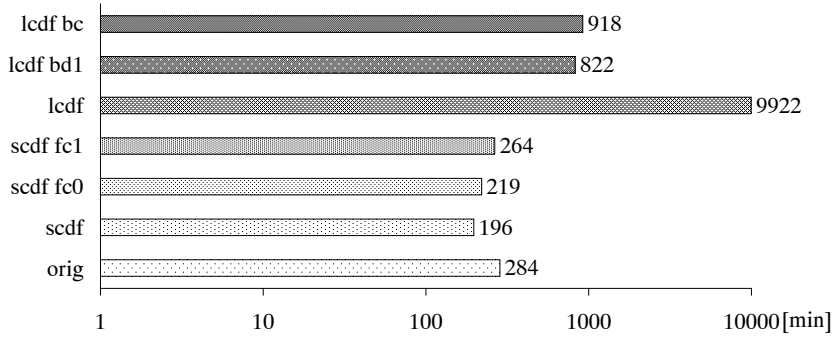


Abbildung 4.38: Intel Paragon SCDF, LCDF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

Originalbelegung. Die Wartezeiten der Jobs in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ sind durchschnittlich kürzer als in der Originalbelegung. Der Mittelwert der Wartezeiten aller Jobs liegt allerdings für alle Varianten der Strategie *Smallest Cumulative Demand First* über dem Vergleichswert. Die Variante der Strategie *Largest Cumulative Demand First* ohne Back-Filling führt zu einer niedrigen Auslastung des massivparallelen Systems und zu extrem langen Wartezeiten der Jobs in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$. Die Varianten mit Back-Filling – *lcdf bd1* und *lcdf bc* – erzielen eine gute Auslastung, aber die mittlere Wartezeit aller Jobs ist länger als in der Originalbelegung. Gleiches gilt auch für die mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$. Lediglich in der Klasse $C_{\leq 32p}$ erreicht die Variante *lcdf bc* annähernd den Vergleichswert der Originalbelegung.

4.3 Analyse der Belegung von Cray T3E

Die Belegung des massiv-parallelen Systems Cray T3E in den Monaten März bis Mai 1997 weist die in der folgenden Liste aufgeführten Job-Zahlen in den einzelnen Klassen auf:

- Gesamtanzahl der Jobs, Jobs der Klasse C_{all}: 3840
- Anzahl der Jobs in der Klasse $C_{\leq 32p}$: 2113
- Anzahl der Jobs in der Klasse $C_{>32p}$: 1727
- Anzahl der Jobs in der Klasse $C_{\leq 10min}$: 1215
- Anzahl der Jobs in der Klasse $C_{>10min}$: 2625

Der betrachtete Zeitraum umfaßt 91 Tage. Die Last der 3840 Jobs entspricht einer 44-tägigen Nutzung der 518 Knoten. Die weiteren Werte der Originalbelegung werden in den einzelnen Vergleichen vorgestellt.

Die Simulation der Belegung durch die Job-Scheduling-Strategien wird in dem betrachteten Zeitraum stark durch Systemunterbrechungen beeinflußt. In den Monaten März und April 1997 gab es jeweils ca. 50 Systemunterbrechungen, im Mai 1997 ca. 30. Durch die Unterbrechungen werden sowohl die Wartezeiten verlängert als auch die Auslastungen während der verschiedenen Zeiten verringert. Die Zeit der Unterbrechung wird nicht als aktive Zeit gerechnet, die Kriterien l_{act} und $l_{act,bat}$ werden also nicht durch lange Unterbrechungen verringert. Da jedoch alle Jobs, die durch eine Systemunterbrechung gestoppt werden, als nicht gestartet betrachtet und erneut in die Warteschlange eingefügt werden, kann die Auslastung vor einer Systemunterbrechung sehr gering sein.

Auf dem System Cray T3E müssen die Knoten, die ein Job zugewiesen bekommt, fortlaufend numeriert werden. Die Simulation wurde zunächst für jede Strategie unter Beachtung dieser Einschränkung durchgeführt. Zusätzlich wurde für jede Strategie die Belegung mit der Möglichkeit, Jobs zu migrieren (Abschnitt 3.2.7), simuliert. Die Migrations-Routine wird aufgerufen, wenn ein Job zwar genügend freie Knoten auf dem massiv-parallelen System vorfindet, aber der größte fortlaufend numerierte freie Bereich nicht genügend Knoten umfaßt. In der Migrations-Routine werden alle Jobs, die zum Zeitpunkt des Aufrufs auf dem System ausgeführt werden, soweit wie möglich nach vorne, also auf die Knoten mit kleiner Nummer, migriert. Für die Simualtion, die die Möglichkeit der Job-Migration berücksichtigt, werden zusätzlich erfaßt:

- Anzahl der Migrationen
- Anzahl der migrierten Jobs
- Anzahl der Knoten, die von Migrationen betroffen sind

Bei einer Implementierung der Migrations-Routine könnte beispielsweise die Anzahl der Knoten die die laufenden Jobs belegen benutzt werden, um nur einige der Jobs zu migrieren, indem zunächst nach laufenden Jobs gesucht wird, die kleinere freie Bereiche auffüllen. Wenn beispielsweise ein Job, der die Knoten 1 bis 8 belegt, beendet wird, werden durch die Migrations-Routine des Simulators alle laufenden Jobs um 8 Knoten nach vorne migriert. Falls nun der Job, der die letzten Knoten belegt, also die Knoten mit den höchsten Nummern, ebenfalls eine Knotenzahl von 8 hat, könnte es besser sein diesen Job auf die Knoten 1 bis 8 zu migrieren und alle anderen laufenden Jobs auf ihren bislang zugewiesenen Knoten weiter auszuführen. Dabei besteht jedoch die Gefahr, daß dieser Job ebenfalls bald beendet wird, so daß erneut Jobs migrieren müssen. Die Verfahren zur Migration der Jobs wurden in der vorliegenden Arbeit nicht betrachtet. Die Job-Migrationen werden nur zur Abschätzung des Einflußes auf die Auslastung analysiert. Dabei bleibt der Aufwand für die Migration unberücksichtigt. Mit dem in Kapitel 3 Abschnitt 3.2.7 vorgestellten zeitlichen Aufwand von $\frac{1}{10}$ Sekunde pro Knoten eines migrierenden Jobs ergibt sich in den folgenden Abschnitten keine Veränderung der Werte für die Systemauslastung. Dieser Wert muß jedoch durch weitere Messungen bestätigt werden.

Die Simulation der Belegung des Systems Cray T3E wird auf die Minute genau durchgeführt (siehe Abschnitt 3.2.1).

4.3.1 First Come First Serve

Die Ergebnisse der simulierten Belegung des massiv-parallelen Systems Cray T3E unter Anwendung der Job-Scheduling-Strategie *First Come First Serve* wird anhand folgender Varianten präsentiert:

- First Come First Serve ohne Back-Filling (*fcfs*)
- First Come First Serve mit First-Come-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*fcfs fc1*)
- First Come First Serve mit Best-Combination-Back-Filling ohne Fairneßbetrachtung (*fcfs bc*)

Für diese Varianten werden die Ergebnisse sowohl mit als auch ohne Migration betrachtet.

Die in Abbildung 4.39 dargestellte Auslastung des Systems Cray T3E in der aktiven Batch-Betriebszeit $l_{act,bat}$ zeigt nahezu die gleichen Verhältnisse der Varianten zur Originalbelegung wie die simulierte Belegung des Systems Intel Paragon. Die Variante ohne Back-Filling (*fcfs*) führt zu einer deutlich schlechteren Auslastung des massiv-parallelen Systems, während die Varianten mit Back-Filling (*fcfs fc1* und *fcfs bc*) das System nur geringfügig schlechter auslasten als die Originalbelegung.

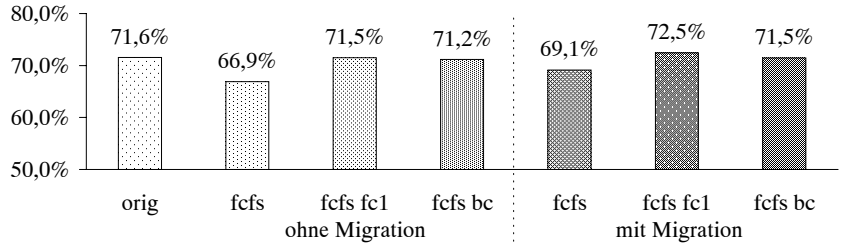


Abbildung 4.39: Cray T3E FCFS: Auslastung in der aktiven Batch-Betriebszeit $l_{act,bat}$

Variante	Migrationen	migrierte Jobs	migrierte Knoten
fcfs	167	921	43077
fcfs fc1	127	601	31713
fcfs bc	81	317	18129

Tabelle 4.5: Cray T3E FCFS: Migrationen

Die Auslastung in der aktiven Batch-Betriebszeit zeigt die gleichen Relationen wie Abbildung 4.39.

Die Simulation mit Migration ergibt im Kriterium Auslastung in der aktiven Zeit l_{act} für die Variante *fcfs* einen deutlich besseren Wert (2,2%) als ohne Migration. Die Varianten mit Back-Filling erreichen durch die Migration geringere Steigerungen (*fcfs fc1*: 1,0% und *fcfs bc*: 0,3%). Der Unterschied ist darauf zurückzuführen, daß die Varianten mit Back-Filling in der Simulation ohne Migration nach Jobs suchen, die die freien Bereiche des Systems ausnutzen. Falls also ein Job gefunden wird, für den genügend freie Knoten auf dem massiv-parallelen System bereit stehen, diese jedoch nicht in einem fortlaufend numerierten Bereich liegen, wird in der Warteschlange weiter nach einem Job gesucht, der gestartet werden kann. Somit werden durch diese Varianten ohne Migration andere Jobs gestartet als mit Migration. Die Variante *fcfs* hingegen startet ohne Migration keinen Job, wenn der erste Job der Warteschlange genügend freie Knoten vorfindet, diese aber nicht fortlaufend numeriert sind. Die Werte der durch die einzelnen Varianten verursachten Migrationen sind in Tabelle 4.5 zusammengestellt.

Die durch die einzelnen Varianten der Strategie *First Come First Serve* erreichte Gesamtauslastung l_{ges} (Abbildung 4.40) liegt für alle Varianten unterhalb der Gesamtauslastung, die durch die Originalbelegung des Systems Cray T3E erreicht wird. Die Varianten *fcfs fc1* und *fcfs bc* benötigen jedoch nur etwa 10 Stunden länger zur Ausführung aller Jobs als die Originalbelegung.

Die in den Abbildungen 4.41 und 4.42 dargestellten Lasten im gemischten (L_{mix}) und interaktiven Betrieb (L_{ia}) weisen für die Varianten *fcfs* und *fcfs bc* eine Erhöhung

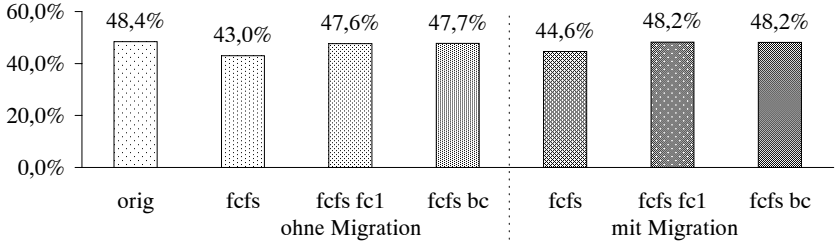


Abbildung 4.40: Cray T3E FCFS: Gesamtauslastung l_{ges}

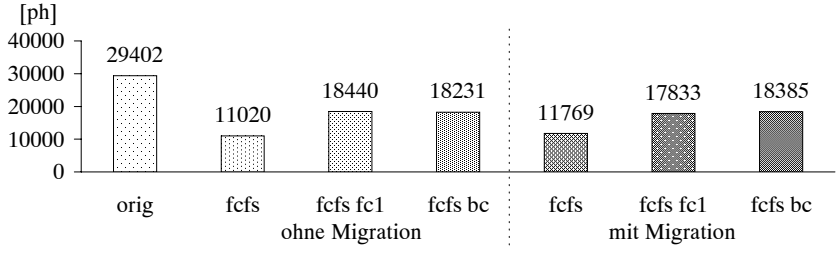


Abbildung 4.41: Cray T3E FCFS: Last im gemischten Betrieb L_{mix}

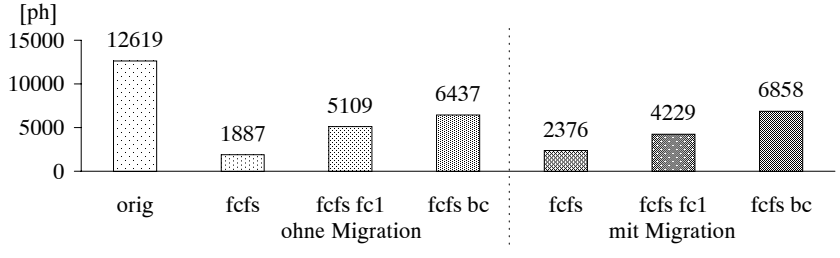


Abbildung 4.42: Cray T3E FCFS: Last im interaktiven Betrieb L_{ia}

der jeweiligen Last auf, wenn Migration zugelassen wird. Die Variante *fcfs fc1* hingegen führt in der Simulation mit Migration zu niedrigeren Lasten im gemischten und im interaktiven Betrieb. Dieser Effekt ist auf die Fairneßbetrachtung der Variante *fcfs fc1* zurückzuführen. Dies wird anhand des interaktiven Betriebs erläutert. Die Last im interaktiven Betrieb wird zum Teil durch Jobs verursacht, die in der vorhergehenden Betriebsart – dem gemischten Betrieb – gestartet wurden. Über die maximale Knotenzahl der Jobs (siehe Kapitel 3, Abschnitt 3.2.5) wird verhindert, daß Jobs mit einer großen Knotenanzahl – hier 256 Knoten – für längere Zeit im interaktiven Betrieb ausgeführt werden. Aufgrund der Fairneßbetrachtung der

fcfs fc1 Variante laufen alle über Back-Filling gestarteten Jobs maximal bis zum Ausführungsende der durch die Strategie gestarteten Jobs. Durch das Back-Filling-Verfahren der Variante *fcfs bc* hingegen werden auch noch kurz vor dem Übergang in den interaktiven Betrieb Jobs gestartet, die wesentlich länger ausgeführt werden als die Jobs, die durch die Strategie gestartet wurden, und durch Migration können nun Jobs mit grösserer Knotenanzahl als ohne Migration gestartet werden. Die Variante *fcfs* ohne Back-Filling kann, wie bereits erwähnt, durch Migration Jobs starten, die ohne Migration erst nach dem Ausführungsende laufender Jobs gestartet werden konnten. Dadurch wird auch die Last während des Übergangs in den interaktiven Betrieb erhöht, denn die im gemischten Betrieb gestarteten Jobs werden beim Übergang in den interaktiven Betrieb und der damit verbundenen Verkleinerung der Partition für den Batch-Betrieb von 383 Knoten auf 4 Knoten nicht abgebrochen.

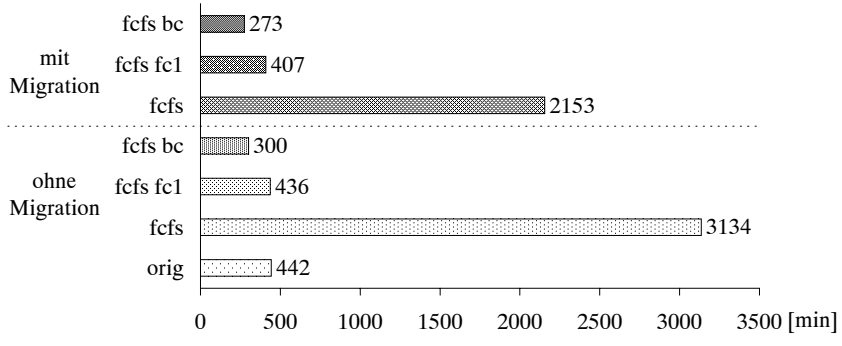


Abbildung 4.43: Cray T3E FCFS: mittlere Wartezeit aller Jobs

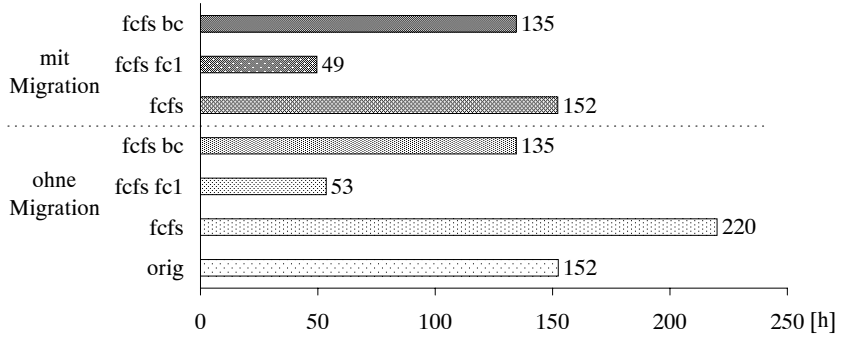


Abbildung 4.44: Cray T3E FCFS: maximale Wartezeit aller Jobs

Die Job-Scheduling-Strategie *First Come First Serve* in Verbindung mit dem *Best-Combination*-Back-Filling-Verfahren führt zu einer deutlich kürzeren mittleren War-

tezeit aller Jobs (Abbildung 4.43) als die Originalbelegung. In der Simulation mit Migration wird dieser Wert noch geringfügig verbessert. Die mittlere Wartezeit, die durch die Variante *fcfs fc1* erreicht wird, entspricht nahezu der Originalbelegung.

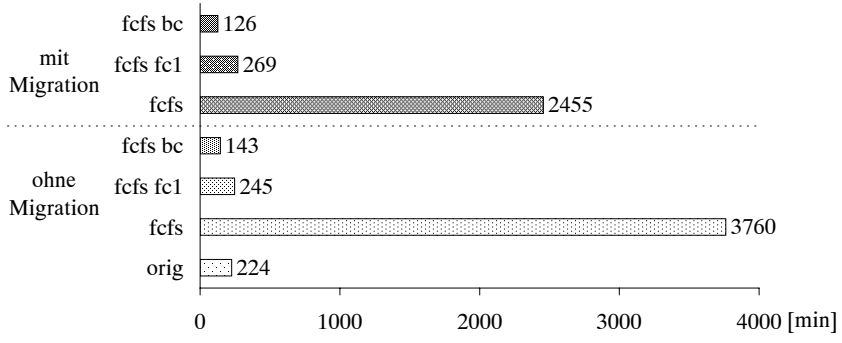


Abbildung 4.45: Cray T3E FCFS: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

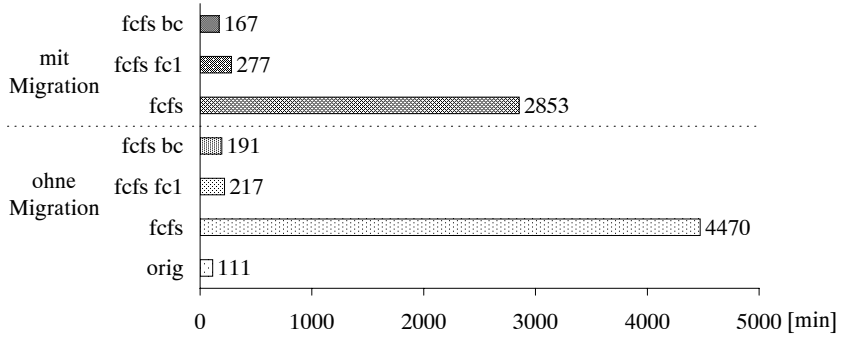


Abbildung 4.46: Cray T3E FCFS: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

Die mittlere Wartezeit in der Klasse der Jobs mit maximal 32 Knoten ist für die Variante *fcfs bc* wesentlich kürzer als in der Originalbelegung. Die Variante *fcfs fc1* überschreitet für dieses Kriterium den Vergleichswert. In der Klasse der Jobs mit maximal 10 Minuten Laufzeit liefern alle Varianten sowohl mit als auch ohne Migration schlechtere Werte als die Originalbelegung. Die Quantile der Wartezeiten der Variante *fcfs bc* werden in Abschnitt 4.3.3 zusammen mit den Varianten der Strategien *Shortest Job-Size First*, *Largest Job-Size First* und *Best Package* betrachtet.

Zusammenfassend läßt sich die Strategie *First Come First Serve* auf dem massiv-parallelen System Cray T3E wie folgt bewerten:

Die Auslastung des massiv-parallelen Systems unter Verwendung der Varianten mit Back-Filling erreichen ein hohes Niveau. Durch Job-Migration erreichen die Werte der Originalbelegung. Die mittlere Wartezeit aller Jobs sowie der Jobs mit maximal 32 Knoten wird durch die Variante *fcfs bc* deutlich verringert. In der Klasse der Jobs mit maximal 10 Minuten Laufzeit erreicht keine Variante der Strategie *First Come First Serve* die mittlere Wartezeit der Originalbelegung.

4.3.2 Shortest Processing Time First, Largest Processing Time First

Die besten Ergebnisse in den einzelnen Kriterien liefern für die *Shortest Processing Time First* und die Job-Scheduling-Strategie *Largest Processing Time First* die folgenden Varianten:

- Shortest Processing Time First mit Best-Demand-Back-Filling ohne Fairneßbetrachtung (*sptf bd0*)
- Largest Processing Time First mit Best-Demand-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lptf bd1*)
- Largest Processing Time First mit Best-Combination-Back-Filling ohne Fairneßbetrachtung (*lptf bc*)

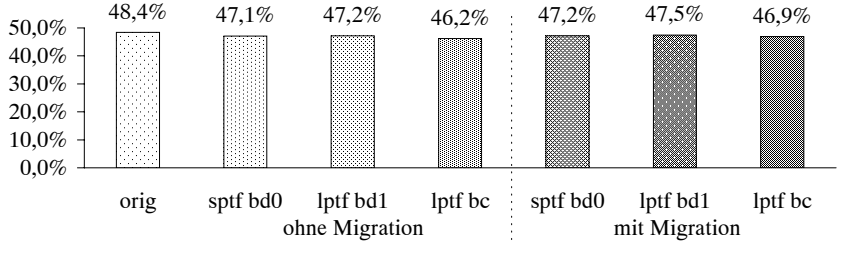


Abbildung 4.47: Cray T3E SPTF, LPTF: Gesamtauslastung l_{ges}

Sowohl die Gesamtauslastung l_{ges} (Abbildung 4.47) als auch die Auslastung in der aktiven Zeit l_{act} (Abbildung 4.48) des massiv-parallelen Systems Cray T3E zeigt, daß keine Variante der Job-Scheduling-Strategien *Shortest Processing Time First* und *Largest Processing Time First* für dieses Kriterium den Wert der Originalbelegung erreicht. Wie bei der simulierten Belegung des massiv-parallelen Systems Intel Paragon bleiben die Werte für dieses Bewertungskriterium auch hinter den Werten

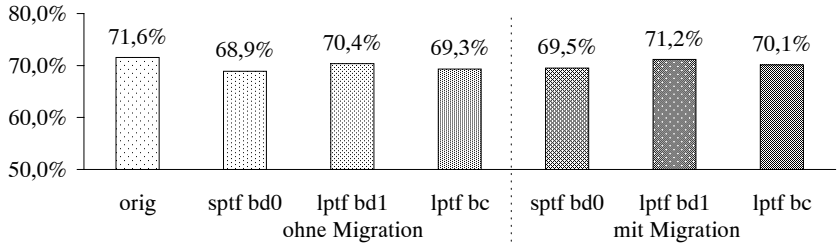


Abbildung 4.48: Cray T3E SPTF, LPTF: Auslastung in der aktiven Zeit l_{act}

der Strategie *First Come First Serve* zurück. Gleiches gilt auch für die Kriterien Gesamtauslastung l_{ges} und Auslastung in der aktiven Batch-Betriebszeit $l_{act,bat}$. Die Anzahl der Migrationen, migrierten Jobs und der betroffenen Knoten (Tabelle 4.6) liegen für die Varianten der Strategie *Largest Processing Time First* auf dem gleichen Niveau wie für die Variante *fcfs bc*. Für die Strategie *Shortest Processing Time First* ergeben sich etwas höhere Werte.

Variante	Migrationen	migrierte Jobs	migrierte Knoten
sptf bd0	94	416	24096
lptf bd1	78	288	17806
lptf bc	85	308	18607

Tabelle 4.6: Cray T3E SPTF, LPTF: Migrationen

Die Lasten im gemischten (L_{mix}) und im interaktiven Betrieb (L_{ia}) sind für alle Varianten niedriger als die Lasten, die durch die Originalbelegung in diesen Betriebsarten erreicht werden, so daß diese Bewertungskriterien keinen Nachteil der Strategien *Shortest Processing Time First* und *Largest Processing Time First* darstellen.

Die in Abbildung 4.49 dargestellte mittlere Wartezeit aller Jobs ist für die Belegung des Systems durch die Variante *sptf bd0* sowohl mit als auch ohne Migration deutlich kürzer als die mittlere Wartezeit aller Jobs in der Originalbelegung. Die Varianten der Job-Scheduling-Strategie *Largest Processing Time First* führen nur zusammen mit Job-Migration zu geringfügig besseren Werten als die Originalbelegung. Diese Werte sind jedoch wesentlich schlechter als die der Variante *fcfs bc* (ohne Migration 300 Minuten, mit Migration 273 Minuten), so daß auch in diesem Kriterium kein Vorteil gegenüber der Strategie *First Come First Serve* erzielt wird.

Die mittlere Wartezeit in der Klasse der Jobs mit maximal 32 Knoten (Abbildung 4.51) ist für alle hier betrachteten Varianten deutlich kürzer als in der Originalbelegung. Die durch die beste Variante der Strategie *First Come First Serve* (*fcfs bc*) erreichten Werte – 143 Minuten ohne Migration beziehungsweise 126 Minuten mit Migration – werden ebenfalls deutlich unterboten. In der Klasse $C_{\leq 10min}$

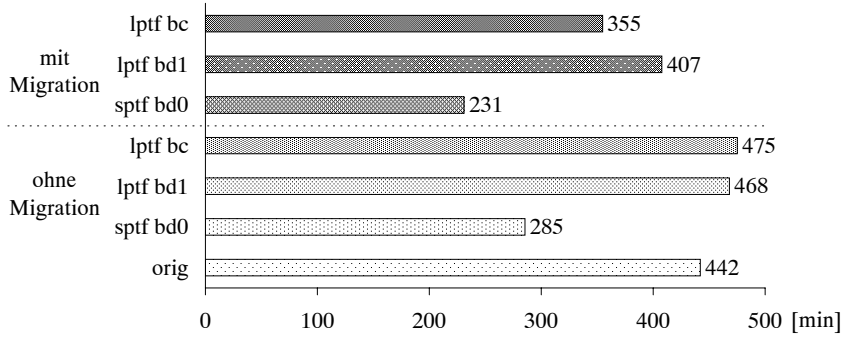


Abbildung 4.49: Cray T3E SPTF, LPTF: mittlere Wartezeit aller Jobs

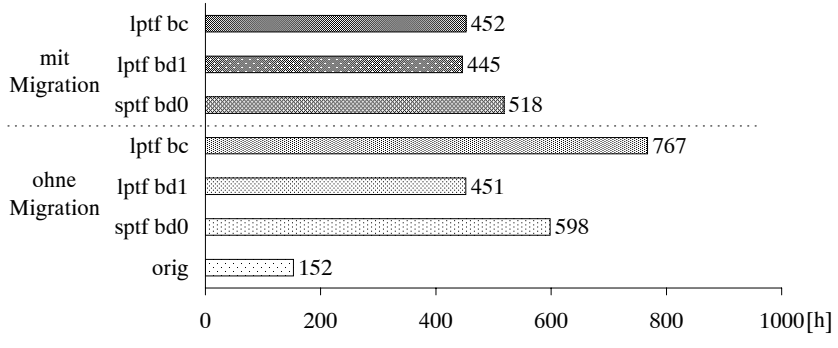


Abbildung 4.50: Cray T3E SPTF, LPTF: maximale Wartezeit aller Jobs

führt lediglich die Variante *sptf bd0* unter Verwendung von Job-Migration zu einer kurzen mittleren Wartezeit. Alle anderen Varianten mit und ohne Job-Migration führen zu einer höheren mittleren Wartezeit als bei der Originalbelegung.

Die beiden Strategien ergeben zusammengefaßt folgendes Bild:

Shortest Processing Time First erreicht kurze mittlere Wartezeiten in den betrachteten Job-Klassen, führt jedoch zu einer niedrigeren Auslastung des massiv-parallelen Systems als die Originalbelegung. Die Strategie *Largest Processing Time First* erreicht relativ gute Werte für die Auslastung des Systems, diese Werte liegen jedoch unter denen, die durch die Strategie *First Come First Serve* erreicht werden. Die mittlere Wartezeit aller Jobs sowie der Jobs mit maximal 10 Minuten Laufzeit zeigt, daß diese Strategie der Job-Scheduling-Strategie *First Come First Serve* auf dem massiv-parallelen System in diesen Kriterien klar unterlegen ist. In der Klasse der Jobs mit maximal 32 Knoten erreichen die Varianten der Strategie *Largest Processing Time First* jedoch sehr kurze Wartezeiten.

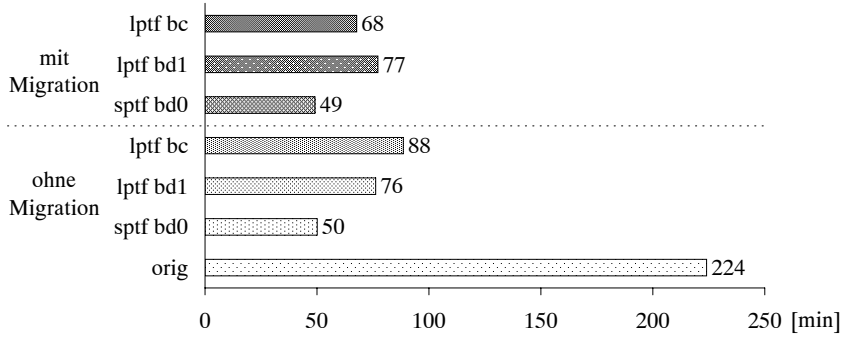


Abbildung 4.51: Cray T3E SPTF, LPTF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

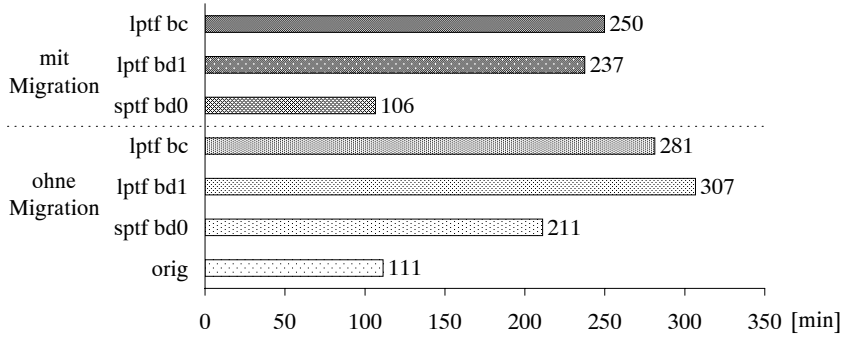


Abbildung 4.52: Cray T3E SPTF, LPTF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

4.3.3 Smallest Job-Size First, Largest Job-Size First, Best Package

Die Job-Scheduling-Strategien, die die Warteschlange anhand der vom Job geforderten Knotenzahl sortieren, werden in diesem Abschnitt betrachtet. Folgende Varianten wurden ausgewählt:

- Smallest Job-Size First ohne Back-Filling (*sjsf*)
- Largest Job-Size First ohne Back-Filling (*ljsf*)

- Largest Job-Size First mit Best-Size-Back-Filling ohne Fairneßbetrachtung und einer maximalen Fragmentierung des massiv-parallelen Systems von 60% (*ljsf bs0f*)
- Largest Job-Size First mit Best-Size-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*ljsf bs1*)
- Largest Job-Size First mit Worst-Size-Back-Filling ohne Fairneßbetrachtung (*ljsf ws0*)
- Best Package (*bp*)

Auf Grund der Auslastungs-Ergebnisse der Strategie *Largest Job-Size First* mit *Best-Size*-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*ljsf bs1* wurde, wie schon für Intel Paragon, versucht durch Einschränkungen des Back-Fillings ohne Fairneßbetrachtung diese Werte zu erreichen. Die beste Auslastung wurden dabei mit der hier als *ljsf bs0f* bezeichneten Variante erzielt, die eine maximale Fragmentierung des Systems von 60% (310 Knoten im Batch-Betrieb, 229 Knoten im gemischten Betrieb auf Cray T3E) zuläßt. Sollten durch Back-Filling mehr Knoten ungenutzt bleiben, wird solange kein Back-Filling mehr ausgeführt, bis der erste Job der Warteshlange gestartet wurde.

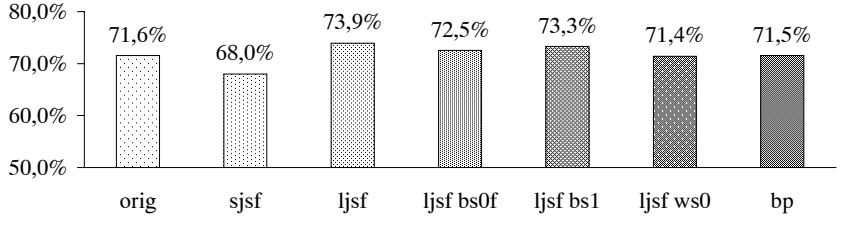


Abbildung 4.53: Cray T3E SJSF, LJSF, BP: Auslastung in der aktiven Zeit l_{act}, ohne Migration

Abbildung 4.53 zeigt die Auslastung des massiv-parallelen Systems Cray T3E während der aktiven Zeit l_{act} durch die simulierten Belegungen der untersuchten Varianten. Dabei wurden keine Jobs migriert. Bei Anwendung der Strategie *Largest Job-Size First* erreicht die Variante *ljsf* ohne Back-Filling eine höhere Auslastung in der aktiven Zeit als die Variante *ljsf ws0* mit *Worst-Size*-Back-Filling ohne Fairneßbetrachtung. Dies ist auf die häufigen Systemunterbrechungen zurückzuführen. Wenn ein Job durch einen Systemausfall beendet wird, wird er in der Simulation als nicht gestartet betrachtet und erneut in die Warteschlange eingeordnet. Die Variante *ljsf ws0* startet nun Jobs mit niedriger Knotenzahl auf freien Knoten, die von den Jobs mit hohen Knotenzahlen nicht genutzt werden. Oft haben diese Jobs kurze Laufzeiten, so daß während der Laufzeit eines Jobs mit hoher Knotenzahl

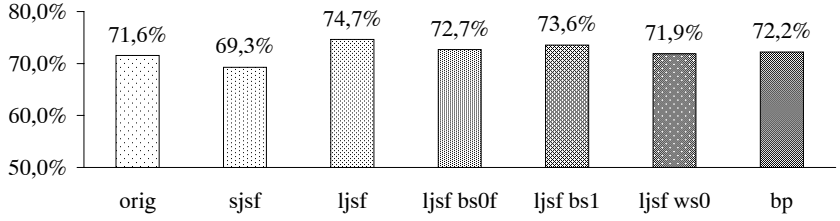


Abbildung 4.54: Cray T3E SJSF, LJSF, BP: Auslastung in der aktiven Zeit l_{act}, mit Migration

mehrere Jobs mit niedriger Knotenzahl ausgeführt werden. Wenn nun eine Systemunterbrechung auftritt, dann wird der Job mit der hohen Knotenzahl und der letzte gestartete Job mit niedriger Knotenzahl beendet. Die anderen bereits beendeten Jobs mit niedriger Knotenzahl führen nun dazu, daß die Zeit, in der sie ausgeführt wurden, als aktive Batch-Betriebszeit betrachtet wird. Da der Job mit hoher Knotenzahl während dieser Zeit zu keiner Last auf dem System führt, wird die Auslastung nur durch die Jobs bestimmt, die vom Back-Filling gestartet wurden. Durch Job-Migration wird die Auslastung in der aktiven Zeit $l_{act,bat}$ noch leicht gesteigert (Abbildung 4.54). Auch hier wird wiederum deutlich, daß die Varianten ohne Back-Filling (*sjsf* und *ljsf*) eine größere Verbesserung erzielen als die Varianten mit Back-Filling, wie in Abschnitt 4.3.1 bereits beschrieben wurde.

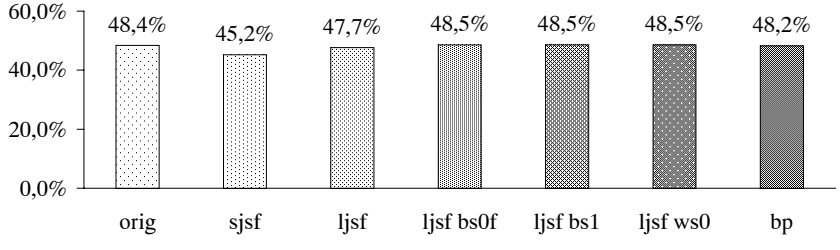


Abbildung 4.55: Cray T3E SJSF, LJSF, BP: Gesamtauslastung l_{ges}, ohne Migration

Die Variante *ljsf* führt gegenüber der Variante *ljsf ws0* trotz der höheren Auslastung in der aktiven Batch-Betriebszeit zu einer schlechteren Gesamtauslastung l_{ges} (Abbildung 4.55), wenn keine Job-Migration durchgeführt werden. Die Ursache für diesen Unterschied sind die Jobs, die in der aktiven Batch-Betriebszeit für die schlechtere Auslastung durch die *ljsf ws0* Variante führen. Diese Jobs müssen von der *ljsf* Variante später ausgeführt werden. Die Ausführungszeit der *ljsf* Variante ist ca. 1 Tag länger als die der *ljsf ws0*. Wenn Jobs migriert werden führen alle Varianten der Strategie *Largest Jobs-Size First* und die Strategie *Best Package* zu

einer Gesamtauslastung von 48,5%. Die Strategie *Smallest Job-Size First* erreicht eine Gesamtauslastung von 46,7% und liegt damit trotz der Job-Migration deutlich unter dem Wert der Original-Auslastung.

Variante	Migrationen	migrierte Jobs	migrierte Knoten
sjsf	141	597	38228
ljsf	88	610	20920
ljsf ws0	122	522	26886
ljsf bs0f	69	343	15396
ljsf bs1	87	539	22050
bp	74	326	15576

Tabelle 4.7: Cray T3E SJSF, LJSF, BP: Migrationen

Die in Tabelle 4.7 aufgelisteten Werte der Job-Migration zeigen, daß die Job-Scheduling-Strategie *Shortest Job-Size First* zu einem deutlich höheren Aufwand führt als die anderen Varianten. Zu der wesentlich höheren Zahl der Migrationen und migrierenden Jobs kommt es durch die Sortierung der Warteschlange nach steigender Knotenzahl der Jobs. Die Jobs, deren Ausführung auf dem massiv-parallelen System beendet wird, geben häufig nicht genügend Knoten für den nächsten zu startenden Job frei. Die Variante *ljsf ws0* führt ebenfalls zu deutlich mehr Migrationen als die anderen Varianten der Strategie *Largest Job-Size First*, da diese Variante durch Back-Filling Jobs mit geringer Knotenzahl bevorzugt. Nach der Fertigstellung der Jobs mit niedriger Knotenzahl bleiben unzusammenhängende Bereiche auf dem massiv-parallelen System frei.

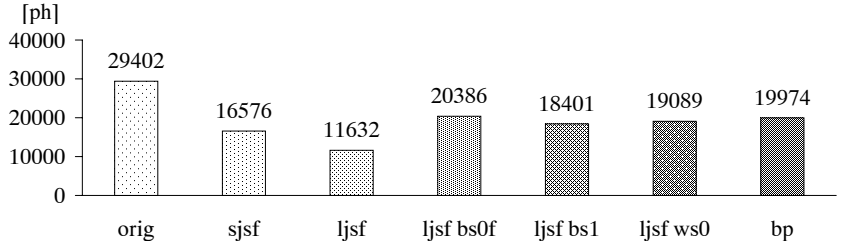


Abbildung 4.56: Cray T3E SJSF, LJSF, BP: Last im gemischten Betrieb L_{mix}, ohne Migration

Die Lasten im gemischten (Abbildung 4.56) und im interaktiven (Abbildung 4.57) Betrieb sind durch die Belegung mit der Strategie *Lagest Job-Size First* ohne Back-Filling am niedrigsten. Alle anderen Varianten führen zu höheren Lasten in den beiden Betriebsarten. Diese sind jedoch deutlich niedriger als die Lasten, die die Originalbelegung in diesen Betriebsarten verursacht. Die in den Abbildungen 4.56 und 4.57 dargestellten Werte unterscheiden sich nur geringfügig von den Werten,

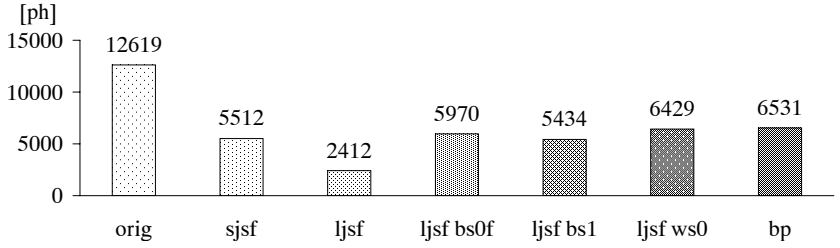


Abbildung 4.57: Cray T3E SJSF, LJSF, BP: Last im interaktiven Betrieb L_{ia}, ohne Migration

die erreicht werden, wenn in der Simulation der Belegung Job-Migration zugelassen werden.

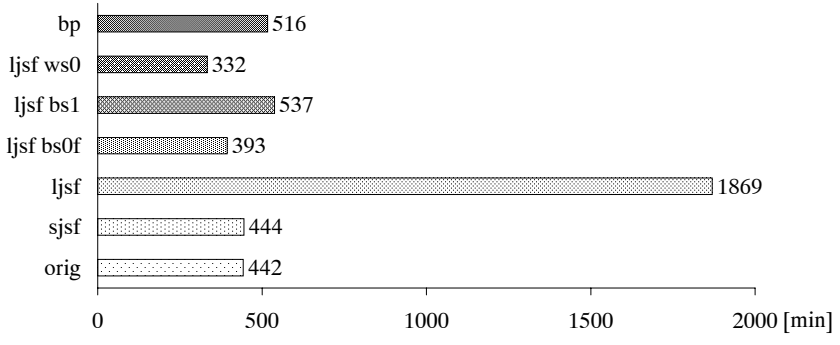


Abbildung 4.58: Cray T3E SJSF, LJSF, BP: mittlere Wartezeit aller Jobs, ohne Migration

Die mittlere Wartezeit aller Jobs ohne Job-Migration (Abblidung 4.58) ist nur bei der Belegung des Systems Cray T3E durch die Varianten *ljsf bs0f* und *ljsf ws0* kürzer als in der Originalbelegung. Der Wert der Variante *ljsf* ist etwa viermal so groß wie der Vergleichswert. Durch das Migrieren von Jobs (Abbildung 4.59) werden die mittleren Wartezeiten der Varianten *bp* und *sjsf* kürzer als in der Originalbelegung.

In der Klasse der Jobs mit einer Knotenzahl von maximal 32 ($C_{\leq 32p}$) liefern die meisten der Varianten der Strategie *Largest Job-Size First* und die Strategie *Best Package* schlechtere Werte für die mittlere Wartezeit als die Originalbelegung. Die Job-Scheduling-Strategie *Smallest Job-Size First* und die Variante *ljsf ws0* hingegen führen zu einer wesentlich kürzeren mittleren Wartezeit in dieser Klasse. In Abbildung 4.60 sind die mittleren Wartezeiten in der Klasse $C_{\leq 32p}$ abgebildet. Diese Werte wurden unter der Verwendung von Job-Migration erreicht. Ohne Migration

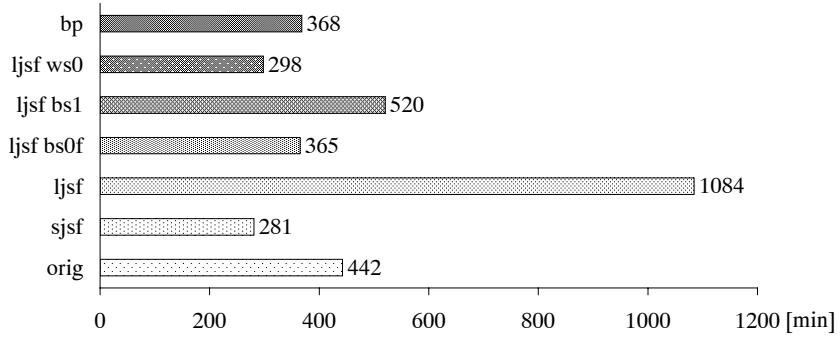


Abbildung 4.59: Cray T3E SJSF, LJSF, BP: mittlere Wartezeit aller Jobs, mit Migration

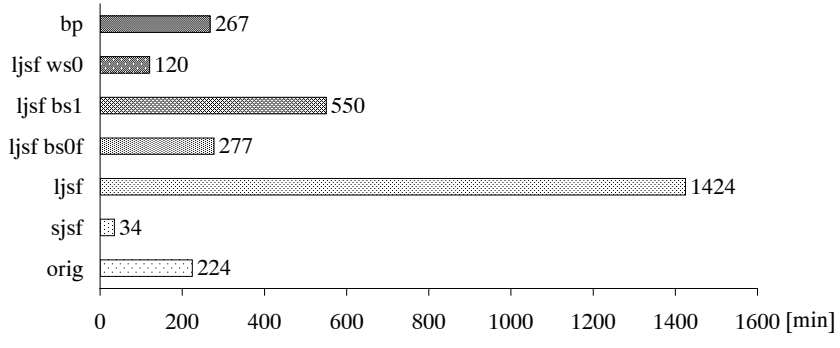


Abbildung 4.60: Cray T3E SJSF, LJSF, BP: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$, mit Migration

sind die Werte für die meisten Varianten geringfügig schlechter, für die Strategie *Best Package* ist der Wert jedoch deutlich schlechter (513 Minuten ohne Migration statt 267 Minuten mit Migration). In der Klasse der Jobs mit maximal 10 Minuten Laufzeit ($C_{\leq 10min}$) erreicht keine der hier betrachteten Varianten den Wert der Originalbelegung für die mittlere Wartezeit (Abbildung 4.61).

Es folgt nun eine Betrachtung der Quantile der Wartezeiten für den Monat Mai 1997. Dieser Monat wurde ausgewählt, weil im Mai deutlich weniger Systemunterbrechungen auftraten als in den anderen Monaten. Neben den bisher in diesem Abschnitt untersuchten Varianten wird zusätzlich noch die Job-Scheduling-Strategie *First Come First Serve* mit *Best Combination* Back-Filling *fcfs bc* betrachtet. Diese Variante führt bei einer geringfügig schlechteren Auslastung als die Varianten der Strategie *Largest Job-Size First* und der Strategie *Best Package* zu sehr niedrigen Wartezei-

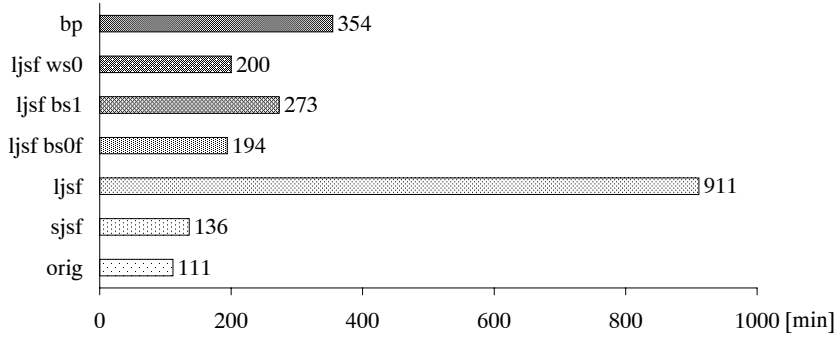


Abbildung 4.61: Cray T3E SJSF, LJSF, BP: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$, mit Migration

ten. In der Untersuchung der Quantile der Wartezeiten werden nur die Ergebnisse betrachtet, die bei der Simulation unter Verwendung von Job-Migration erzielt wurden. Die Werte der Quantile werden in Stunden angegeben.

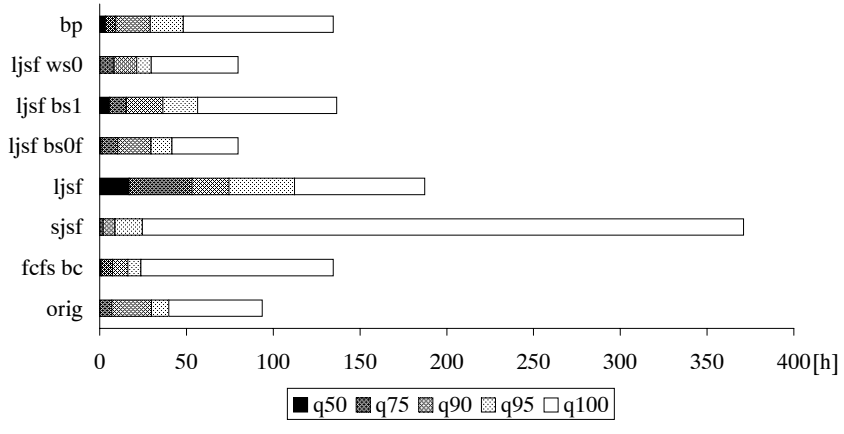


Abbildung 4.62: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten aller Jobs Mai 1997, mit Migration

Die Betrachtung der Quantile der Wartezeiten aller Jobs (Abbildung 4.62 und Tabelle 4.8) zeigt, daß die Varianten *ljsf bs0f* und *ljsf ws0* sehr gute Werte liefern. Diese Werte liegen für alle Quantile nahe bei den Werten der Originalbelegung. Die beiden Varianten *fcfs bc* und *sjsf*, die zu einer wesentlich kürzeren mittleren Wartezeit aller Jobs führten, haben ein wesentlich kleineres Q_{95}-Quantil als die Originalbelegung. Die maximale Wartezeit dieser beiden Varianten ist allerdings deutlich länger als die

Quantile der Wartezeiten in der Klasse C_{all} [h]					
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	0	7	30	40	94
fcfs bc	1	7	16	24	135
sjsf	0	2	9	25	371
ljsf	17	53	74	112	187
ljsf bs0f	1	10	30	42	80
ljsf bs1	6	15	36	56	137
ljsf ws0	0	8	21	30	80
bp	3	9	29	48	135

Tabelle 4.8: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten aller Jobs Mai 1997, mit Migration

der Originalbelegung.

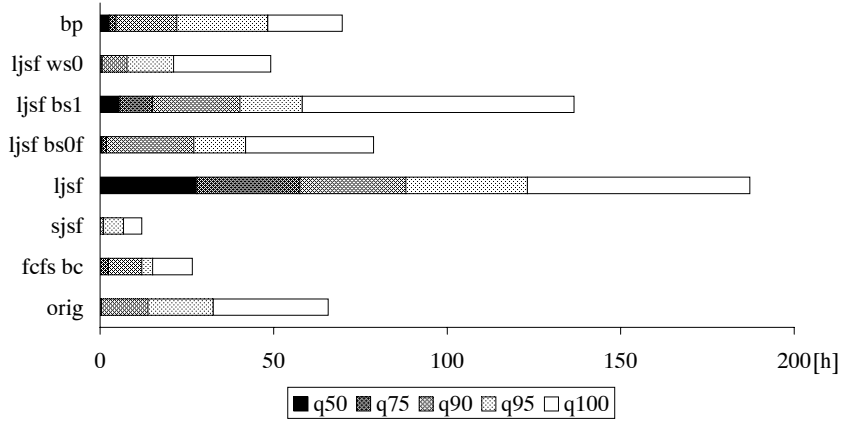


Abbildung 4.63: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 32p}$ Mai 1997, mit Migration

In der Klasse der Jobs mit maximal 32 Knoten ergibt sich für die Varianten *fcfs bc*, *sjsf* und *ljsf ws0* eine sehr gute Verteilung der Wartezeiten. Die in Abbildung 4.63 und Tabelle 4.9 dargestellten Werte der Quantile für diese Varianten sind deutlich besser als die der Originalbelegung. Der Wert des Q_{90}-Quantils der Wartezeiten für die Variante *ljsf bs0f* bedeutet, daß 10% der Jobs frühestens nach 27 Stunden ausgeführt werden. Für einen Job, der beispielsweise Nachmittags an das *NQS* übermittelt wird, bedeutet das, daß er erst am Abend des nächsten Tages gestartet wird.

Quantile der Wartezeiten in der Klasse $C_{\leq 32p}$ [h]					
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	0	0	14	33	66
fcfs bc	0	2	12	15	27
sjsf	0	0	1	7	12
ljsf	28	58	88	123	187
ljsf bs0f	1	2	27	42	79
ljsf bs1	6	15	40	58	137
ljsf ws0	0	1	8	21	49
bp	3	5	22	48	70

Tabelle 4.9: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 32p}$ Mai 1997, mit Migration

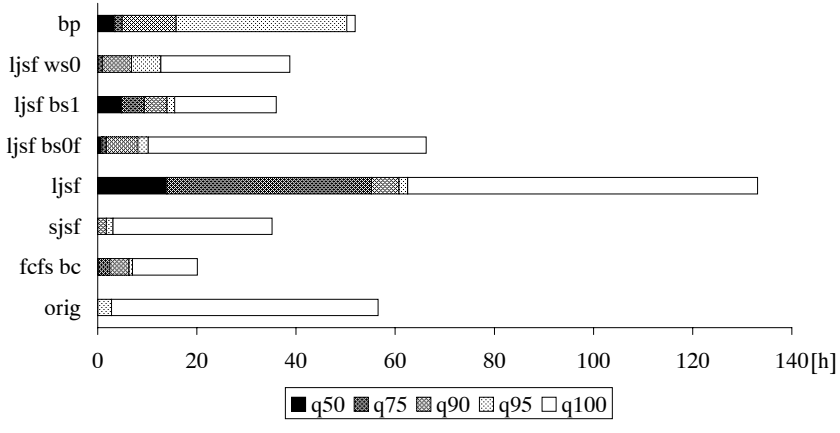


Abbildung 4.64: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 10min}$ Mai 1997, mit Migration

Die maximale Wartezeit in der Klasse der Jobs mit maximal 10 Minuten Laufzeit wird durch die Varianten *fcfs bc*, *sjsf* und *ljsf ws0* im Vergleich zur Originalbelegung verkürzt (Abbildung 4.64 und Tabelle 4.10). Die Verteilung der Wartezeiten in dieser Klasse kann jedoch nur für die Strategie *Smallest Job-Size First* als gut bezeichnet werden. Die Variante *ljsf bs0f* der Strategie *Largest Job-Size First* führt in dieser Klasse zu deutlich schlechteren Ergebnissen als die Originalbelegung. Während in der Originalbelegung 90% aller Jobs mit weniger als 10 Minuten Laufzeit sofort gestartet wurden, werden in der simulierten Belegung durch diese Variante 50% dieser Jobs frühestens nach einer Stunde gestartet.

Quantile der Wartezeiten in der Klasse $C_{\leq 10min}$ [h]					
	Q_{50}	Q_{75}	Q_{90}	Q_{95}	$Max.$
orig	0	0	0	3	57
fcfs bc	0	2	6	7	20
sjsf	0	0	2	3	35
ljsf	14	55	61	63	133
ljsf bs0f	1	2	8	10	66
ljsf bs1	5	9	14	16	36
ljsf ws0	0	1	7	13	39
bp	3	5	16	50	52

Tabelle 4.10: Cray T3E FCFS, SJSF, LJSF, BP: Quantile der Wartezeiten der Jobs in der Klasse $C_{\leq 10min}$ Mai 1997, mit Migration

Die Strategien *Smallest Job-Size First*, *Largest Job-Size First* und *Best Package* lassen sich folgendermaßen zusammenfassen:

Die Job-Scheduling-Strategie *Smallest Job-Size First* führt zu einer relativ schlechten Auslastung des massiv-parallelen Systems. Für 5% aller Jobs wird die Wartezeit deutlich länger als in der Originalbelegung. In den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ werden allerdings wesentlich bessere Wartezeiten erzielt. Die Strategie *Largest Job-Size First* lastet das massiv-parallele System Cray T3E in allen Varianten sehr gut aus. Die Betrachtung der Wartezeiten aller Jobs liefert ebenfalls gute Ergebnisse. Die Jobs der Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ werden allerdings im Vergleich zur Originalbelegung stark verspätet ausgeführt. Die einzige Ausnahme bildet hierbei die Variante *ljsf ws0*, die zumindest in der Klasse $C_{\leq 32p}$ zu sehr kurzen Wartezeiten führt. In der Klasse $C_{\leq 10min}$ führt aber auch diese Variante zu längeren Wartezeiten. Die Strategie *Best Package* lastet das massiv-parallele System ebenfalls gut aus. Die Wartezeiten aller Jobs sowie der beiden betrachteten Job-Klasssen sind jedoch länger als in der Originalbelegung.

4.3.4 Smallest Cumulative Demand First, Largest Cumulative Demand First

In diesem Abschnitt werden die Ergebnisse der Strategien *Smallest Cumulative Demand First* (*SCDF*) und *Largest Cumulative Demand First* (*LCDF*) für die Belegung des massiv-parallelen Systems Cray T3E präsentiert. Folgende Varianten werden betrachtet:

- Smallest Cumulative Demand First mit Best-Size-Back-Filling ohne Fairneßbetrachtung (*scdf bs0*)
- Largest Cumulative Demand First mit First-Come-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lcdf fc1*)

- Largest Cumulative Demand First mit Best-Demand-Back-Filling und Fairneßbetrachtung anhand der Job-Laufzeiten (*lcdf bd1*)

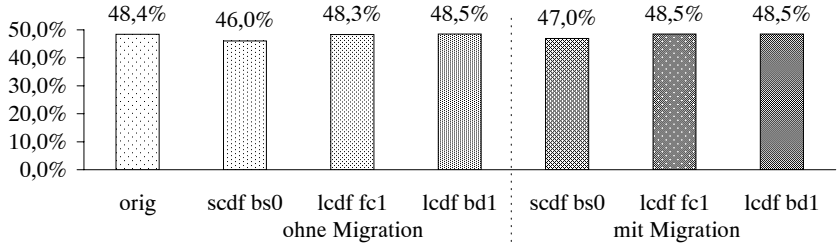


Abbildung 4.65: Cray T3E SCDF, LCDF: Gesamtauslastungt l_{ges}

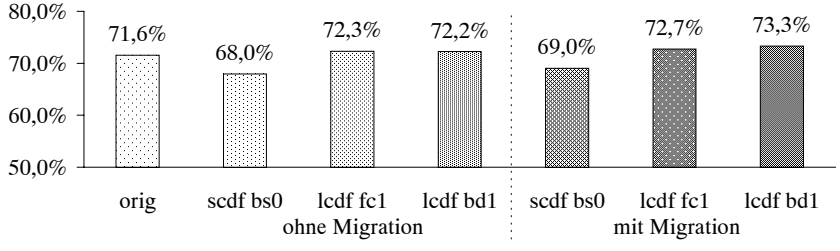


Abbildung 4.66: Cray T3E SCDF, LCDF: Auslastung in der aktiven Zeit l_{act}

Abbildung 4.65 zeigt, daß die Auslastung in der aktiven Zeit l_{act} durch die Varianten *lcdf fc1* und *lcdf bd1* der Strategie *Largest Cumulative Demand First* über dem Wert der Originalbelegung liegt. Die Belegung durch die Strategie *Shortest Cumulative Demand First* führt für die beste Variante (*scdf bs0*) sowohl mit als auch ohne Migration zu einer deutlich schlechteren Auslastung als die Originalbelegung.

Die Lasten im gemischten und im interaktiven Betrieb sind durch die Belegung mit allen Varianten der *Smallest Cumulative Demand First* und der Strategie *Largest Cumulative Demand First* deutlich niedriger als durch die Originalbelegung, so daß diese Werte keinen Nachteil der Strategien darstellen.

Die mittlere Wartezeit aller Jobs (Abbildung 4.67) ist für die Variante *scdf bs0* der Strategie *Smallest Cumulative Demand First* deutlich kürzer als der Vergleichswert der Originalbelegung. Die maximale Wartezeit liegt mit 762 Stunden (ohne Migration) jedoch deutlich höher als die maximale Wartezeit in der Originalbelegung (152 Stunden). Für die Strategie *Largest Cumluative Demand First* erreichen die betrachteten Varianten eine ähnlich kurze mittlere Wartezeit wie die Originalbelegung, wenn Job-Migration zugelassen werden.

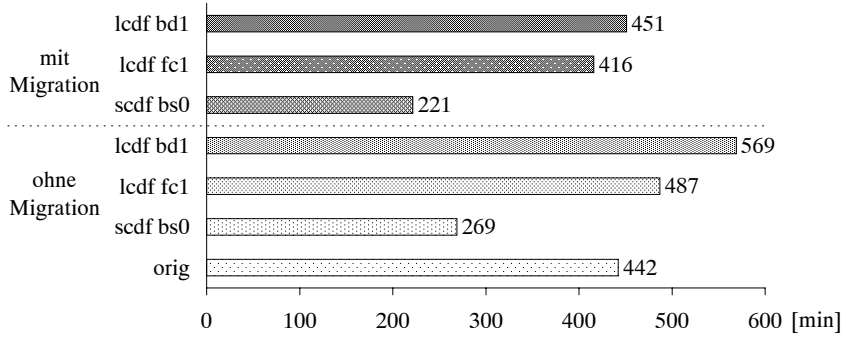


Abbildung 4.67: Cray T3E SCDF, LCDF: mittlere Wartezeit aller Jobs

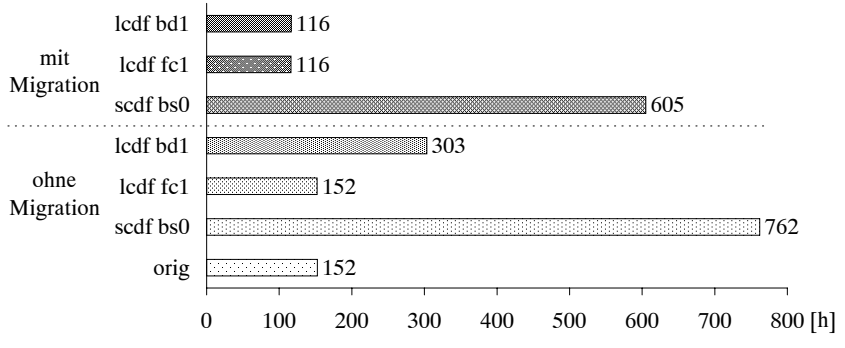


Abbildung 4.68: Cray T3E SCDF, LCDF: maximale Wartezeit aller Jobs

In der Klasse der Jobs mit maximal 32 Knoten erreicht die *scdf bs0* Variante sowohl ohne als auch mit Job-Migration eine wesentlich kürzere mittlere Wartezeit als die Originalbelegung (Abbildung 4.70). Die maximale Wartezeit in dieser Klasse beträgt für diese Variante 34 Stunden. Dies ist etwa die Hälfte der maximalen Wartezeit, die die Originalbelegung in dieser Klasse erreicht (66 Stunden). Für die Strategie *Largest Cumulative Demand First* erreicht lediglich die Varainte *lcdf fc1* einen akzeptablen Wert für die mittlere Wartezeit in dieser Klasse, wenn Job-Migration zugelassen werden. Die maximale Wartezeit in dieser Klasse ist für diese Variante mit 94 Stunden deutlich länger als in der Originalbelegung.

Die Strategien *Largest Cumulative Demand First* und *Smallest Cumulative Demand First* liefern für die mittlere Wartezeit in der Klasse der Jobs mit maximal 10 Minuten Laufzeit ein ähnliches Bild wie in der Klasse der Jobs mit maximal 32 Knoten. Auch hier führt die Strategie *Smallest Cumulative Demand First* zu sehr kurzen Wartezeiten. Alle Varianten der Strategie *Largest Cumulative Demand First* führen

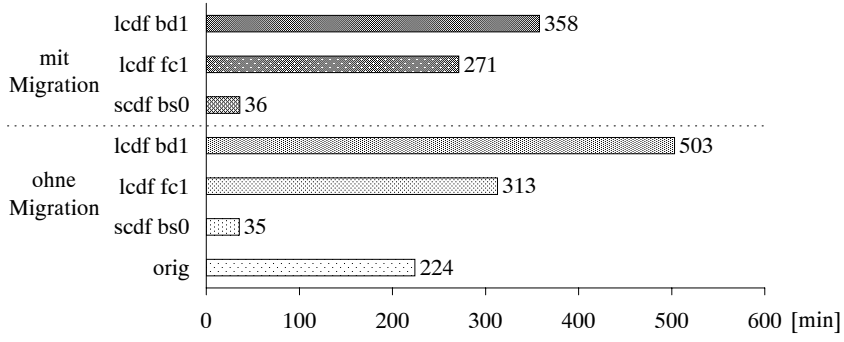


Abbildung 4.69: Cray T3E SCDF, LCDF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 32p}$

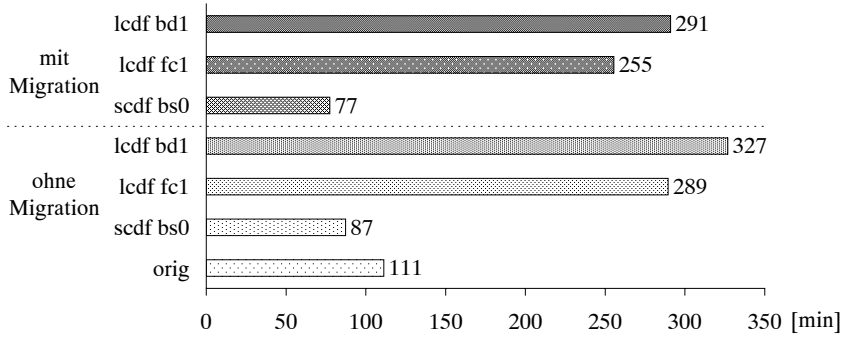


Abbildung 4.70: Cray T3E SCDF, LCDF: mittlere Wartezeit der Jobs in der Klasse $C_{\leq 10min}$

zu wesentlich längeren Wartezeiten der Jobs in dieser Klasse.

Zusammengefaßt ergeben die beiden Strategien *Smallest Cumulative Demand First* und *Largest Cumulative Demand First* folgendes Bild:
Die besten Back-Filling-Varianten der Strategie *Smallest Cumulative Demand First* lasten das massiv-parallele System Cray T3E wesentlich schlechter aus als die Originalbelegung. Die Wartezeiten sind für ca. 5% aller Jobs länger als in der Originalbelegung. In den beiden Klassen der Jobs mit geringer Knotenanzahl ($C_{\leq 32p}$) und mit kurzer Laufzeit ($C_{\leq 10min}$) sind die Wartezeiten der Jobs deutlich kürzer als in der Originalbelegung. Die Strategie *Largest Cumulative Demand First* lastet das System besser als die Originalbelegung aus. Die Wartezeiten der Jobs insbesondere in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ sind deutlich länger als in der Originalbelegung.

4.4 Beurteilung der Strategien

Die abschließende Beurteilung der Job-Scheduling-Strategien stützt sich auf die Ergebnisse, die in der Simulation der Belegung der beiden massiv-parallelen Systeme Intel Paragon und Cray T3E erzielt wurden und in diesem Kapitel bisher präsentiert wurden. Da die Gesamtauslastung der Systeme von der Übermittlungszeit des letzten Jobs in jedem Monat abhängig ist, wurde zusätzlich die Belegung der beiden Systeme simuliert unter der Annahme, daß alle Jobs eines Monats bereits zu Beginn der Simulation zur Verfügung stehen. Alle anderen Randbedingungen wie zum Beispiel Batch-Betriebszeit und Ausfallzeiten der Systeme wurden in dieser Simulation weiterhin berücksichtigt, Job-Migration wurden auf dem System Cray T3E nicht zugelassen. Die dabei gewonnenen Werte lassen sich nicht mit den Werten der Originalbelegung vergleichen, aber sie können für einen Vergleich der Strategien untereinander genutzt werden. Neben der Gesamtauslastung wird der Anteil der freien Batch-Betriebszeit an der gesamten Batch-Betriebszeit betrachtet. Die Ergebnisse der simulierten Belegung des massiv-parallelen Systems Cray T3E werden hier zwar präsentiert, aber sie beeinflußen die Beurteilung der Strategien nicht so stark wie die Ergbnisse für Intel Paragon, da der betrachtete Zeitraum wesentlich kürzer ist und die vielen Systemunterbrechungen die Ergebnisse verfälschen. Dies gilt auch für die in den vorherigen Abschnitten präsentierten Ergebnisse.

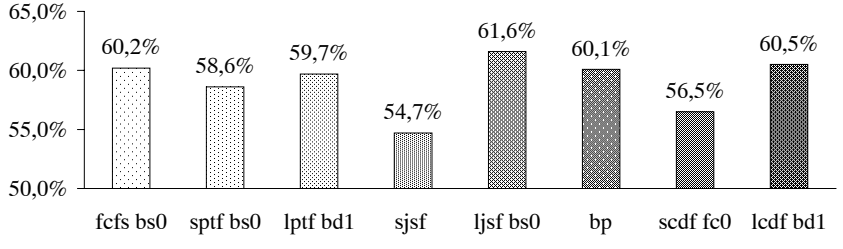


Abbildung 4.71: Intel Paragon: Gesamtauslastung l_{ges} ohne Berücksichtigung der Übermittlungszeiten

Die Abbildungen 4.71 und 4.72 zeigen die Gesamtauslstungen l_{ges} für die besten Back-Filling-Varianten der einzelnen Job-Scheduling-Strategien auf den beiden massiv-parallelen Systemen. Für das System Intel Paragon benötigt die Job-Scheduling-Strategie *Largest Job-Size First* mit *Best-Size*-Back-Filling ohne Fairneßbetrachtung (*ljsf bs0*) ca. 493 Tage zur Ausführung aller Jobs. In der Simulation der Belegung mit Berücksichtigung der Übermittlungszeiten benötigte diese Variante ca. 634 Tage. Auf dem System Cray T3E hat die Variante *ljsf bs0* ohne Berücksichtigung der Übermittlungszeiten eine Gesamtlaufzeit von ca. 72 Tagen gegenüber einer Gesamtlaufzeit von ca. 91 Tagen, wenn die Übermittlungszeiten der Jobs berücksichtigt werden. Während auf Intel Paragon der Unterschied zwischen der besten (*ljsf bs0*) und der schlechtesten Variante (*sjsf*) 6,9% beträgt (das entspricht einem

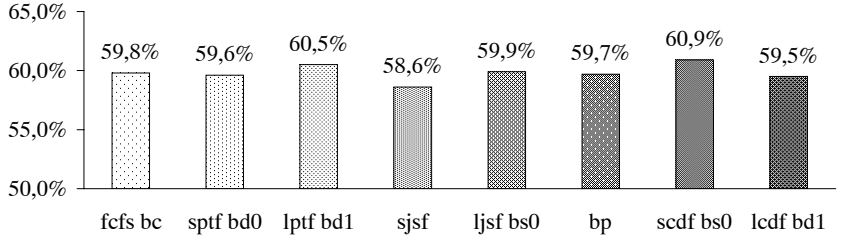


Abbildung 4.72: Cray T3E: Gesamtauslastung l_{ges} ohne Berücksichtigung der Übermittlungszeiten

um ca. 62 Tage längeren Zeitraum), unterscheiden sich die beste (*scdf bs0*) und die schlechteste Variante (*sjsf*) auf dem System Cray T3E nur um 2,3% (2 Tage und 17 Stunden).

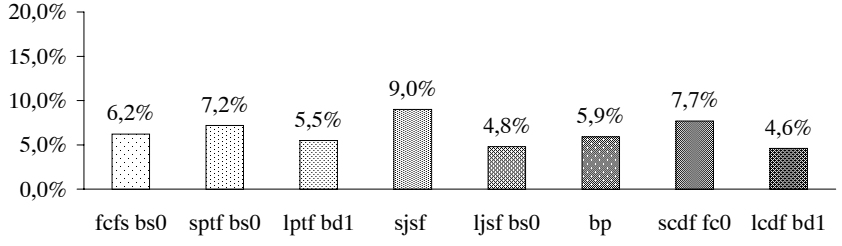


Abbildung 4.73: Intel Paragon: Anteil der freien Zeit an der gesamten Batch-Betriebszeit $t_{idle,bat}$ ohne Berücksichtigung der Übermittlungszeiten

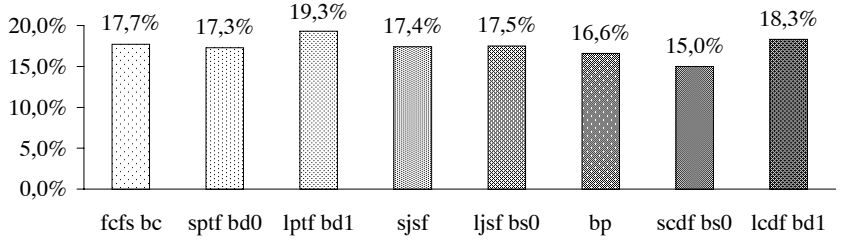


Abbildung 4.74: Cray T3E: Anteil der freien Zeit an der gesamten Batch-Betriebszeit $t_{idle,bat}$ ohne Berücksichtigung der Übermittlungszeiten

Die Strategie *Shortest Cumulative Demand First* lastet das System Cray T3E besser aus als alle anderen Strategien. Das System Intel Paragon hingegen wird von

dieser Strategie schlechter ausgelastet als von den meisten anderen Strategien. Der Grund dafür sind die häufigen Systemunterbrechungen, die eine bessere Auslastung des Systems Cray T3E durch die anderen Strategien verhindern. Dies wird auch aus Abbildung 4.74 ersichtlich. Die Variante *scdf bs0* hat den geringsten Anteil der freien Zeit an der gesamten Batch-Betriebszeit. Die freie Batch-Betriebszeit umfaßt in dieser Simulation nur Ausfallzeiten des Systems und die Zeit vor einer Systemunterbrechung, da Jobs, die durch eine Systemunterbrechung gestoppt werden, als nicht gestartet betrachtet werden. Durch eine leere Warteschlange kann keine freie Batch-Betriebszeit entstehen, da in diesem Fall alle Jobs abgearbeitet wurden und die Simulation somit abgeschlossen ist. Der Unterschied des freien Anteils an der Batch-Betriebszeit zwischen den Systemen Intel Paragon und Cray T3E ist ebenfalls auf die Systemausfälle zurückzuführen.

Es folgt nun die abschließende Beurteilung der Strategien.

Smallest Job-Size First

Die Strategie *Smallest Job-Size First* lastet die beiden Systeme Intel Paragon und Cray T3E am schlechtesten von allen hier betrachteten Job-Scheduling-Strategien aus. Auf Intel Paragon erreicht sie eine Gesamtauslastung von 43,4%. Im Vergleich zur Originalbelegung sind das 4,6% weniger. Das entspricht einem um 67 Tage längeren Zeitraum. Auf Cray T3E beträgt die Gesamtauslastung durch diese Strategie 45,2% gegenüber 48,4% in der Originalbelegung. Deshalb wird diese Strategie als die schlechteste aller hier betrachteten Strategien eingestuft. Die Verteilung der Wartezeiten aller Jobs sind auf dem System Intel Paragon in den meisten Monaten durch diese Strategie wesentlich schlechter als durch die Originalbelegung. Auf dem System Cray T3E ist die Verteilung der Wartezeiten aller Jobs nahezu gleichwertig zur Originalbelegung. Die Wartezeiten der Jobs der Klasse mit maximal 32 Knoten ($C_{\leq 32p}$) sind auf beiden Systemen kürzer als durch alle anderen Strategien. In der Klasse der Jobs mit maximal 10 Minuten Laufzeit ($C_{\leq 10min}$) sind die Wartezeiten auf dem System Intel Paragon deutlich länger als in der Originalbelegung. Auf Cray T3E werden 5 bis 10% der Jobs dieser Klasse deutlich später ausgeführt als durch die Originalbelegung.

Smallest Cumulative Demand First

Die Auslastung der beiden massiv-parallelen Systeme Intel Paragon und Cray T3E durch die Strategie *Smallest Cumulative Demand First* mit Back-Filling ist etwas besser als durch die Strategie *Smallest Job-Size First* (Intel Paragon 44,7%, Cray T3E 46,0%). Dadurch wird sie als zweitschlechteste Strategie klassifziert. Ein weiterer Nachteil dieser Strategie ist die eingeschränkte Realisierbarkeit, da sie die tatsächliche Laufzeit der Jobs zur Sortierung der Warteschlange benötigt. Die Verteilung der Wartezeiten aller Jobs durch die Strategie ist auf beiden Systeme deutlich schlechter als durch die Originalbelegung. Die Wartezeiten der Jobs mit maximal 32 Knoten und mit maximal 10 Minuten Laufzeit sind sowohl auf dem System Intel Paragon als auch auf Cray T3E sehr kurz.

Shortest Processing Time First

Die Gesamtauslastung durch die Strategie *Shortest Processing Time First* mit Back-Filling beträgt auf Intel Paragon 45,7% (48,0% in der Originalbelegung) und auf Cray T3E 47,1% (48,4% in der Originalbelegung). Die Auslastung ist also besser als durch die Strategie *Smallest Cumulative Demand First*. Aus diesem Grund wird die Job-Scheduling-Strategie *Shortest Processing Time First* vor den beiden Strategien *Smallest Job-Size First* und *Smallest Cumulative Demand First* eingestuft. Diese Strategie nutzt die tatsächliche Job-Laufzeit zur Sortierung der Warteschlange, wodurch sie nur eingeschränkt realisierbar ist. Etwa 5% aller Jobs werden auf beiden Systemen deutlich später ausgeführt als in den Originalbelegungen. Die Wartezeiten der Jobs in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ sind auf beiden System kürzer als durch die Originalbelegungen.

Largest Processing Time First

Die Strategie *Largest Processing Time First* lastet die beiden massiv-parallelen Systeme Intel Paragon und Cray T3E besser aus als die drei bisher beurteilten Job-Scheduling-Strategien. Auf Intel Paragon erreicht sie 46,6%. Das sind 1,4% weniger als in der Originalbelegung. Auf Cray T3E liegt die Gesamtauslastung mit 47,2% um 1,2% unter dem erreichten Wert der Originalbelegung. Diese Strategie hat ebenfalls den Nachteil der eingeschränkten Realisierbarkeit, da sie die Warteschlange anhand der tatsächliche Laufzeit der Jobs sortiert. Die Verteilung der Wartezeiten aller Jobs ist auf beiden Systemen ähnlich gut wie durch die Originalbelegung. Das gleiche gilt für die Klasse der Jobs mit 32 Knoten $C_{\leq 32p}$. Die Jobs der Klasse mit maximal 10 Minuten Laufzeit $C_{\leq 10min}$ warten deutlich länger auf ihre Ausführung als in der Originalbelegung.

First Come First Serve

Die Strategie *First Come First Serve* führt auf den betrachteten Systemen zu einer geringfügig schlechteren Gesamtauslastung als die Originalbelegung. Auf dem System Intel Paragon erreicht sie mit 47,5% eine um 0,5% schlechtere Auslastung. Die Gesamtlaufzeit ist um ca. 6 Tage länger. Auf Cray T3E ist die Gesamtlaufzeit ca. 1 Tag länger als durch die Originalbelegung. Dort erreicht die Strategie *First Come First Serve* eine um 0,7% schlechtere Gesamtauslastung von 47,7%. Auf Intel Paragon liefert diese Strategie ähnlich kurze Wartezeiten aller Jobs und der Jobs mit maximal 32 Knoten ($C_{\leq 32p}$) wie die Originalbelegung. In der Klasse $C_{\leq 10min}$ sind die Wartezeiten länger als durch die Originalbelegung. Auf dem System Cray T3E sind die Verteilungen der Wartezeiten in allen Klassen sehr gut.

Largest Cumulative Demand First

Die besten drei Strategien, *Largest Cumulative Demand First*, *Best Package* und *Largest Job-Size First* erreichen die Gesamtauslastung der Originalbelegung sowohl auf Intel Paragon (48,0%) als auch auf Cray T3E (48,4%). Die Gesamtauslastung ohne Berücksichtigung der Übermittlungszeiten ist durch die Strategie *Largest Cumulative Demand First* auf dem System Intel Paragon geringfügig besser als für die Strategie *Best Package*. Die Strategie *Largest Cumulative Demand First* wird

als schlechteste dieser drei Strategien eingestuft, da sie die massiv-parallelen Systeme in der aktiven Zeit und in der aktiven Batch-Betriebszeit schlechter auslastet als die beiden anderen Strategien. Ein weiterer Nachteil dieser Strategie ist die Verwendung der tatsächlichen Laufzeit der Jobs zum Sortieren der Warteschlange, da sie dadurch nur eingeschränkt realisierbar ist. Die Verteilung der Wartezeiten aller Jobs ist durch die Strategie *Largest Cumulative Demand First* schlechter als durch die Strategien *Best Package* und *Largest Job-Size First*, liegt jedoch auf dem gleichen Niveau wie die Verteilung der Wartezeiten durch die Originlabelegung. Die Jobs der Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ haben in der Belegung durch die *Largest Cumulative Demand First* längere Wartezeiten als in der Originalbelegung.

Best Package

Die Strategie *Best Package* wird als zweitbeste Strategie eingestuft. Die Gesamtauslastung des Systems Intel Paragon ist ohne Berücksichtigung der Übermittlungszeiten zwar geringfügig schlechter als durch die Strategie *Largest Cumulative Demand First*, aber die Strategie *Best Package* lastet beide massiv-parallelen Systeme in der aktiven Zeit und in der aktiven Batch-Betriebszeit besser aus als die Strategie *Largest Cumulative Demand First*. Verglichen mit der Strategie *Largest Job-Size First* ist die Gesamtauslastung des Systems Intel Paragon ohne Berücksichtigung der Übermittlungszeiten durch die Strategie *Best Package* deutlich schlechter. Die Verteilung der Wartezeiten aller Jobs ist auf den Systemen Intel Paragon und Cray T3E bei Anwendung der Strategie *Best Package* schlechter als bei Anwendung der Strategie *Largest Job-Size First*, aber besser als durch die Originalbelegung. Die Jobs in der Klasse mit maximal 32 Knoten $C_{\leq 32p}$ warten auf beiden massiv-parallelen Systemen etwa genauso lang wie in der Originalbelegung auf ihre Ausführung. In der Klasse der Jobs mit maximal 10 Minuten Laufzeit $C_{\leq 10min}$ werden die Wartezeiten der Jobs durch die Belegung der Systeme mit der Strategie *Best Package* länger als durch die Originalbelegung. Ein weiterer Nachteil der Job-Scheduling-Strategie *Best Package* ist der Overhead durch die Strategie selbst (siehe Abschnitt 3.3.3).

Largest Job-Size First

Die Strategie *Largest Job-Size First* erreicht unter den drei Strategien *Largest Cumulative Demand First*, *Best Package* und *Largest Job-Size First*, die die Gesamtauslastung der Originalbelegung erreichen, die besten Werte für alle weiteren Auslastungen, wie zum Beispiel die Auslastung in der aktiven Zeit, die Auslastung in der aktiven Batch-Betriebszeit und die Gesamtauslastung in der Simulation ohne Berücksichtigung der Übermittlungszeiten. Aus diesen Gründen wird die Job-Scheduling Strategie *Largest Job-Size First* als die beste der hier betrachteten Strategien klassifiziert. Die Verteilung der Wartezeiten für alle Jobs ist besser als durch die Originalbelegung. In den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ werden durch einige Back-Filling-Varianten der Strategie *Largest Job-Size First* Wartezeiten erreicht, die genauso kurz wie die der Originalbelegung sind. Dadurch wird jedoch die Auslastung geringfügig verschlechtert. Die Back-Filling-Varianten mit der höchsten Auslastung lasten die Systeme Intel Paragon und Cray T3E geringfügig besser aus

als die Originalbelegung. Die Wartezeiten in den Klassen $C_{\leq 32p}$ und $C_{\leq 10min}$ sind in der Belegung der Systeme durch diese Varianten jedoch länger als in der Originalbelegung.

Die Job-Scheduling-Strategie *Best Package* und Back-Filling-Varianten der Strategien *Largest Cumulative Demand First* und *Largest Job-Size First* führen zu einer ähnlich hohen Gesamtauslastung der massiv-parallelen Systeme Intel Paragon und Cray T3E wie die Originalbelegung der Systeme. Die Zeitplanerstellung der auf den massiv-parallelen Systemen Intel Paragon und Cray T3E eingesetzten Scheduling-Systeme wurde bisher durch manuelles Eingreifen der Systemadministratoren optimiert. Die Strategie *Best Package* und einige Back-Filling-Varianten der Job-Scheduling-Strategie *Largest Job-Size First* nutzen nicht die tatsächliche Laufzeit der Jobs zum sortieren der Warteschlange und könnten somit auf den Systemen implementiert werden und die Systemadministratoren entlasten.

Die Verteilung der Wartezeiten aller Jobs ist ebenfalls ähnlich wie durch die Originalbelegung. In der Klasse $C_{\leq 10min}$ der Jobs mit maximal 10 Minuten Laufzeit sind die Wartezeiten jedoch länger. Auf dem System Cray T3E gibt es für die Jobs dieser Klasse die Express-Queues (siehe Abschnitt 3.2.2). Die Daten des Tools *TREND* für den betrachteten Zeitraum von März 1997 bis Mai 1997 enthalten jedoch keinen Job, der in eine Express-Queues einsortiert wurde. Aus diesem Grund konnten die Jobs der Klasse $C_{\leq 10min}$ in der Simulation nicht bevorzugt werden. Auf dem System Intel Paragon wurden nur die Jobs, die durch die Sonderregelung für Test-Jobs (siehe Abschnitt 3.1.2) gestartet wurden, in eine Warteschlange mit maximal einer Stunde Laufzeit einsortiert. Alle anderen Jobs wurden in Warteschlangen mit einer maximalen Laufzeit von vier Stunden einsortiert. Dadurch gab es auch in der Simulation der Belegung dieses Systems keine Möglichkeit die Jobs mit kurzer Laufzeit zu bevorzugen. Durch die Verwendung der Express-Queues auf dem massiv-parallelen System Cray T3E wird es möglich sein, die Wartezeiten in der Klasse $C_{\leq 10min}$ der Jobs mit maximal 10 Minuten Laufzeit zu verkürzen. Dadurch könnten die Strategien *Best Package* und *Largest Job-Size First* auch in dieser Klasse ähnlich kurze Wartezeiten wie die Originalbelegung erreichen.

5. Zusammenfassung und Ausblick

Der Betrieb eines massiv-parallelen Rechnersystems, das durch mehrere Anwender gemeinsam genutzt wird, stellt an den Job-Scheduler des Systems zwei unterschiedliche Forderungen:

1. Die Auslastung des massiv-parallelen Systems soll möglichst hoch sein, um das System wirtschaftlich zu betreiben.
2. Die Wartezeiten der Jobs, die die Benutzer auf dem massiv-parallelen System ausführen wollen, sollen möglichst kurz sein, damit die Ergebnisse schnell zur Verfügung stehen.

Im Forschungszentrum Jülich werden zwei massiv-parallele Systeme betrieben. Zum einen das System Intel Paragon, das bereits seit mehreren Jahren als Produktionssystem eingesetzt wird, zum anderen das System Cray T3E, das erst seit kurzer Zeit zur Verfügung steht.

Die Zeitplanerstellung des auf dem massiv-parallelen Rechner Intel Paragon eingesetzten Job-Scheduler-Systems wird von den Systemadministratoren manuell optimiert. Auf dem neuen System Cray T3E wird die manuelle Optimierung der Zeitplanerstellung schwieriger, da dieses System mehr Prozessoren (518) als das System Intel Paragon (138) hat und wesentlich mehr Jobs auf diesem System ausgeführt werden. In den betrachteten Zeiträumen wurden durchschnittlich 1280 Jobs pro Monat auf Cray T3E und 683 Jobs pro Monat auf Intel Paragon im Batch-Betrieb ausgeführt.

Für das massiv-parallele System Cray T3E wurde ein politischer Scheduler angekündigt. Mit diesem politischen Scheduler wird es möglich, eine für dieses System entwickelte Job-Scheduling-Strategie zu implementieren. Um die oben aufgestellten Forderungen nach hoher Auslastung des Systems und kurzen Wartezeiten der Jobs zu erfüllen, wurden in der vorliegenden Arbeit verschiedene Job-Scheduling-Strategien untersucht.

Die Belegung der beiden massiv-parallelen Systeme wurde mit einem für diese Untersuchung entwickelten Simulator durchgeführt. Damit die simulierten Belegungen

mit den realen Belegungen der beiden Systeme verglichen werden können, wurden alle Randbedingungen der Systeme berücksichtigt. Dies sind:

- Job-Informationen der Arbeitslasten
- Batch-Betriebszeiten
- zeitliche Einschränkungen für bestimmte Job-Größen (maximale Knotenzahl)
- Überwachung des *NQS*-Limits (maximale Zeit während der ein Benutzer das System allein benutzen darf)
- Sonderregelungen für bestimmte *NQS*-Warteschlangen
- Ausfallzeiten der Systeme

Durch die Veränderung der Job-Scheduling-Strategie und der damit verbunden Bevorzugung bestimmter Job-Gruppen wird sich das Verhalten der Benutzer ebenfalls ändern. Sie werden nach Möglichkeit Jobs der bevorzugten Gruppen übermitteln. Dies konnte natürlich in der Simulation nicht berücksicht werden.

Die Untersuchung ergab, daß drei Strategien (*Largest Cumulative Demand First*, *Best Package* und *Largest Job-Size First*) die Systeme höher auslasten als alle anderen Strategien. Von diesen drei Strategien kann die Strategie *Largest Cumulative Demand First* nur eingeschränkt realisiert werden, weil sie die tatsächliche Laufzeit der Jobs, die erst nach dem Ausführungsende eines Jobs feststeht, zur Job-Auswahl benutzt. Hinzu kommt, daß diese Strategie für viele Jobs zu längeren Wartezeiten führt. Insbesondere die Jobs mit geringer Knotenzahl und kurzer Laufzeit werden durch diese Strategie stark benachteiligt. Unter den beiden verbleibenden Job-Scheduling-Strategien (*Best Package* und *Largest Job-Size First*) wird die Strategie *Largest Job-Size First* favorisiert, da diese Strategie zu kürzeren Wartezeiten als die Strategie *Best Package* führt. Der Overhead durch die Berechnung aller möglichen Kombinationen der wartenden Jobs ist ein weiterer Nachteil der Job-Scheduling-Strategie *Best Package*.

Die Entwicklung einer Job-Scheduling-Strategie für das massiv-parallele System Cray T3E sollte sich auf die Strategie *Largest Job-Size First* konzentrieren. Dabei sollte besonders das Back-Filling für die Klasse der Jobs mit maximal 10 Minuten Ausführungszeit optimiert werden, da die Jobs dieser Klasse geringfügig länger warten als in der realen Belegung des Systems Cray T3E. Dies ist der einzige Nachteil dieser Strategie, der in der Untersuchung deutlich wurde.

A. Abgrenzung der Begriffe

Im Bereich des *Scheduling* werden von verschiedenen Autoren einige Begriffe mit unterschiedlicher Bedeutung verwendet, deshalb wird für diese Arbeit eine Abgrenzung der Begriffe gegeben. Die Begriffe werden nicht neu definiert, sondern sie werden so umschrieben, wie sie in der Literatur am häufigsten verwendet werden. Soweit es das Verständnis nicht erschwert, werden deutsche Begriffe verwendet, viele Begriffe sind jedoch in ihrer deutschen Übersetzung nicht geläufig oder sogar völlig unverständlich, wie z.B. die deutsche Übersetzung von *Thread* (dt. *Faden*). Die in dieser Arbeit verwendeten Begriffe werden zuerst genannt, die entsprechende Übersetzung (*dt.* für Deutsch, *engl.* für Englisch) folgt dahinter in Klammern.

Allokation, Zuteilung (engl. Allocation):
Die Zuteilung der zu einer Partition zusammengefaßten Knoten an einen Job.

Knotextwechsel (engl. Context-Switch):
Der Wechsel zwischen zwei Jobs auf einem Knoten.

Job (dt. Auftrag):
Das ausführbare Programm, das zur Ausführung an den Scheduler übermittelt wird inklusive der zugehörigen Daten, die für die Ausführung benötigt werden.

Fragmentierung (engl. Fragmentation):
Die Zerstückelung des Systems, wobei nicht-nutzbare Bereiche (Fragmente) entstehen.
Es wird zwischen *interner Fragmentierung* und *exterener Fragmentierung* unterschieden. *Interne Fragmentierung* entsteht, wenn ein Programm nicht alle ihm zugewiesenen Knoten während der gesamten Zeit nutzt, *externe Fragmentierung* heißt, daß Knoten keinem Job zugeordnet sind, obwohl noch Jobs auf ihre Ausführung warten.

Knoten (engl. Node):
Die einzelne Recheneinheit, die mit anderen Recheneinheiten kommunizieren kann. Sie besteht meistens aus einem Prozessor, kann aber auch mehrere Prozessoren beinhalten, beispielsweise einen Rechen- und einen Kommunikationsprozessor.

massiv-paralleler Rechner, massiv-paralleles System (engl. Massivly Parallel Processor):
Ein Computer mit einigen 10 bis zu einigen 1000 Knoten, die über ein Verbindungsnetzwerk mit hoher Übertragungsleistung verbunden sind. (Abkürzung MPP)

Mesh, Submesh (dt. Gitter, Untergitter):
Das Verbindungsnetzwerk der Knoten eines MPPs, über das die Knoten miteinander kommunizieren bzw. ein Teil (*Submesh*) davon.
Partition (engl. Partition):
Die Gesamtheit der Knoten, die einem einzelnen Job zugeordnet wird. Die Knoten müssen nicht zusammenhängend sein.
Partitionierung (engl. Partitioning):
Die Aufteilung des Rechnersystems in Partitionen.
Prozeß (engl. Process)
Der Teil eines Jobs, der einem einzelnen Knoten zur Ausführung übergeben wird.
Prozessor (engl. Processor):
Das eigentliche Rechenwerk eines Knoten.
Scheduler (dt. Zeitplaner):
Das System, das die Auswahl eines Jobs oder eines Threads (*Scheduling*), wenn nötig die Partitionierung des MPPs und die Partitions-Zuteilung (*Allocation*) vornimmt.
Task (dt. Aufgabe):
Im Zusammenhang mit Scheduling wird ein Prozeß häufig auch als Task bezeichnet.
Taskforce (dt. Aufgabengruppe):
Eine Taskforce ist eine Gruppe von Prozessen eines Jobs, die auf mehreren Knoten gleichzeitig ausgeführt wird.
Thread (dt. Faden):
Der Begriff Thread wird im Zusammenhang mit Scheduling synonym mit Prozeß verwendet[1].
Time Slicing (dt. Zeitscheibenzuordnung):
Die Zeit wird in feste Zeitscheiben eingeteilt. Nach Ablauf einer Zeitscheibe wird der Knoten neu zugeteilt, daß heißt, ein neuer Job kann zur Ausführung gebracht werden. Es besteht jedoch auch die Möglichkeit, daß ein Job mehrere Zeitscheiben nacheinander erhält. Time Slicing wird oft auf allen Knoten des MPP synchronisiert, um einen Context-Switch auf mehreren Prozessoren zu ermöglichen.

[1] Üblicherweise ist ein Thread – auch leichter Prozeß genannt – ein Teil eines Prozesses. Die Threads eines Prozesses haben einen gemeinsamen Adreßraum, gemeinsame Dateien, Variablen, Timer und Signale. Jeder Thread hat einen eigenen Stack, Register Satz und Status. Threads werden genutzt, um verschiedene Tätigkeiten eines Prozesses gleichzeitig auszuführen, wobei jeder Thread sequentiell abgearbeitet wird (siehe [Tan95] S. 169ff).

Literaturverzeichnis

[Arn93] A. Arnold: *PARvis: Eine X-basierte Umgebung zur Visualisierung von parallelen Programmen.* Forschungszentrum Jülich, Jül-2848, Jülich, 1993

[AR96] A. Arnold, M. Röth: *User's Guide to VAMPIR. An X-based Visulization Environment for Parallel Programs. Version 2.8b7.* Interne Dokumentation Forschungszentrum Jülich, 1996

[Arn96] A. Arnold: *VAMPIR Trace Format Specification.* Interne Dokumentation Forschungszentrum Jülich, 1996

[AWD96] A.W. Apon, T.D. Wagner, L.W. Dowdy: *A Self Monitoring Approach to Adaptive Processor Allocation.* Department of Computer Science Vanderbilt University, Nashville Tennessee, 1996
http://cswww.vuse.vanderbilt.edu/~peg/publications/sta.ps

[CV96] B. Chen, A.P.A. Vestjens: *Scheduling on identical machines: How good is LPT in an on-line setting?* Eindhoven University of Technology Memorandum 96-11, Eindhoven, 1996
ftp://ftp.win.tue.nl/pub/techreports/cosor/96-11.ps

[DFK96] J. Docter, K. Fälski, R. Knecht: *Paragon XP/S at KFA Jülich.* Forschungszentrum Jülich, KFA-ZAM-TKI-0230, Jülich, 1996

[Feit95] D.G. Feitelson: *A Survey of Scheduling in Multiprogrammed Parallel Systems.* IBM Research Report RC 19790 (87657) Revised Version, New York, 1995
http://www.cs.huji.ac.il/~feit/survey.ps.Z

[Feit96] D.G. Feitelson: *Packing Schemes for Gang Scheduling.* Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph (eds.), Lecture notes in Computer Science Vol. 1162 pp.89-110, Springer-Verlag, 1996
http://www.cs.huji.ac.il/~feit/pack2.ps.Z

[FJ97] D.G. Feitelson, M.A. Jette: *Improved Utilization and Responiveness with Gang Scheduling.* Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph (eds.), Lecture notes in Computer Science Vol. 1291 pp.238-261, Springer-Verlag, 1997
http://www.cs.huji.ac.il/~feit/time.ps.Z

[GR96] J. Gehring, F. Ramme: *Architecture-Independet Request-Scheduling with Tight Waiting-Time Estimations.* IEEE IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing pp.41-54, 1996
http://www.uni-paderborn.de/pc2/services/public/1996/96010.ps.Z

[HB84] K. Hwang, F.A. Briggs: *Computer Architecture and Parallel Processing* McGraw Hill Book Company, 1984

[KMN96] R.B. Konuru, J.E. Moreira, V.K. Naik: *Application-Assisted Dynamic Scheduling on Large-Scale Multi-Computer Systems.* IBM Research Report RC 20390, New York, 1996
http://www.watson.ibm.com/PS/7991.ps.gz

[LHR95] D.A. Lifka, M.W. Henderson, K. Rayl: *Users Guide to the Argonne SP Scheduling System.* Argonne National Laboratory Mathematics and Computer Science Technical Memorandum ANL/MCS-TM-201, Argonne Illinois, 1995
ftp://info.mcs.anl.gov/pub/tech_reports/reports/TM201.ps.Z

[LV90] S.T. Leutenegger, M.K. Vernon: *The Performance of Multiprogrammed Multiprocessor Scheduling Policies.* ACM Sigmetrics Conference of Measurement & Modeling of Computer Systems pp.226-236, 1990

[LWLN97] V. Lo, K. Windisch, W. Liu, B. Nitzberg: *Non contiguous Processor Allocation Algorithms for Mesh-connected Multicomputers.* IEEE Transactions on Parallel and Distributed Computing pp.712-726, 1997
ftp://ftp.cs.uoregon.edu/pub/lo/noncontig.ps.gz

[MEB88] S.Majumdar, D.L. Eager, R.B. Bunt: *Scheduling in Multiprogrammed Parallel Systems.* ACM Sigmetrics 1988 Conference of Measurement & Modeling of Computer Systems pp.104-113, 1988

[Moh96] B. Mohr: *Torus Resources and Node Display Tool Description.* Interne Dokumentation Forschungszentrum Jülich, 1996
http://www.kfa-juelich.de/ZAM/trend

[NSS95] V.K. Naik, S.K. Setia, M.S. Squillante: *Processor Allocation in Multiprogrammed Distributed Memory Parallel Computer Systems.* IBM Research Report RC 20239, New York, 1995
http://www.watson.ibm.com/PS/7842.ps.gz

[Oust82] J.K. Ousterhout: *Scheduling Techniques for Concurrent Systems.* Third International Conference on Distributed Computing Systems pp.22-30, 1982

[Przy92] F. Przybylski: *Scheduling-Verfahren für Rechner mit verteiltem Speicher: Eine vergleichende Übersicht.* Forschungszentrum Jülich Jül-2598, Jülich 1992

[Rott96] J.H.T. Rottinghuis: *Evaluating Scheduling Strategies for Parallel Supercomputers.* Master Thesis, Faculty of Technical Mathematics and Informatics, Department of Technical Informatics, University of Technology, Delft, 1996
http://pgs.twi.tudelft.nl/leerstoel/afstudeerders/scripties/96rottinghuis.ps

[RSDS94] E. Rosti, E. Smirni, L.W. Dowdy, G. Serazzi, B.M. Carlson: *Robust Partitioning of Multiprocessor Systems.* Performance Evaluation Special Issue on Parallel Systems pp.141-165, 1994
http://cswww.vuse.vanderbilt.edu/~peg/publications/perfeval.ps

[RSSD95] E. Rosti, E. Smirni, G. Serazzi, L.W. Dowdy: *Analysis of Non-Work-Conserving Processor Partitioning Policies.* IEEE IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing pp.101-110, 1995

[SAP95] I. Stoica, H. Abdel-Wahab, A. Pothen: *A Microeconomic Scheduler for Parallel Computers.* IEEE IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing pp.122-135, 1995

[Squi95] M.S. Squillante: *On the Benefits and Limitations of Dynamic Partitioning in Parallel Systems.* IEEE IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing pp.136-147, 1995

[SRDS93] E. Smirni, E. Rosti, L.W. Dowdy, G. Serazzi: *Evaluation of Multiprocessor Allocation Policies.* Technical Report, Computer Science Department Vanderbilt University, Nashville Tennessee, 1993
http://cswww.vuse.vanderbilt.edu/~peg/publications/TRcomparison.ps

[SRSD95] E. Smirni, E. Rosti, G. Serazzi, L.W. Dowdy, K.C. Sevcik: *Performance Gains from Leaving Idle Processors in Multiprocessor Systems.* Technical Report, Computer Science Department Vanderbilt University, Nashville Tennessee, 1995
http://cswww.vuse.vanderblit.edu/~peg/publications/idle.ps

[SSG95] T. Suzuoka, J. Subhlok, T. Gross: *Evaluating Job Scheduling Techniques for Highly Parallel Computers.* School of Computer Science Carnegie Mellon University Technical Report CMU-CS-95-149, Pittsburgh Pennsylvania, 1995
ftp://reports.adm.cs.cmu.edu/usr/anon/1995/CMU-CS-95-149.ps

[Tan95] A.S. Tanenbaum: *Distributed Operating Systems.* Prentice-Hall Inc., 1995

[TLWF94] J. Turek, W. Ludwig, J.L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, P.S. Yu: *Scheduling Parallelizable Tasks to Minimize Average Response Time.* ACM Symp. Parallel Algorithms and Architectures pp.200-209, 1994
http://www-ds.e-technik.uni-dortmund.de/reports/spaa.de

[Wal95] C.A. Waldspurger: *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management.* Massachusetts Institute of Technology Technical Report MIT/LCS/TR-667, Cambridge Massachusetts, 1995
http://www.psg.lcs.mit.edu/papers/CarlPhD.ps.gz

Die vorliegende Arbeit wurde am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen erstellt.

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich GmbH anfertigen zu können.

Weiterhin bedanke ich mich bei Herrn Prof. Dr. W.E. Nagel und Herrn D. Erwin für die intensive Betreuung und die konstruktiven Vorschläge bei der Erstellung der Arbeit. Bei Herrn H. Bast der Firma Intel bedanke ich mich für die Unterstützung bei Problemen zu Intel Paragon XP/S 10, bei Frau J. Docter und Frau K. Fälski für die Unterstützung in den Bereichen Cray T3E und Accounting-Daten und bei Herrn A. Arnold für die Hilfe bei der Erstellung des Interfaces zu VAMPIR.