Freie Universität Berlin
Institut für Informatik

Diplomarbeit

Vergleich von Co-Reservierungs-Algorithmen in Grid-Umgebungen

Bearbeiter: Jörg Meltzer
Matrikelnummer 3481446
Betreuer: Prof. Dr.-Ing. habil. Jochen Schiller
(Freie Universität Berlin)
Prof. Dr. Alexander Reinefeld
(Konrad-Zuse-Zentrum für Informationstechnik Berlin und Humboldt-Universität zu Berlin)

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Zusammenfassung

In dieser Arbeit werden effiziente Co-Reservierungs-Algorithmen vorgestellt, mit denen Rechenressourcen mehrerer Cluster im Grid reserviert werden können. Es werden fünf Algorithmen vorgestellt, die flexible Co-Reservierungsanfragen ermöglichen. In einer Co-Reservierungsanfrage müssen die angeforderten CPUs der Teilreservierungen exakt festgelegt werden. Die Cluster und die Startzeiten, bei denen die Teilreservierungen durchgeführt werden, ermittelt ein Co-Reservierungs-Algorithmus erst bei der Anfragebearbeitung. Die Benutzer können die Startzeit in der Anfrage eingrenzen, in dem sie eine früheste Startzeit und eine Deadline angeben. In den bisher entwickelten Reservierungsverfahren können Benutzer nur starre Co-Reservierungsanfragen stellen, in denen sie für alle Attribute der Teilreservierungen exakte Werte angeben müssen. Eine flexible Co-Reservierungsanfrage erlaubt einem Grid-Reservierungsdienst optimierte Reservierungsvarianten durchzuführen, da er aus vielen Reservierungsvarianten auswählen kann. Die Anwendungen von lokalen Benutzern können durch die Grid-Reservierungen verzögert werden. Bei der Auswahl der Reservierungsvarianten kann ein Grid-Reservierungsdienst dies berücksichtigen und den negativen Einfluß der Grid-Reservierungen verringern. Bei zwei Co-Reservierungs-Algorithmen werden auch zusätzliche Sortiervorgaben des Klienten bei der Auswahl der Reservierungsvarianten beachtet. Die Algorithmen wurden durch Simulationen mit dem Workload eines realen großen Supercomputers ausgewertet. Die Untersuchungen zeigen, daß sowohl die lokalen als auch die Grid-Benutzer von den flexiblen Anfragen profitieren können. Den lokalen Benutzern wurden die Ressourcen weniger oft durch zu früh gestellte Reservierungsanfragen entzogen. Die Grid-Benutzer können ihrerseits komfortabel eine optimierte Reservierungsvariante durchführen lassen.

Danksagung

Mein Dank gilt Prof. Dr. Alexander Reinefeld, Leiter des Bereichs Computer Science am Konrad-Zuse-Zentrum für Informationstechnik Berlin und Dr.-Ing. habil. Jochen Schiller, Dekan des Fachbereichs Mathematik und Informatik der Freien Universität Berlin. Weiterhin danke ich meinem Betreuer Thomas Röblitz vom Konrad-Zuse-Zentrum für Informationstechnik, meiner Familie und meinen Freunden für ihre Unterstützung.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit befaßt sich mit dem Thema, wie Rechenressourcen verschiedener Cluster in Grid-Computing-Umgebungen effizient gleichzeitig genutzt werden können. Verschiedene Forschergruppen untersuchten dies in den letzten Jahren [1, 2, 3, 4, 5].
In Grid-Computing-Umgebungen können verschiedene Typen von Ressourcen durch verschiedene Institutionen gemeinsam genutzt werden: z.B. Massenspeicher, Rechenressourcen, Datenbanken und spezielle Anwendungen [5, 6]. Ein Benutzer hat im Grid eine größere Auswahl an Ressourcen zur Verfügung, als nur die Ressourcen seines Heimatinstituts. Die Ressourcen werden durch die Institutionen autonom verwaltet, sie sind geographisch verteilt und über ein Netzwerk miteinander verbunden. Grid-Computing-Grundlagen und -Begriffe werden in Kapitel 2 vorgestellt. Aus verschiedenen Gründen ist es interessant, die Rechenressourcen verschiedener Cluster gleichzeitig nutzen zu können: Ein Grund ist, daß manche Anwendungen über mehrere Cluster verteilt werden müssen, weil sie zu viele Ressourcen benötigen, um in einem Cluster ausgeführt werden zu können. Dies ist zum Beispiel in komplexen Strömungssimulationen der Fall, die im Flowgrid Projekt [7] untersucht werden. Je mehr Details in einer Strömungssimulation betrachtet werden und je größer des Auflösung des Strömungsmodells ist, desto mehr CPU-Ressourcen werden für die Simulation benötigt. Ein zweiter Grund für eine Verteilung von Anwendungen auf mehrere Cluster ist, daß Benutzer eine bessere Antwortzeit für ihre Anwendungen erhalten können.
Angenommen eine Strömungsmessung benötigt eine große Anzahl an Prozessoren. Wenn jeder der Cluster nicht genügend freie CPUs für einen unmittelbaren Job-Start bereitstellt, dann kann der Simulationsstart lange verzögert werden. Es ist möglich, daß die Simulation nach einer Verteilung der Sub-Jobs auf mehrere Cluster unmittelbar starten kann. Möglicherweise dauert die Simulation selbst länger, sie ist jedoch eventuell schneller verarbeitet, wenn die Wartezeit im nicht verteilten Fall hinzugezählt wird.
Die Verteilung der Jobs kann aber auch Probleme verursachen. Abbildung 1 zeigt Effekte, die bei der Verteilung der Sub-Jobs auftreten können. In einer traditionellen Grid-Umgebung werden die Sub-Jobs vom *Grid-Ressource-Broker* nach dem „Best-Effort“ Prinzip auf die Batchsysteme verteilt [6]. Im rechten Teil der Abbildung ist zu sehen, daß die Sub-Jobs der Anwendung gleich viele Batch-Jobs vor sich haben, so daß sie voraussichtlich auch zur selben Zeit starten würden.
Weil die Laufzeiten von Batch-Jobs im voraus nicht bekannt sind, können die Sub-Jobs zu unterschiedlichen Zeiten startbereit werden. Im linken Teil der Abbildung ist zu sehen, daß der Sub-Job in Cluster4 bereits starten kann, während die anderen noch weitere Jobs vor sich in der Warteschlange haben. Wegen der Kommunikation zwischen den Prozessen der Anwendung, müssen alle Sub-Jobs gleichzeitig gestartet werden. Früh startbare Sub-Jobs belegen somit die Ressourcen, obwohl sie diese erst benötigen, wenn auch der letzte Sub-Job startbereit wird.
Es existieren bereits ineffiziente Verfahren mit denen Ressourcen mehrerer Cluster genutzt werden können: In einer der am häufigsten eingesetzten Grid-Middlewares, dem *Globus-Toolkit* [6], wird mit der Software *DUROC* [2] die Ressourcenverwaltung koordiniert. Die Ressourcenauswahl wird beim Planen der Jobs durch Wartezeitvorhersagen für die Ressourcen unterstützt. Die Sub-Jobs werden vor dem Start der Anwendung in einer Barriere synchronisiert. Erst wenn alle Sub-Jobs bereit sind, werden die Jobs gestartet.

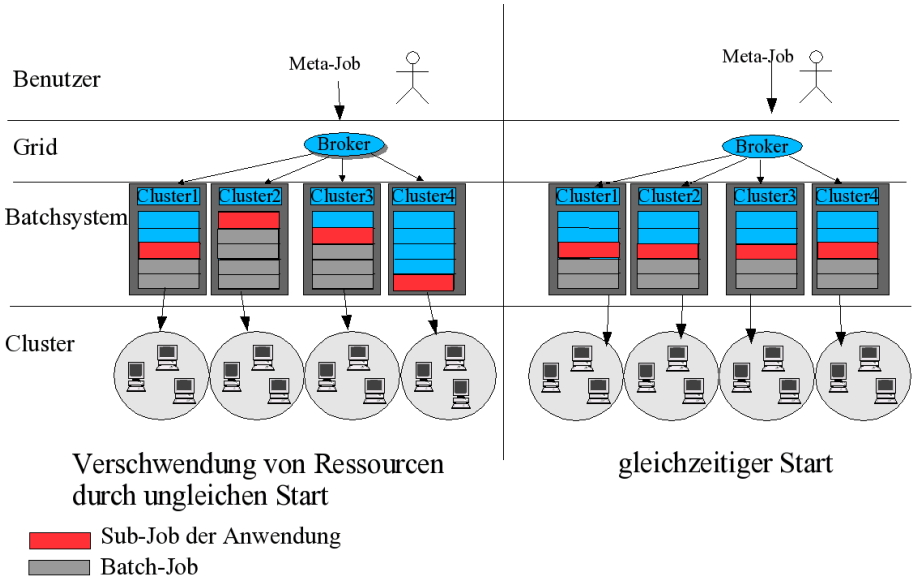


Abbildung 1: Zwei Varianten der Ausführung einer Multi-Cluster-Anwendung

Einen zuverlässigeren Weg, den gleichzeitigen Programmstart zu ermöglichen, bieten Vorausreservierungen. Wenn für alle benötigten Ressourcen bei den benutzten Clustern Reservierungen erfolgreich durchgeführt werden, dann ist garantiert, daß alle Jobs zur vereinbarten Startzeit starten können. Vorausreservierungen werden heute in kommerziellen Batchsystemen, wie PBSPro [8] und LSF [9] angeboten. Eine Open-Source Alternative ist mit dem Scheduler Maui [10] möglich. Er ersetzt die Standard-Scheduler von Batchsystemen, die von sich aus keine Vorausreservierungen unterstützen. Eine Kombination mehrerer Reservierungen wird auch Co-Reservierung genannt. Mit der von Foster et al. entwickelten Software *GARA* [3] können Co-Reservierungen durchgeführt werden. Die Cluster besitzen bei GARA lokale Reservierungsdienste, über die Ressourcen im voraus reserviert werden können. GARAs Fokus beschränkt sich jedoch auf die Zusammenarbeit zwischen den lokalen Reservierungsdiensten und den verschiedenen zugrundeliegenden Batchsystemen. In Co-Reservierungsanfragen müssen der genaue Ort, die genaue Startzeit und der exakte Ressourcenbedarf von Teilreservierungen angegeben werden. Die starren Co-Reservierungsanfragen bieten Grid-Benutzern deshalb keinen Komfort und stellen ein Hindernis für eine effiziente Ressourcenverwaltung im Grid dar. Einige Probleme die bei GARA noch nicht gelöst sind, sind beispielsweise:

- Die Teilreservierungsanfragen können scheitern, wenn beim angefragten Cluster nicht genügend Ressourcen verfügbar sind, Ein Benutzer muß die Reservierungsanfragen daher selbst variieren, um zu anderen Konditionen eine Co-Reservierung erhalten zu können.
- Die Teilreservierungen können wartende Jobs in den Batchsystemen in unfairer Weise verzögern, da sie außerhalb der sonst geltenden, lokalen Scheduling-Regeln einen Zugang zu den Ressourcen gewähren.
- Die Grid-Benutzer verfügen potentiell nur über wenige Informationen zu den verfügbaren Ressourcen im Grid. Sie können daher selbst keine optimierten Reservierungsanfragen stellen. Die Anfrageoptimierungsziele könnten sein: die angeforderten Ressourcen

zur frühesten möglichen Startzeit zu reservieren oder sie zu besonders günstigen Konditionen zu erhalten.

- Die Ressourcen eines Clusters können bei zu vielen Reservierungsanfragen lokalen Benutzern völlig entzogen werden.

In dieser Arbeit werden fünf Co-Reservierungs-Algorithmen vorgestellt. Sie bauen aufeinander auf und werden mit dem Ziel entwickelt, die Aufzählungspunkte Schritt für Schritt zu berücksichtigen. Die Algorithmen erlauben Klienten flexible Reservierungsanfragen stellen können. In einer Reservierungsanfrage müssen die angeforderten CPUs der Teilreservierungen exakt festgelegt werden. Die Orte und die Startzeiten der Teilreservierungen werden erst bei der Anfragebearbeitung durch den Co-Reservierungs-Algorithmus ermittelt. Die Klienten können die Startzeit in der Anfrage eingrenzen, in dem sie eine früheste Startzeit und eine Deadline angeben. Diese Anfragen erlauben potentiell viele Reservierungsvarianten. Bei zwei Algorithmen können die Benutzer in den Anfragen zusätzliche Optimierungsziele vorgeben. Ein klientenseitiger Dienst nimmt zu den flexiblen Anfragen konkrete Reservierungsversuche vor. Sein Ziel ist es, eine mögliche Variante für jede Teilreservierung durchzuführen.
Simulationen mit dem Workload eines realern Supercomputers werden durchgeführt, um die Algorithmen zu untersuchen. Zum ersten Aufzählungspunkt werden die Fehlschlagraten von Co-Reservierungsanfragen und die Anzahlen der Reservierungsversuche bei Anfragebearbeitungen untersucht. Zum zweiten Aufzählungspunkt wird ein Fairnesskriterium untersucht. Außerdem wird analysiert, wie gut die Anfrageoptimierungziele bei den Algorithmen, die diese Optimierungen unterstützen, beachtet werden. Die Batchsystem-Auslastungen und die Job-Wartezeiten werden ebenfalls untersucht.
In Kapitel 3 wird das Co-Reservierungsmodell beschrieben. Anschließend werden die Algorithmen in Kapitel 4 entworfen. In Kapitel 5 wird genauer auf die verwendete Simulationsumgebung eingegangen und in Kapitel 6 auf die Untersuchungen.

2 Grid-Computing

Mit dem Globus-Toolkit [6] können Grids, in denen Benutzer für ihre Anwendungen nur Ressourcen eines Clusters benötigen, effizient realisiert werden. In Abbildung 2 ist dargestellt, wie in diesem Fall ein Programm ausgeführt werden kann:

1. Benutzer

Ein Benutzer legt in einem Meta-Job den Ressourcenbedarf des Programms (Anzahl der CPUs, Speicher, Netzwerkverbindungen etc.) fest und übermittelt den Meta-Job an einen Ressource-Broker.

2. Grid / Ressource-Broker

In periodischen Abständen übermitteln die Cluster des Grids Statusinformationen der Ressourcen an einen Informationsdienst. Mit der Meta-Job-Beschreibung des Programms sucht ein Ressource-Broker über den Informationsdienst nach den benötigten Ressourcen. Programme können üblicherweise nicht direkt an den Rechnern eines Clusters ausgeführt werden. Die Rechner werden am Cluster lokal von einem Batchsysteme verwaltet. Batchsystem-Benutzer müssen die Ressourcenanforderungen ihres Programms abschätzen, dessen Parameter und Laufzeitumgebung definieren und diese zusammen in Form einer Job-Beschreibung im Batchsystem einreichen. Der Ressource-Broker übermittelt den Meta-Job des Grid-Benutzers als lokalen Job in ein Batchsystem eines Clusters.

3. Cluster-Ressource-Verwaltung / Batchsystem

Typischerweise werden Batch-Jobs in einer Warteschlange verwaltet, die nach und nach abgearbeitet wird. Wenn ein Job an der Reihe ist und alle Ressourcen verfügbar sind, wird er vom Batchsystem-Scheduler ausgewählt und auf den Rechnern des Clusters gestartet.

4. Cluster

Das Programm, das im Meta- (1) und Batch-Job (3) beschrieben ist, wird auf einem oder mehreren Rechnern des Clusters ausgeführt.

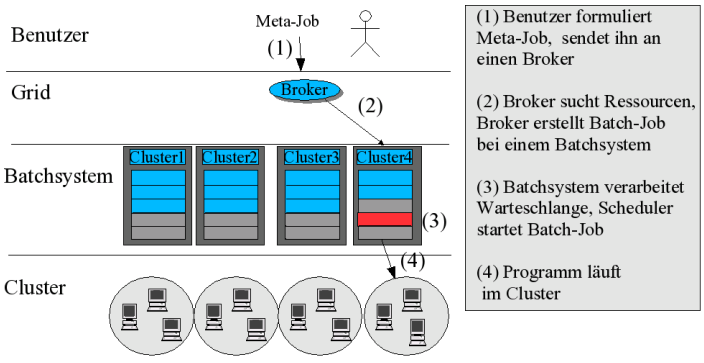


Abbildung 2: Ausführung eines Meta-Jobs im Grid

3 Co-Reservierungsmodell

Abbildung 3 zeigt ein Beispiel für eine Co-Reservierungsanfrage. Anhand des Beispiels werden nun die Begriffe des Co-Reservierungsmodells erklärt:

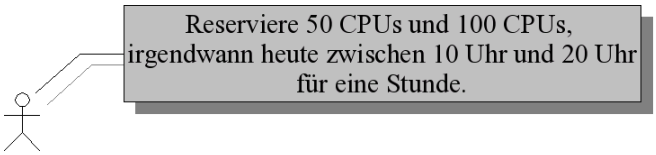


Abbildung 3: Beispiel für eine Co-Reservierungsanfrage

Teilreservierung

Eine Teilreservierung kann in dieser Arbeit ausschließlich für eine Anzahl an homogen CPUs vorgenommen werden. Weitere Ressourcentypen werden nicht betrachtet.

Co-Reservierungsanfrage

Tabelle 1 zeigt das formale Modell einer Co-Reservierungsanfrage. In der Formatspalte der Tabelle werden die Attributwertebereiche in einer Haskell-Syntax angegeben[1]. Eine Co-Reservierungsanfrage kann aus mehreren Teilreservierungsanfragen aufgebaut werden. Die angeforderten Teilreservierungen haben die gleiche Dauer und müssen an jeweils verschiedenen Clustern zur selben Startzeit durchgeführt werden. Eine früheste Startzeit *f_start* und eine späteste Endzeit *deadline* schränken die Startzeiten der Teilreservierungen ein. Die beiden Zeiten sind definiert als Sekunden seit '00:00:00 1970-01-01 UTC' und geben damit ein exaktes Datum an. Die Dauer jeder Teilreservierung wird mit dem Parameter *dauer* in Sekunden angegeben. Die Startzeitspanne einer Reservierungsanfrage ist somit definiert als $deadline - f\_start - dauer$.

Attribute	Beschreibung	Format	Beispiel Abbildung 3
f_start	Früheste Startzeit jeder Teilreservierung	`Zeit`	`1088764000`
deadline	Späteste Endzeit jeder Teilreservierung	`Zeit`	`1088800000`
dauer	Dauer jeder Teilreservierung	`Zeit`	`3600`
res	Teilreservierungen/#CPUs	`[Int]`	`[100,50]`

Tabelle 1: Co-Reservierungsanfrage

Ressource / Cluster-Kandidat

Ein Cluster-Kanditat ist ein möglicher Cluster, an dem eine Teilreservierung ausgeführt werden kann. Er bietet eine maximale Anzahl an CPUs *max_cpus* über einen Co-Reservierungsdienst an und hat eine eindeutige ID *cid* (Tabelle 2).

Teilreservierungsvariante / TR

Eine Kombination von einer Teilreservierung, einer Startzeit und einem Cluster-Kandidat beschreibt eine Teilreservierungsvariante, die bei einer Anfragebearbeitung potentiell reserviert werden kann. Die ID der Teilreservierung *res_id* bezieht sich auf die Position in der Ressourcenliste *res* der Co-Reservierungsanfrage.

[1] Eine kurze Übersicht über die verwendeten Haskell-Konzepte wird in Appendix A gegeben.

Attribute	Beschreibung	Format	Beispiel
cid	ID des Clusters	`ID`	`1`
max_cpus	Maximale Anzahl verfügbarer CPUs	`Int`	`200`

Tabelle 2: Ressource / Cluster-Kandidat

Attribute	Beschreibung	Format	Beispiel
cid	ID des Cluster-Kandidaten	`ID`	`1`
res_id	ID der Teilreservierung	`ID`	`0`
start	Startzeit der Teilreservierung	`Zeit`	`1088764000`

Tabelle 3: Teilreservierungsvariante / TR

Co-Reservierung

Ziel der Co-Reservierungsalgorithmen ist eine Co-Reservierung vorzunehmen. Eine Co-Reservierung ist eine der potentiell vielen möglichen Co-Reservierungsvarianten.

Co-Reservierungsvariante

Abbildung 4 zeigt ein Beispiel für eine Co-Reservierungsvariante zur Beispielanfrage und Tabelle 4 zeigt das formale Modell. Eine Co-Reservierungsvariante ist ein Tupel bestehend aus der Startzeit *start* und der Ressourcenzuordnung *map* der Teilreservierungen. Das erste Element in *map* enthält den Cluster-Kandidat, dem die erste Teilreservierung der Co-Reservierungsanfrage zugeordnet ist, das zweite Element die Zuordnung der zweiten Teilreservierung und so weiter.

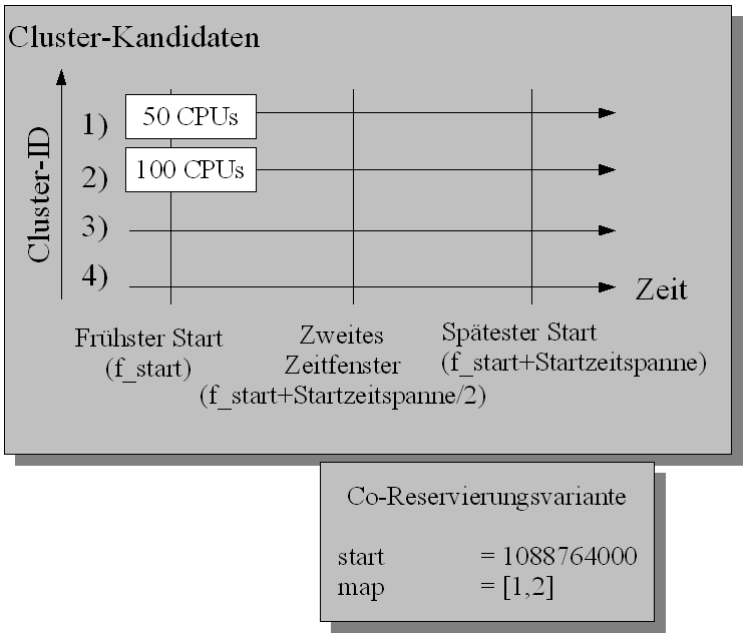


Abbildung 4: Co-Reservierungsvariante zur Beispielanfrage von Abbildung 3

Attribute	Beschreibung	Format	Beispiel Abbildung 4
start	Startzeit der Teilreservierungen	`Zeit`	1088764000
map	Abbildungen Teilreservierung zu Cluster	`[ID]`	`[1, 2]`

Tabelle 4: Co-Reservierungsvariante

Die Bildung einer Co-Reservierungsvariante umfaßt drei Schritte: die Festlegung der möglichen Startzeiten, die Auswahl einer Kombination an Cluster-Kandidaten und die Bestimmung einer Reihenfolge der Teilreservierungen. Für die Beispielanfrage aus Abbildung 3 sollen 50 und 100 CPUs reserviert werden. Die Startzeit der Teilreservierungen muß zwischen der frühesten Startzeit *f_start* und der spätesten Startzeit *f_start+Startzeitspanne* liegen. Beim Beispiel der Co-Reservierungsvariante von Abbildung 4 wurden drei Startzeiten betrachtet, von denen die erste ausgewählt wurde. Die nächste Auswahlmöglichkeit besteht bei der Ressourcenabbildung. Insgesamt stehen vier Cluster-Kandidaten zur Anfragebearbeitung zur Verfügung, von denen die ersten beiden ausgewählt wurden. Sechs Kombinationen der Kandidaten sind denkbar, wenn jeder Cluster-Kandidat über genügend CPUs verfügt. Zu einer gegebenen Auswahl an Cluster-Kandidaten kann schließlich noch die Reihenfolge der Teilreservierungen variiert werden. Zwei Permutationen bietet das Beispiel. Insgesamt gibt im Beispiel durch Startzeit-, Cluster- und Ressourcenvariation 36 verschiedene Co-Reservierungsvarianten. Im Allgemeinen läßt sich ihre Anzahl mit Gleichung 1 berechnen.
Seien z die Anzahl möglicher Startzeiten, c die verfügbaren Cluster und r die Anzahl der Teilreservierungen, dann berechnet sich die Anzahl der Variationen wie folgt:

$$Variationen = z \cdot \binom{c}{r} \cdot r! \tag{1}$$

Ein Co-Reservierungsdienst kann versuchen so oft Co-Reservierungsvarianten durchzuführen, bis ihm eine Co-Reservierung gelingt oder keine weiteren Varianten mehr möglich sind. Diese Vorgehensweise bringt drei Vorteile:

1. Die Reservierung einer Co-Reservierungsvariante muß nicht unmittelbar zum Erfolg führen, denn die Verfügbarkeit von Ressourcen an den Clustern kann insbesondere für zukünftige Zeitpunkte nur abgeschätzt werden. Ohne die Unterstützung eines Co-Reservierungsdienstes, der selbständig die Varianten nach und nach bis zum Erfolg zu reservieren versucht, müßte der Klient jede Co-Reservierungsvariante selbst durch starre Reservierungen formulieren. Den Aufwand nach Reservierungsfehlschlägen andere Varianten auszuprobieren nimmt der Dienst dem Klienten ab. Tabelle 5 zeigt die Variantenvielfalt von Co-Reservierungsanfragen für verschiedene Parameter. Bereits wenige Teilreservierungen führen zu einer hohen Variantenanzahl, so daß eine Automatisierung des Reservierungsprozesses vorteilhaft ist.
2. Auch wenn zu einem Zeitpunkt Ressourcen an einem Cluster verfügbar sind, kann eine Teilreservierung unerwünscht sein. Lokale Fairness-Regeln in Batchsystemen sollten durch die Teilreservierungen nicht verletzt werden. Ein Co-Reservierungsdienst kann dies bei der Auswahl der Teilreservierungsvarianten bei einem Cluster-Kandidat berücksichtigen. Die Varianten, die mit den Fairness-Regeln im Konflikt stehen, können im Vorfeld verworfen werden.

z = 1	c = 4	c = 8	c = 12
r = 1	4	8	12
r = 2	12	56	132
r = 3	24	336	1320
r = 4	24	1680	11880

Tabelle 5: Mögliche Co-Reservierungsvarianten nach Gleichung 1

3. Bei einem automatisierten Reservierungsprozess kann ein Co-Reservierungsdienst auch besser zusätzliche Auswahlkriterien berücksichtigen. Wenn ein Klient zum Beispiel ein dringendes Experiment zum frühesten möglichen Zeitpunkt durchführen will, dann definiert er die Anfrageparameter *f_start* als gegenwärtigen Zeitpunkt und gibt als Parameter *deadline* den spätesten noch akzeptablen Wert ein. Zum Zeitpunkt der Anfrage und der frühen Startzeiten haben die Cluster durch laufende und wartende Jobs hohe Auftragsbestände, dadurch sind frühe Reservierungsvarianten häufiger von Reservierungsfehlschlägen betroffen. Die automatisierte Prüfung und Reservierung von frühen Co-Reservierungsvarianten hilft hier besonders, denn der Klient müßte unter Umständen viel Zeit investieren, um eine ähnliche frühe Co-Reservierungsvariante mit selbst gestellten unflexiblen Anfragen zu erreichen.

4 Algorithmen

In diesem Kapitel werden fünf Co-Reservierungsalgorithmen vorgestellt. Algorithmus 1 beschreibt ein generelles Schema mit dem Co-Reservierungen durchgeführt werden können. In Algorithmus 2 wird ein Mechanismus eingeführt, der die Anzahl von Teilreservierungsversuchen bis zum Finden einer erfolgreich durchführbaren Co-Reservierungsvariante verringert. Algorithmus 3 führt eine Fairness-Regel ein, in der die Ressourcen für wartende Batch-Jobs, wie in Kapitel 3 vorgeschlagen, geschützt werden. In Algorithmus 4 ist es möglich, Co-Reservierungsanfragen zu den frühesten möglichen Startzeiten durchzuführen. Algorithmus 5 zeigt wie beliebige, kombinierte Sortierkriterien formuliert werden können. An ihm wird demonstriert wie Co-Reservierungsvarianten gleichzeitig nach der frühesten möglichen Startzeit und den geringsten virtuellen Kosten optimiert werden können.

4.1 Generelle Vorgehensweise

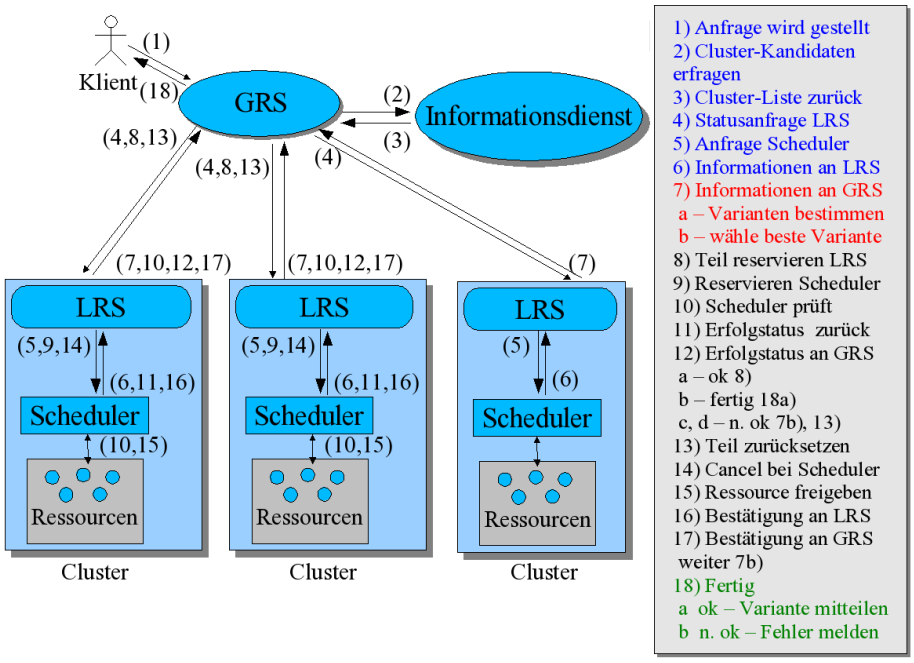


Abbildung 5: Zusammenarbeit der Komponenten bei der Bearbeitung einer Co-Reservierungsanfrage

Eine Co-Reservierungsanfrage wird durch mehrere verteilte Komponenten bearbeitet, in Abbildung 5 sind diese dargestellt. Die Aufgaben der Komponenten werden nun erläutert:

Globaler Reservierungsdienst - GRS

Der GRS nimmt die Co-Reservierungsanfrage eines Klienten entgegen (1). Von einem Informationsdienst erhält er eine Liste von Clustern, die Reservierungdienste anbieten (2,3). In Schritt (4) erfragt er von den lokalen Reservierungsdiensten (LRS) der verfügbaren Cluster Zustandsinformationen. Mit Hilfe der gewonnenen Informationen werden vom Co-Reservierungsalgorithmus mögliche Co-Reservierungsvarianten berech-

net (7a). Eine der Varianten wird anschließend ausgewählt (7b). Die Teilreservierungen der Variante werden nach und nach an die vorgesehenen Cluster verteilt (8, 12a). Wenn alle Teile reserviert sind, wird dem Klient die durchgeführte Variante mitgeteilt (12b). Wenn sich eine der Ressourcen der Variante zu einer bestimmten Zeit nicht reservieren läßt, dann kann der GRS versuchen die bestehenden Teile zu erhalten und verbleibende von einer anderen Variante zu reservieren (12 c). Falls keine Variante mit den schon reservierten Teilen möglich ist, dann muß er die letzte Teilreservierung zurücksetzen lassen (12 d).

Lokaler Reservierungsdienst - LRS
Der LRS stellt die Schnittstelle zum lokalen Batchsystem dar und bietet Methoden für die Ermittelung von Zustandsinformationen an. Er führt die Anfragen (Status-, Reservierungs- oder Rücksetzungsanfrage für eine Teilreservierung (5,9,14)) systemspezifisch bei dem zugehörigen Batchsystem-Scheduler durch.

Batchsystem-Scheduler
Der Scheduler reserviert letztendlich die CPU-Ressourcen (10) oder gibt sie frei (15).

In den vorgestellten Algorithmen gilt eine Reservierungsanfrage nach (18) entweder als bewilligt oder abgelehnt.

4.2 Algorithmus 1: Trivialer Algorithmus

Als erstes wird ein sehr einfacher Algorithmus vorgestellt[2]. Seine Eingabe ist eine Reservierungsanfrage (*f_start, deadline, dauer, res*) (siehe Tabelle 1). Die Aufgabe ist eine Co-Reservierungsvariante (*start, map*) zu reservieren (siehe Tabelle 4 und Abbildung 4).
Algorithmus 1 ist in drei Schritte aufgeteilt:

1. Bestimmung der Teilreservierungsvarianten (Abschnitt 4.2.1)
2. Bildung der Co-Reservierungsvarianten (Abschnitt 4.2.2)
3. Reservierungsschritt (Abschnitt 4.2.3)

4.2.1 Bestimmung von Teilreservierungsvarianten

Die Bestimmung der Teilreservierungsvarianten umfaßt die Schritte 4-7 der generellen Vorgehensweise (vgl. Abschnitt 4.1). In den Schritten 4-6 wird die Verfügbarkeit der Ressourcen ermittelt. Algorithmus 1 nimmt an, daß immer Ressourcen verfügbar sind, d.h. diese Schritte entfallen. Für jedes der Attribute einer Teilreservierungsvariante bestehen mehrere Wahlmöglichkeiten (vgl. Abbildung 4 und Tabelle 3). In Algorithmus 1 wird aus den verschiedenen Kombinationsmöglichkeiten der Attributwerte eine Liste von Teilreservierungsvarianten erstellt. Die Funktion *tr* in Listing 1 zeigt, wie diese Liste ermittelt wird. Für die Attribute *(cid,res_id,start)* werden drei Listen mit ausgewählten Werten erzeugt, jede mögliche Kombination der Werte gebildet und, falls zulässig, die Teilreservierungsvarianten erstellt.

[2]Zur Beschreibung der Algorithmen wird die funktionale Programmiersprache Haskell verwendet. Die für das Verständnis der Algorithmen nötigen Begriffe und Schlüsselwörter sind in Appendix A erklärt.

```
1  --   Anfrage = (f_start, deadline, dauer, res)
2  type Anfrage = (Zeit, Zeit, Zeit, [Int])
3  --   Ressource = (cid, max_cpus)      Ressourcekandidaten vom Informationsdienst
4  type Ressource = (ID, Int)
5  --   TR = (cid, res_id, start)
6  type TR = (ID, ID, Zeit)
7
8  startzeiten f_start deadline dauer minsize max_zeitfenster =
9    [f_start, f_start + increment .. (deadline - dauer)]
10     where startzeitspanne = deadline - f_start - dauer
11           increment       = max minsize (startzeitspanne / max_zeitfenster)
12
13 tr::[Ressource] -> Anfrage -> [TR]
14 tr cluster (f_start, deadline, dauer, res) minsize max_zeitfenster
15   = [(cid, res_id, start) |
16         -- Generatoren für aufgespannten Teilreservierungsvariantenraum
17         start <- (startzeiten f_start deadline dauer minsize max_zeitfenster)
18         (cid, max_cpus) <- cluster,
19         res_id <- [0 .. length res],
20         -- Zugelassene Varianten
21         max_cpus >= (sortiere_desc res)!!res_id]
```

Listing 1: Bestimmung betrachteter Teilreservierungsvarianten

Die Wertebereiche der Teilreservierungsattribute werden nun beschrieben:

cid

Die Cluster-ID *cid* wird aus einer Liste der Cluster-Kandidaten *cluster* ausgewählt, die der Informationsdienst dem Globalen Reservierungsdienst nach einer Anfrage (Schritt 3 der Abbildung 5) zur Verfügung stellt.
Jedes Element aus *cluster* ist ein Tupel aus einer Cluster-ID und dem Wert der maximal verfügbaren CPU-Anzahl an einem Cluster *max_cpus*. Teilreservierungsvarianten werden nicht erstellt, bei denen die maximale CPU-Anzahl der Cluster für die Teilreservierung nicht ausreicht.

res_id

Die großen Teilreservierungen werden bei Algorithmus 1 zuerst vorgenommen. Da große Teilreservierungen vermutlich häufiger scheitern als kleine, ist es günstig undurchführbare Co-Reservierungsvarianten frühzeitig zu erkennen. Um die Durchführung der Co-Reservierung effizienter zu gestalten, werden die Teilreservierungen *res* der Co-Reservierungsanfrage daher absteigend sortiert. Das Attribut *res_id* bezieht sich auf die Position der Teilreservierung innerhalb der sortierten Liste.

start

Eine Liste der Startzeiten (Zeile 8) aus denen die Werte *start* gebildet werden können, werden in Algorithmus 1 wie folgt bestimmt: Sei die Reservierungsanfrage nun *(f_start, deadline, dauer, res)*. In Algorithmus 1 wird die maximale Anzahl an Startzeiten definiert durch *max_zeitfenster*. In konstantem zeitlichen Abstand definiert durch *increment* werden Startzeiten aus der gesamten Startzeitspanne ausgewählt. Es können auch weniger als *max_zeitfenster* viele ausgewählt werden, denn bei sehr kurzen Startzeitspannen macht es kaum Sinn, etwa sekündlich, die Verfügbarkeit der Ressourcen zu

prüfen. Batch-Jobs laufen in einem Zeitrahmen von Minuten oder Stunden, die Auslastung des Batchsystems würde in den folgenden Sekunden sehr wahrscheinlich noch die selbe sein. Eine Co-Reservierung würde durch die hohe Anfragefrequenz in einem kurzen Intervall bei einem belegten Cluster nicht wesentlich erfolgreicher ausgeführt werden können aber viele Reservierungsfehlschläge verursachen. Ein Parameter *minsize* gibt deshalb eine untere Schranke für die Startzeitinkrements an.

4.2.2 Bestimmung von Co-Reservierungsvarianten

Im nächsten Schritt wird aus einer Liste der Teilreservierungsvarianten *tr* durch Kombination der Elemente eine Liste von Co-Reservierungsvarianten *cr* konstruiert. Eine Datenstruktur wird benötigt, in der schon kombinierte Co-Reservierungsvarianten eingetragen werden. Es bietet sich an, dies in einer Baumstruktur festzuhalten, in der jeder Pfad zu einem Blatt des Variantenbaumes eine Co-Reservierungsvariante repräsentiert. Ein Beispiel der Baumerzeugung ist in Abbildung 6 dargestellt. Die Höhe des Variantenbaumes entspricht der Ressource-ID *res_id* einer Teilreservierungsvariante. Als Kinder der Wurzel werden von *tr* alle Teilreservierungsvarianten für die erste Ressource (*res_id 0*) eingefügt. Bei den folgenden Knoten sind die Einschränkungen von Co-Reservierungsvarianten zu berücksichtigen. In die Kinderlisten von Nicht-Wurzelknoten werden nur noch die Teilreservierungsvarianten eingefügt, deren *res_id* der Tiefe des Baums entspricht, deren Startzeit *st* gleich der des Elternknotens ist und deren Cluster-ID *cid* im Pfad noch nicht vorkommt. Die Funktion *baum* in Listing 2 zeigt im Detail wie die Teilreservierungsvarianten in die Baumstruktur überführt werden.

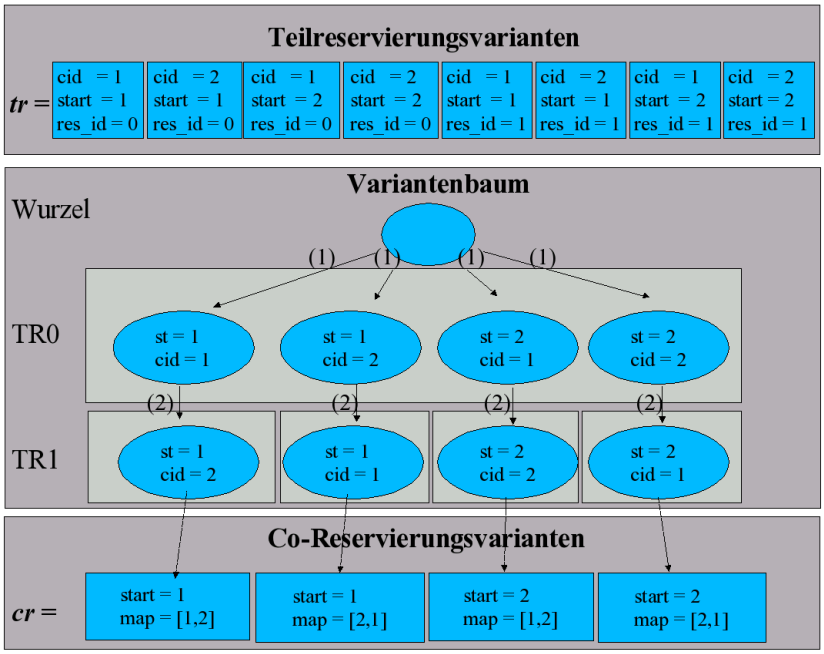


Abbildung 6: Variantenbaum

Datentypen:
In der Variantenbaumstruktur ist ein Knoten definiert als *N kinder cid st* (N ist Haskell-Konstruktor). *kinder* beschreibt die Unterbäume, die für alle Teilreservierungsvarianten für die nächsten Ressourcen gebildet werden, *cid* und *st* beschreiben Cluster-ID und Startzeit der durch den Knoten repräsentierten Teilreservierungsvariante.

Knotenerzeugung:
Die Funktion *ekind* erzeugt einen Knoten für die Teilreservierungsvariante *(cid, re, st)*. *tr* sind die noch für den Unterbaum des Knotens einzufügenden Teilreservierungsvarianten. Die Baumwurzel wird mit dem Tripel *(-1,-1,-1)* beschrieben. Initial sind alle Teilreservierungsvarianten *tr* in den Baum einzufügen. Die Teilreservierungsvarianten werden randomisiert, damit bei der Reservierung eine gleichmäßige Verteilung der Teilreservierungen auf die verfügbaren Cluster und Startzeiten gegeben ist.

Rekursion:
Die Variantenbaumliste *ekinder* beschreibt die zur Teilreservierung *(cid, re, st)* zugehörigen Unterbäume. Alle Varianten aus *tr*, deren *res_id* um eins größer als *re* ist (Zeile 33 Listing 2) sind jeweils Knoten der nächsten Baumebene. Für die Wurzel wird das *res_id* Attribut auf -1 gesetzt und erlaubt somit das Hinzufügen aller Teilreservierungsvarianten für die erste Ressource (*res_id 0*) (siehe Schritt 1 in Abbildung 6). Die Liste der möglichen Teilreservierungsvarianten, die noch in einen Unterbaum eingefügt werden kann, wird anschließend für jeden weiteren eingefügten Knoten gefiltert, so daß die selbe Cluster-ID nicht mehr vergeben werden darf (Zeile 30), die Startzeiten der Teilreservierungsvarianten gleich sind (Zeile 28) und Varianten nur für die nächsten Teilreservierungen erstellt werden dürfen (Zeile 29).

Aufnahme vollständiger Pfade:
Der Variantenbaum ist eine Vorlage für den Reservierungsschritt. In den Baum sollten deshalb nur vollständige Co-Reservierungsvarianten eingefügt werden. Die Pfade sollten deshalb auch immer maximale Länge haben. Aus Gründen der Übersichtlichkeit wurde dies aus Listing 2 ausgeblendet.

```
1  -- Typ und Datenstrukturdefinitionen
2  --    Anfrage    (f_start, dauer, deadline, res)
3  type Anfrage = (Zeit, Zeit, Zeit, [Int])
4  --    TR    (cid, re, st)
5  type TR = (ID, ID, Zeit)
6  -- Haskell Definition Variantenbaum
7  --    Variantenbaum       kinder          cid st
8  data Variantenbaum = N [Variantenbaum] ID Zeit | Nil
9
10 -- Wurzel Erzeugung
11 baum :: [TR] -> Variantenbaum
12 baum tr = ekind (-1,-1,-1) (randomize tr)
13
14 ekind :: TR -> [TR] -> Variantenbaum
15 ekind (cid, re, st) tr = N ekinder cid st
16  where
17    ekinder :: [Variantenbaum]
18    ekinder =
19        [ekind (iid, ire, ist) filtertr |
20        -- Kandidaten für Kinder
21        (iid, ire, ist) <- tr,
22        -- Varianten, die in den Unterbaum von TR (iid,ire, ist)
23        -- eingefügt werden dürfen
24        filtertr :: [TR]
25        filtertr <- [[(id2,re2,st2)|
26                      (id2,re2,st2) <- tr, -- Kandidaten für Kindeskinder
27                     -- Kindeskinder einschränken
28                     ist == st2,  -- gleiche Startzeit
29                     ire /= re2,  -- aktuelle Ressource-ID ist schon eingefügt
30                     iid /= id2]] -- Clusterverschiedenheit beachten
31
32        -- Baumtiefe == Ressource-ID der Teilreservierungsvariante beachten
33        re + 1 == ire]
```

Listing 2: Bestimmung der Co-Reservierungsvarianten

Unvollständige Pfade werden durch folgende Rekursion entfernt:

- Wenn ein Pfad vor Erreichen der maximalen Tiefe verendet, dann wird statt der nächsten Teilreservierung ein Nil Zeiger eingefügt.
- Auf der Vaterebene werden Nil Zeiger aus deren Kinderliste entfernt. Ist die neue Kinderliste leer, wird auch der Vaterknoten durch einen Nil Zeiger ersetzt.
- Die Rekursion wird fortgesetzt, bis ein Knoten mindestens ein Kind nach dem Filtern behält oder die Wurzel erreicht ist.

4.2.3 Reservierungsschritt

In diesem Abschnitt wird der Reservierungsschritt auf der Ebene des GRS dargestellt, das sind die Schritte 8 und 13 im Schema der generellen Vorgehensweise (Abbildung 5). Der Variantenbaum beschreibt durch seine Pfade zu den Blättern Co-Reservierungsvarianten, die vorgenommen werden können. Im Reservierungsschritt wird nun dieser Baum in Reihenfolge der Tiefensuche durchlaufen. Dem Auf- und Absteigen im Baum während der Tiefensuche entsprechen Durchführungen und Rücksetzungen der Teilreservierungen bei den Clustern der jeweils betrachteten Knoten. Die zu reservierende Co-Reservierungsvariante wird immer nur schrittweise für die nächste Teilreservierung gebildet. Eine Co-Reservierungsanfrage ist erfolgreich, wenn bei der Tiefensuche bis zu einem Blatt abgestiegen werden konnte und somit alle ihre Teilreservierungsvarianten vorgenommen sind. Sie ist gescheitert, wenn wegen Teilreservierungsfehlschlägen nie bis zu einem Blatt abgestiegen werden konnte.
Wenn beim Abstieg im Baum die durch den nächsten Knoten repräsentierte Teilreservierung reserviert werden soll, dann kann diese entweder scheitern oder erfolgreich verlaufen. Bei einem Fehlschlag werden alle Varianten des fehlgeschlagenen Unterbaum verworfen. Abbildung 7 zeigt eine partielle Reservierung bis zu einem Knoten *knoten*. Als erstes wird versucht die Variante des ersten Kindes von *knoten* zu reservieren (1). Bei Reservierungserfolg wird der Algorithmus für kind1 wiederholt. Scheitert (1), dann wird versucht weitere Kinder von *knoten* zu reservieren (2). Wenn von einem Knoten kein weitergehender Pfad mehr möglich ist, dann muß die durch *knoten* repräsentierte Teilreservierung zurückgesetzt werden und versucht werden weiter bei Kindern des Vaterknotens zu reservieren (3).

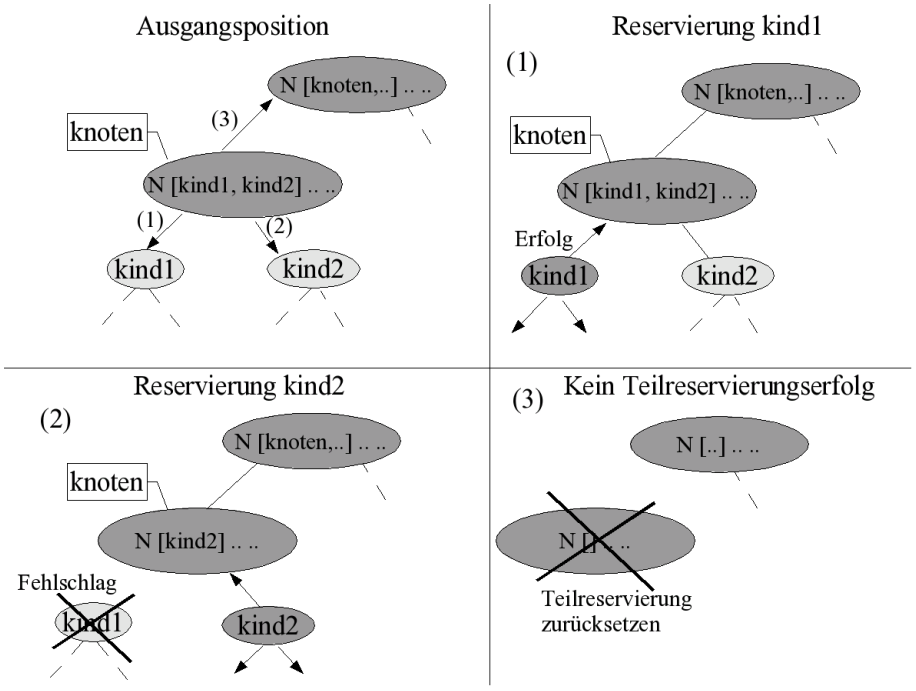


Abbildung 7: Reservierungsschritt am Beispiel eines Knotens - Ausschnitt Variantenbaum

4.2.4 Analyse

In Kapitel 6.4 wird Algorithmus 1 untersucht. Ziel der Untersuchung war es die Anzahl der benötigten Zeitfenster pro Cluster-Kandidat zu bestimmen, so daß gleichzeitig eine effiziente Reservierung und hohe Erfolgsaussichten für Co-Reservierungsanfragen ermöglicht wurden. Generell wurden für Anfragen weniger Varianten probiert, je weiter in der Zukunft Startzeiten für Co-Reservierungsvarianten lagen und je mehr Cluster-Kandidaten für Teilreservierungen zur Verfügung standen. Die Erfolgschancen der Anfragen stiegen unter den selben Bedingungen. Unter den beschriebenen günstigen Umständen genügten bereits wenige Zeitfenster, um Anfragen erfolgreich reservieren zu können. In Simulationen wurde festgestellt, daß, wenn viele Co-Reservierungsvarianten für Anfragen gebildet wurden, nie alle ausprobiert werden mußten. Die Ursache dafür war, daß Teilreservierungsfehlschläge bei Algorithmus 1 häufiger bei Beginn der Durchführung von Co-Reservierungsvarianten auftraten, als an deren Ende. Wenn viele Co-Reservierungsvarianten für Anfragen möglich waren, traten trotzdem in Einzelfällen sehr ineffiziente Anfragebearbeitungen auf. Höhere Anzahlen an betrachteten Startzeiten können in weiteren Algorithmen erforderlich sein, um Teilreservierungen fairer zu plazieren oder um optimalere Co-Reservierungsvarianten durchzuführen. Eine Vorabprüfung der Ressourcenverfügbarkeit ist deshalb für fortgeschrittene Algorithmen sinnvoll.

4.3 Algorithmus 2: CPU-Zähl-Algorithmus

Algorithmus 2 gleicht dem ersten Algorithmus bis auf den Schritt der Teilreservierungsvariantenbestimmung, in dem nun die Ressourcenverfügbarkeit geprüft wird. In Abschnitt 4.3.1 wird dieser Schritt für Algorithmus 2 beschrieben. Erkenntnisse aus den Untersuchungen zu Algorithmus 2 werden in Abschnitt 4.3.2 kurz zusammengefaßt.

4.3.1 Bestimmung von Teilreservierungsvarianten

In Algorithmus 2 wird nur noch eine Teilmenge der Teilreservierungsvarianten von Algorithmus 1 bei der Variantenbaumerstellung zugelassen. Die Teilreservierungsvarianten werden nun durch einen Statusanfrageschritt für jeden Cluster-Kandidaten individuell bestimmt (Schritt 4 Abbildung 5). Anschließend werden die Teilreservierungsvarianten zum Globalen Reservierungsdienst übertragen (Schritt 7 Abbildung 5). Bei jedem Cluster-Kandidaten werden die Teilreservierungsvarianten einem zusätzlichen Test unterzogen. Von den in Algorithmus 1 betrachteten Teilreservierungsvarianten werden diejenigen im Vorfeld verworfen, bei denen nicht genügend freie CPUs für die angeforderten Ressourcen zur Verfügung stehen. Im späteren Reservierungsschritt können die im Vorfeld bereits sehr wahrscheinlich nicht zu reservierenden Teilreservierungsvarianten entfallen. Teilreservierungen sollten nun nur noch wegen selten eintretenden Ereignissen fehlschlagen. Diese sind zum Beispiel: nebenläufige Reservierungsanfragen für einander überlappende Zeitintervalle, Veränderungen am Job-Workload zwischen Status- und Reservierungsanfrage, Netzwerkverbindungsprobleme oder Absturz des Reservierungsdienstes.
Der Abzählalgorithmus ist in Listing 3 dargestellt. Bei einer Statusanfrage werden zu allen Reservierungsstart- und Endereignissen (inklusive den Zeiten der zu bildenden Teilreservierungsvariante) während der Zeitspanne zwischen Start- und Endzeit der Variante *start* bis *start + dauer* die belegten CPUs gezählt. Der größte Wert aller gezählten CPUs entscheidet, ob die Teilreservierungsvariante zur Startzeit *start* berücksichtigt wird (Zeile 17).

```
1  --   Anfrage = (f_start, deadline, dauer, res)
2  type Anfrage = (Zeit, Zeit, Zeit, [Int])
3  --   Ressource = (cid, max_cpus)      Ressourcekandidaten vom Informationsdienst
4  type Ressource = (ID, Int)
5
6  tr :: [Ressource] -> Anfrage -> [TR]
7  tr cluster (f_start, dauer, deadline, res)
8  tr = [(cid, res_id, start)|
9        -- cid, res_id, start Kombinationen aufspannen wie in Algorithmus 1
10       (cid, max_cpus) <- cluster,
11       res_id <- [0 .. length res],
12       start <- startzeiten
13       -- zu start belegte CPUs bestimmen
14       belegt_cpus <- statusanfrage cid start dauer,
15
16       -- Filter TR, wenn zu wenig CPUs verfügbar
17       max_cpus - belegt_cpus > (sort_desc res)!!res_id
18       ]
19
20 --   Reservierungtabelle   [(start, ende, Anzahl CPUs)]
21 type Reservierungtabelle = [(Zeit, Zeit, Int)]
22 tabelle = Reservierungstabelle des Batchsystem-Schedulers
23 zeiten = Start und Endzeiten aller Reservierungen
24
25 -- LRS Statusanfrage
26 -- während der Zeitspanne [start .. start+dauer] maximal belegte CPUs
27 statusanfrage cid start dauer = max_list (map (zaehl_cpus tabelle) ereignisse)
28     where ereignisse = zeiten ++ [start, start+dauer]
29
30 -- Summe aller belegten CPUs zum Zeitpunkt t
31 zaehl_cpus tabelle t = sum [cpus |
32                             (start,ende,cpus) <- tabelle,
33                             start <= t, ende > t]
```

Listing 3: Teilreservierungsvariantenfilterung in Algorithmus 2

4.3.2 Analyse

In der Untersuchung in Kapitel 6.5 wurden die Reservierungseffizienz (Teilreservierungsfehlschläge und -zurücksetzungen) von Simulationen mit Algorithmus 1 und 2 verglichen.
Unter der Annahme, daß die Benutzer sich bei der kleinsten Vorausbuchzeit an der Dauer und dem CPU-Bedarf ihrer Co-Reservierungsanfragen orientieren oder alternativ große Zeiträume für die Startzeiten ihrer Anfragen angeben, genügt die Reservierung der Anfragen mit Algorithmus 1.
Teilreservierungsfehlschläge ereignen sich bei Anfragen mit dem Algorithmus 1 besonders häufig, wenn wenige Cluster nur zu frühen Startzeiten angefragt werden, da Ressourcen dann sowohl durch Jobs als auch durch andere Teilreservierungen belegt sein können. Je mehr Startzeiten pro Cluster-Kandidat bei den Bearbeitungen berücksichtigt werden, desto mehr lohnt die Reservierung mit Algorithmus 2. Die einmalige Statusanfrage verhindert somit viele potentiell fehlschlagende Reservierungsanfragen bei ihnen. Bei Anfragen mit Algorithmus 2 konnte in den Simulationen fast immer die jeweils erste betrachtete Co-Reservierungsvariante reserviert werden, denn die Ressourcenverfügbarkeit änderte sich zwischen den Status-

und den folgenden Teilreservierungsanfragen nur selten.

4.4 Algorithmus 3: Faire Reservierungen

Die ersten beiden Algorithmen berücksichtigen zum Zeitpunkt der Reservierungsanfrage nicht die gegenwärtige Last, die ein Batchsystem zu verarbeiten hat. In der Zeit unmittelbar nach der Anfrage können viele Jobs warten, bis für sie Ressourcen frei werden. Die ersten beiden Algorithmen nehmen Teilreservierungen auch zu diesen Zeiten vor, so daß diese Jobs unter Umständen noch länger warten müssen.
In Algorithmus 3 wird, ebenfalls wie in Algorithmus 2, eine Statusanfrage des GRS bei den LRS vorgenommen. Um die Fairness zu wartenden Jobs zu gewährleisten, wird zusätzlich bei der Prüfung der Teilreservierungsvarianten während der Statusanfrage die Last aller eingeplanten Jobs berücksichtigt. Bei zu hoher Last werden durch den LRS ebenfalls Teilreservierungsvarianten verworfen. Die Höhe der Last wird mit der *Auftragsbestandszeit* gemessen. Die Definition und dieser Zeit wird in Abschnitt 4.4.1 gegeben. In Abschnitt 4.4.2 wird beschrieben, welche Probleme bei der Berechnung der *Auftragsbestandszeit* auftreten. In Abschnitt 4.4.3 wird eine Forschungsarbeit vorgestellt, die zur Lösung dieser Probleme beiträgt. In Abschnitt 4.4.4 wird schließlich gezeigt wie die Auftragsbestandszeit von den LRS berechnet wird. Die Ergebnisse der Untersuchungen zu Algorithmus 3 werden in Abschnitt 4.4.5 kurz zusammengefaßt.

4.4.1 Auftragsbestandszeit

In dieser Arbeit definiert die Auftragsbestandszeit die Zeit, die für die Verarbeitung des Auftragsbestands in einem Batchsystem notwendig ist. Der Auftragsbestand bezeichnet alle zum Anfragezeitpunkt der Statusanfrage bereits angesetzten wartenden und laufenden Jobs. Für den Reservierungsdienst hat die Auftragsbestandszeit eine zusätzliche Bedeutung. Sie gibt an, ab wann die Startzeit einer Teilreservierungsvariante als zu wartenden Jobs fair betrachtet wird. Eine Teilreservierung sollte frühestens zu einer Zeit starten dürfen, die einem statt ihr geplanten Job entspricht. Deshalb werden in Algorithmus 3 nur Teilreservierungsvarianten zugelassen, deren Startzeiten nach Ablauf der Auftragsbestandszeit liegen. Die beiden ersten Algorithmen umgehen dieses Fairness-Prinzip und können Teilreservierungen somit unfair vor wartende Jobs des Auftragsbestands einreihen. Die Verarbeitung einer Teilreservierung vor wartenden Jobs des Auftragsbestands könnte dazu führen, daß nicht genügend Ressourcen für diese verbleiben und sie zusätzlich verzögert werden.

4.4.2 WallClock-Zeit Problematik

Die Auftragsbestandszeit wird aus den Laufzeiten der Jobs des Auftragsbestandes und deren CPU-Anforderungen berechnet. Die Berechnung ist nicht trivial, denn die Laufzeiten der Jobs sind erst zu einem späteren Zeitpunkt rückblickend bekannt. Mit einer Historie von gelaufenen Jobs und dem WallClock-Zeit Parameter in einer Job-Beschreibung läßt sich die Laufzeit jedoch abschätzen. Die WallClock-Zeit gibt an wie lange ein Job maximal laufen darf. Sie sollte der ungefähren Ausführungszeit des Jobs gleichen. Lee et al. [11] untersuchten das Verhalten von Batchsystem-Benutzern hinsichtlich der WallClock-Genauigkeiten. Zunächst stellten sie fest, daß Benutzer von Batchsystemen oft zu hohe WallClock-Zeiten für ihre Jobs angaben. In einem Beispiel aus [12] berichten Lee et al. davon, daß 35 Prozent aller Jobs eine

Genauigkeit von weniger als 10 Prozent besitzen. In der Regel werden Jobs abgebrochen, wenn die angegebene WallClock-Zeit überschritten wird. Deshalb sind die meisten Benutzer bei ihrer Abschätzung sehr vorsichtig. In einem Experiment versuchten Lee et al. Benutzern Prämien für gute Abschätzungen anzubieten und versicherten ihnen zusätzlich, daß Jobs nicht abgebrochen werden. Die meisten Benutzer schätzten jedoch trotz der Zusagen die WallClock-Zeiten weiterhin unverändert ein.
In der Auftragsbestandszeitberechnung werden die Laufzeiten der laufenden und wartenden Jobs mit einer WallClock-Genauigkeit verrechnet. Sie ist definiert als Verhältnis der Laufzeit zur WallClock-Zeit. Die Genauigkeit der WallClock-Zeiten bestimmt somit die Qualität der Auftragsbestandszeitvorhersage.
In Reservierungszenarien bewirken Abweichungen der errechneten Auftragsbestandszeit von der tatsächlichen Auftragsbestandszeit Nachteile für lokale Jobs und Co-Reservierungen. Wenn die Auftragsbestandszeit zu pessimistisch abgeschätzt wird, können Reservierungsvarianten nicht durchgeführt werden, weil sie wegen der Job-Fairness nicht als Variante in Betracht gezogen wurden. Bei zu optimistischer Abschätzung werden Teilreservierungen zugelassen, die in unfairer Weise vor wartenden Jobs ausgeführt werden könnten.

4.4.3 Verwandte Arbeit: Laufzeitabschätzung von Jobs

Warren Smith et al. [13] beschreiben wie Job-Laufzeiten mit Historien abgeschätzt werden können. Je nach dem welche Informationen ein Workload beinhaltet, schlagen sie eine Unterteilung der Jobs in verschiedene Kategorien vor. Diese Kategorien partitionieren den Workload, z.B. kann der Workload durch den Warteschlangennamen geteilt werden oder durch den Benutzernamen. Die Laufzeit eines Jobs kann nun z.B. durch den Mittelwert der Laufzeit der Jobs seiner zugehörigen Kategorie abgeschätzt werden.

4.4.4 Auftragsbestandszeitberechnung

Batchsystem Scheduler, wie der Maui Scheduler [10], verfügen über Schnittstellen, die Auskunft über die durchschnittliche WallClock-Genauigkeit von Jobs erteilen. Listing 4 zeigt wie die Auftragsbestandszeit für die Fairnessberechnung einer Teilreservierungsvariante abgeschätzt wird. Als generelle Vorgehensweise die Auftragsbestandszeit zu ermitteln, summiert man die mit der WallClock-Genauigkeit gewichteten CPU-Sekunden aller lokalen Jobs und teilt diese durch die Gesamtanzahl aller CPUs des Clusters.
Ressourcenbelegungen durch Reservierungen werden mit in die Auftragsbestandszeitberechnung integriert. Sie können ebenfalls die Verarbeitung des Auftragsbestands beeinflussen, wenn sie zeitgleich mit Jobs des Auftragsbestands eingeplant sind. Zu klären bleibt, für welche Reservierungen dies der Fall ist. Es bietet sich an die Startzeit *tr_start* der auf Fairness zu prüfenden Teilreservierungsvariante als Schwellwert zu benutzen. Die Überlegung dabei ist, daß Reservierungen, die später als *tr_start* geschedult sind, bei ihrer Anfrage die Chance haben eine frühere Startzeit für sich zu beantragen und sie somit überholt werden dürfen. Die CPU-Sekunden der Teile von Co-Reservierungen werden nicht gewichtet, weil ein Klient eine Teilreservierung z.B. auch für mehrere Jobs benutzen kann und letztlich die Nutzung des Co-Reservierungsdienstes nicht genau bekannt ist.
In der beschriebenen Auftragsbestandszeitberechnung wird angenommen, daß zu jeder Zeit alle Knoten durch Jobs belegt sind. Dies ist in der Praxis nicht der Fall. Der Ablaufplan

```
1 abzeit tr_start =
2   --CPU-Sekunden von allen lokalen Batch-Jobs
3   (sum [wc_acc * wc * ncpus| (wc,ncpus)<-lokale_jobs] +
4   --CPU-Sekunden von anderen betrachteten Teilreservierungen
5   sum [wc * ncpus | (dauer,cpus, start)<-tr_reservierungen, start < tr_start]) /
6   totalcpus
```

Listing 4: Auftragsbestandszeitberechnung in Algorithmus 3

enthält Lücken, die von wartenden Jobs nicht benutzt werden können, weil ihre Laufzeit zu groß für die Lücke ist, oder sie zu viele CPUs anfordern. Es wird darauf verzichtet diese Lücken zu berücksichtigen, weil deren Berechnung die Nachbildung des Batchsystem-Schedulers erfordert.

4.4.5 Analyse

Die Untersuchungen für Algorithmus 3 werden im Detail in den Kapiteln 6.6 und 6.7 vorgestellt. Die wichtigsten Erkenntnisse der Untersuchungen werden nun zusammengefaßt. Bei dem Vergleich zwischen Simulationen mit Algorithmus 2 und Algorithmus 3 fiel auf, daß die Auftragsbestandszeitberücksichtigung sowohl positive als auch negative Effekte auf Jobs und Reservierungen hatten.
Bei der Untersuchung der Kriterien waren folgende Ergebnisse festzustellen:

Co-Reservierungsfehlschlagrate

Für Co-Reservierungsanfragen ist bei Reservierung mit Algorithmus 3 anstatt Algorithmus 2 mit einem Anstieg der Co-Reservierungsfehlschlagrate zu rechnen. In den zuvor betrachteten Algorithmen wurden Reservierungsanfragen nur verworfen, wenn auch tatsächlich die CPUs zu den angefragten Zeiten in Batchsystemen belegt waren. Wenn bei der Bearbeitung einer Co-Reservierungsanfrage mit Algorithmus 3 in genügend Batchsystemen die Aufragsbestandszeit auch für die späteste Startzeit unterschritten wird, dann werden keine Co-Reservierungsvarianten gebildet und sie schlägt fehl.

Verteilung der Reservierungslast

Bei Algorithmus 3 können neben höheren Co-Reservierungsfehlschlagraten auch Lastbalancierungseffekte auftreten. Für jeden Cluster werden, je nach Auftragsbestandszeit, unterschiedlich viele Teilreservierungsvarianten zu einer Anfrage gebildet. Die Durchführung einer Co-Reservierungsvariante mit einem Cluster, der niedrige Auftragsbestandszeiten hat, ist daher wahrscheinlicher. Es kann daher vorkommen, daß die Reservierungslast von Batchsystemen mit hohen Auftragsbestandszeiten zu Batchsystemen mit geringen Auftragsbestandszeiten übergehen.

Durchschnittliche Auslastung der Batchsysteme

Batchsysteme, die weniger stark ausgelastet sind, ziehen, wenn sie gleichzeitig niedrigere Auftragsbestandszeiten haben, mehr Reservierungslast an. Dies kann zu einer Angleichung der durchschnittlichen Auslastung der Batchsysteme führen. Je nach dem in welchem Batchsystem sich Jobs befinden, können sie längere oder kürzere Wartezeiten bei der ungleichen Verteilung der Reservierungslast haben.

Fairness zu Jobs

Im Durchschnitt warten bei Algorithmus 3 zum Zeitpunkt von Teilreservierungsstarts

weniger Jobs, die bereits zur Submit-Zeit von Teilreservierungen warteten, als bei Algorithmus 2. Beliebig unfaire Durchführungen von Reservierungen, zum Beispiel direkt nachdem genügend laufende Jobs fertig werden, sind aufgrund der Auftragsbestandszeitgrenze nicht möglich. Eine Fairnessgarantie kann Algorithmus 3 jedoch nicht bieten. Je größer die Vorausbuchzeiten der Anfragen sind, desto schwieriger läßt sich die gewünschte Fairness-Semantik aufrechterhalten. Jobs werden zwischen Teilreservierungs-Submit und -start neu eingeplant und können die bei Submit wartenden Jobs zusätzlich verzögern. Sich in den Batchsystemen verändernde WallClock-Ungenauigkeiten haben ebenfalls Einfluß auf den Grad der Fairness von Co-Reservierungsanfragen zu lokalen Jobs. Wenn lokale Benutzer, die oft ungenaue Laufzeitabschätzungen für ihre Jobs liefern, beginnen genauere Abschätzungen zu geben, dann kann Algorithmus 3 dies nicht sofort durch eine nötige höhere Auftragsbestandszeit berücksichtigen. Die Ursache liegt in der Bestimmung der WallClock-Ungenauigkeit durch die Analyse einer Historie eines längeren Zeitraums.
Es kann ebenfalls vorkommen, daß bei Algorithmus 3 Co-Reservierungsanfragen unfairerweise verworfen werden. Es wurde angenommen, daß die Ressourcen für wartende Jobs vor einer Belegung durch Teilreservierungen geschützt werden müssen. Durch Algorithmus 3 kann nicht mit Sicherheit vorhersagt werden, ob die nicht berücksichtigten Teilreservierungsvarianten tatsächlich wartende Jobs behindert hätten. Oft bleiben Ressourcen ungenutzt, weil wartende Batch-Jobs zu hohe Ressourcenanforderungen haben, um noch freie Ressourcen zu verwenden. Aufgrund dieser Erkenntnis wird in Abschnitt 4.5 noch eine Optimierung zu Algorithmus 3 vorgestellt.

4.5 Algorithmus 3: Optimierung durch Backfill-Reservierungen

In diesem Abschnitt wird eine Optimierung zu Algorithmus 3 vorgestellt. Bei der Analyse von Simulationen mit Algorithmus 3 (Kapitel 6.6) fiel auf, daß viele Co-Reservierungsanfragen mit frühen Startzeiten wegen der Auftragsbestandszeitbedingung unnötigerweise fehlschlugen.
Batchsysteme hatten oft über längere Zeiten freie CPUs, derer sich kein wartender Job bediente. Entweder forderten wartende Jobs zu diesen Zeiten zu viele CPUs an oder ihre WallClock-Zeiten waren zu groß, um diese Lücken zu schließen. Kleine Teilreservierungen hätten unter Umständen diese Lücken auffüllen können, ohne daß wartende Jobs behindert worden wären. Die Optimierung besteht darin einen Teil der bei der LRS Statusanfrage unnötigerweise verworfenen Teilreservierungsvarianten doch zuzulassen (siehe Abschnitt 4.4). In Abschnitt 4.5.1 wird vorgestellt wie die Teilreservierungsvarianten im optimierten Algorithmus gebildet werden. Anschließend wird in Abschnitt 4.5.2 eine kurze Zusammenfassung der Untersuchungen zum optimierten Algorithmus gegeben.

4.5.1 Bestimmung von Teilreservierungsvarianten

Die Zulassung von Teilreservierungen zu frühen Startzeiten ist vergleichbar mit dem Backfilling von Jobs im normalen Batchsystem-Betrieb. Batchsystem-Scheduler lassen niedriger priorisierte Jobs außerhalb des regulären Planes vor höher priorisierten starten, wenn diese sonst ungenutzte Ressourcen verwenden. Job-Backfilling erhöht damit in der Regel die Auslastung des Batchsystems ohne höher priorisierte Jobs zu benachteiligen.

In Abbildung 8 ist ein Batchsystem-Schedule zu sehen, in dem die vorgeschlagene Optimierung verwendet werden kann. Es sei *t_anfrage* die aktuelle Zeit in einem Batchsystem, zu der die Statusanfrage des GRS vom LRS bearbeitet wird, *t_ende* die Endzeit der zu prüfenden Teilreservierungsvariante und *t_sched* die Zeit des nächsten Schedulingereignisses.
Teilreservierungsvarianten werden von der Auftragsbestandszeitfilterung ausgeschlossen, wenn *t_ende* kleiner ist als *t_sched*. Somit ist es für den Scheduler möglich die Teilreservierung als „Backfill-Reservierung“ einzuplanen, wenn er dies für alle anderen wartenden Jobs nicht tun konnte. Für Teilreservierungsvarianten, deren Startzeit nach *t_sched* liegen, gilt wieder das Auftragsbestandszeitkriterium. Beim nächsten Scheduling könnte ein wartender Job als Backfill-Job die Lücke im Schedule nutzen und wäre, weil er vor Submit der Teilreservierungsanfrage schon wartete, auch dazu berechtigt. Eine exakte Berechnung des Schedules bis zu einer bestimmten Zeit erfordert die exakte Kenntnis über die Konfiguration des Batchsystem-Schedulers. Außerdem kann das Problem zudem auf das als NP-vollständig geltende „zweidimensionale Bin-Packing-Problem“ reduziert werden [14, 4]. Deshalb wird darauf verzichtet Zeiten nach *t_sched* als Kandidaten für Teilreservierungsvarianten zu prüfen.

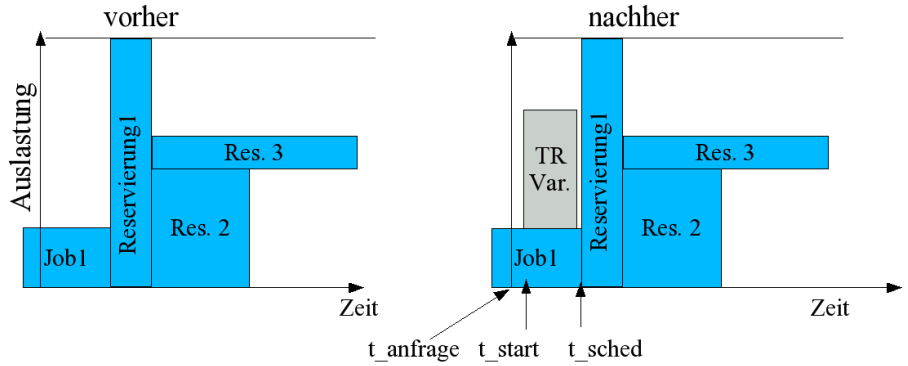


Abbildung 8: Backfill Reservierungen - Optimierung Algorithmus 3

4.5.2 Analyse

Untersuchungen zum Backfill-Algorithmus werden in Kapitel 6.7 vorgestellt. Es wurde festgestellt, daß Co-Reservierungsanfragen durch Backfill-Reservierungen durchgeführt werden konnten, die in vergleichbaren Simulationen mit Algorithmus 3 fehlschlugen. Bei Algorithmus 3 undurchführbare Anfragen konnten Backfill-startzeiten nutzen um Teilreservierungen trotz Mißachtung der Auftragsbestandszeitbedingung durchzuführen. Insbesondere Batchsysteme mit überwiegend größeren Jobs ließen mehr Teilreservierungsvarianten zu, da bei diesen, auch falls Jobs warteten, bei Algorithmus 3 Ressourcen öfter ungenutzt blieben.

4.6 Algorithmus 4: Startzeitoptimierung

Algorithmus 4 ist eine Modifikation zum Backfill-Algorithmus, in der die frühesten Co-Reservierungsvarianten zuerst im Reservierungsschritt betrachtet werden. Dazu ist eine Sortierung des Variantenbaums nötig, die in Abschnitt 4.6.1 vorgestellt wird. Die Ergebnisse der Untersuchungen zu Algorithmus 4 werden in Abschnitt 4.6.2 in zusammengefaßter Form vorgestellt.

```
1 -- Algorithmus 1-3 - unsortiert
2 baum :: Anfrage -> [TR] -> Variantenbaum
3 baum a tr = ekind (-1,-1,-1) (randomize tr)
4
5 -- Algorithmus 4 - Quicksort der Teilreservierungsvarianten nach Startzeit
6 baum :: Anfrage -> [TR] -> Variantenbaum
7 baum a tr = ekind (-1,-1,-1) startsort (randomize tr )
```

Listing 5: Startsortierung der Co-Reservierungsvarianten in Algorithmus 4, Änderung zu Listing 2

4.6.1 Bestimmung von Co-Reservierungsvarianten

Um eine Startzeitsortierung der Co-Reservierungsvarianten zu erreichen, ist eine kleine Änderung im Schritt des Variantenbaumaufbaus nötig (siehe Abschnitt 4.2.2). In den ersten drei Algorithmen ist die erste Teilreservierung der Co-Reservierungsvarianten im Baum nicht nach Startzeit sortiert, sondern zufällig angeordnet. Wenn die Teilreservierungsvarianten für die erste Ressource in startzeitsortierter Reihenfolge in den Baum eingefügt werden, dann sind alle auf ihnen aufbauenden Co-Reservierungsvarianten ebenfalls nach Startzeit sortiert, weil die Kinder eines Variantenbaumknotens die Startzeit des Vaters erben (Listing 2, Zeile 28). Die einmalige Sortierung des Variantenbaums hält auch zu jeder Zeit im anschließenden Reservierungsschritt stand. Sollte eine Teilreservierung für die erste Ressource fehlschlagen, dann wird die nächste Teilreservierungsvariante aus der Kinderliste der Wurzel ausgewählt. Wegen der Sortierung dieser Liste ist dies eine Teilreservierung der nächst späteren Co-Reservierungsvariante. Schlägt eine Co-Reservierungsvariante bei einer späteren Teilreservierung fehl, dann hat die nächste Variante, wegen der gleichen ersten Teilreservierung, die gleiche Startzeit. Listing 5 zeigt die vorgenommene Änderung.

4.6.2 Analyse

Die Untersuchungen zu Algorithmus 4 werden in Kapitel 6.8 vorgenommen. In Simulationen wurde analysiert wie früh mit diesem Algorithmus, im Vergleich zu Startzeit optimierenden Varianten von Algorithmus 2 und 3, Co-Reservierungen vorgenommen werden konnten. Die Modifikation von Algorithmus 2, in der Fairness zu Jobs nicht beachtet wurde, reservierte im Vergleich zu den anderen Algorithmen die frühesten Co-Reservierungsvarianten, die Modifikation mit Algorithmus 3 die spätesten. Entscheidend für die Auswahl früher Co-Reservierungsvarianten war für Algorithmus 4 die Möglichkeit Backfill-Reservierungen zu benutzen. Algorithmus 4 konnte besonders für kurze Anfragen mit geringem Ressourcenbedarf Backfill-Reservierungen benutzen, um früheste Startzeiten zu reservieren. Je höher die Anzahl der zur Bearbeitung einer Anfrage zur Verfügung stehenden Cluster-Kandidaten und je niedriger die Anzahl der Teilreservierungen war, desto häufiger konnte durch Backfill-Reservierungen die früheste Startzeit reserviert werden.

4.7 Algorithmus 5: Kombinierte Sortierkriterien - Startzeit/Kosten

Der nächste Algorithmus soll bei der Durchführung von Co-Reservierungsanfragen mehrere Sortierkriterien berücksichtigen. In Abschnitt 4.7.1 werden verwandte Arbeiten vorgestellt

und anschließend der eigene Algorithmus.

4.7.1 Verwandte Arbeiten

Es werden nun zwei verwandte Arbeiten vorgestellt.

Deadline-Budget-Algorithmus von Buyya et. al:
Buyya et al. erforschten wie bei der Grid Ressourcenverwaltung ökonomische Aspekte berücksichtigt werden können [15]. Sie stellten einen Algorithmus vor, in dem Deadline- und Budget-Beschränkungen beim Scheduling von Grid-Jobs berücksichtigt werden [16].
In ihrem Algorithmus ist jeder Ressource ein unterschiedlicher aber fester Tarif zugeordnet. Eine Anwendung besteht in ihrem Modell aus einer Liste von Jobs, die – so lange das Budget für deren Ausführung ausreicht – verarbeitet werden müssen.
Abbildung 9 zeigt den Ablauf der Ressourcenbelegung für eine Liste von Jobs.

Anfrage
In der Anfrage definiert der Klient eine Budget-Obergrenze und eine Liste an zu verarbeitenden Jobs.

Budgetbegrenzung
Die Jobs werden von einem Broker den Ressourcen zugewiesen, bis das Budget aufgebraucht ist oder alle Jobs verarbeitet wurden.

Ressource-Konfiguration
Die Kosten für einen Job werden individuell für jede Ressource über einen Stundensatz für die vom Job belegten CPUs und die benötigte Zeit berechnet.

Scheduling-Algorithmus
Jede Ressource liefert dem Scheduling-Algorithmus drei Informationen:

- Benchmark-Werte einer CPU
- früheste mögliche,geschätzte Startzeit
- Preis pro CPU-Stunde

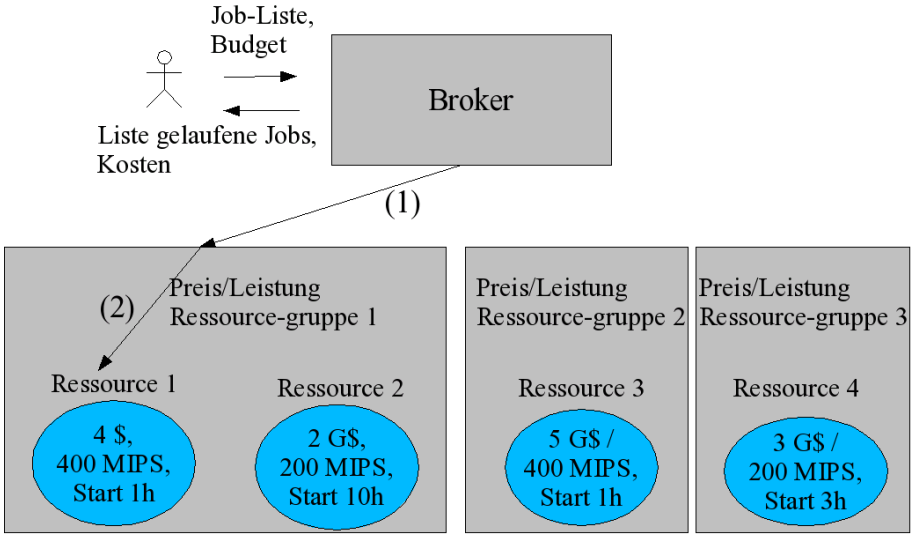


Abbildung 9: Kombinierte Kosten-Startzeit Optimierung für Grid-Jobs nach Buyya et al. [16]

Entscheidend für die Auswahlbewertung ist für den Scheduling-Algorithmus der Stundensatz im Zusammenhang mit der Performance einer CPU. Bei einer besseren Performance ist auch ein höherer Stundensatz für die Ressource akzeptabel, denn der Job hat eine entsprechend kürzere Laufzeit. Die Auswahl einer Ressource für den Job wird in zwei Schritten vorgenommen. Zunächst werden Ressourcen in Preis-Leistungs-Gruppen zusammengefaßt, dann wird innerhalb einer Preis-Leistungs-Gruppe die Ressource mit frühester Startzeit ausgewählt. Wenn die Deadline nicht innerhalb einer Preis-Leistungsgruppe einzuhalten ist, wird aus der nächst teureren Gruppe ausgewählt.

Reservierungsdienst für flexible Reservierungsanfragen von Röblitz et. al:
Röblitz et al. beschreiben einen Reservierungsdienst, mit dem flexible Reservierungsanfragen für eine Ressource durchgeführt werden können [17]. Reservierungsvarianten werden in mehreren Stufen sortiert und gruppiert, ähnlich dem obigen Modell. Eventuell variieren bei diesem Ansatz die Werte auf den nachgelagerten Sortierstufen stark von den optimalen. Wenn die Werte zweier Kriterien im Widerspruch zueinander stehen ist eine kombinierte Optimierung nicht möglich. Beispielsweise ist die Reservierung von frühen und günstigen Reservierungsvarianten nicht möglich, falls frühe Varianten tendenziell eher die teureren sind und günstige eher spät. Im nächsten Abschnitt wird Algorithmus 5 vorgestellt, dessen Design-Ziel Durchführungen von Co-Reservierungen mit dem bestem Trade-Off von mehreren Optimierungkriterien ist.

4.7.2 Algorithmus Modell

Der fünfte Algorithmus soll die Reservierung von Co-Reservierungsvarianten mit dem besten Trade-Off mehrerer Kriterien ermöglichen. Abbildung 10 veranschaulicht den Unterschied zu dem mehrstufigen Sortierungsmodell des vorigen Abschnitts. Teurere Co-Reservierungsvarianten werden bei dieser Herangehensweise bei einer Tiefensuche eher gefunden, wenn diese frühere Startzeiten ermöglichen. Eine kombinierte Kosten-Startzeit-Optimierung für Co-Reservierungsanfragen wird nun als Beispiel für Algorithmus 5 vorgestellt.

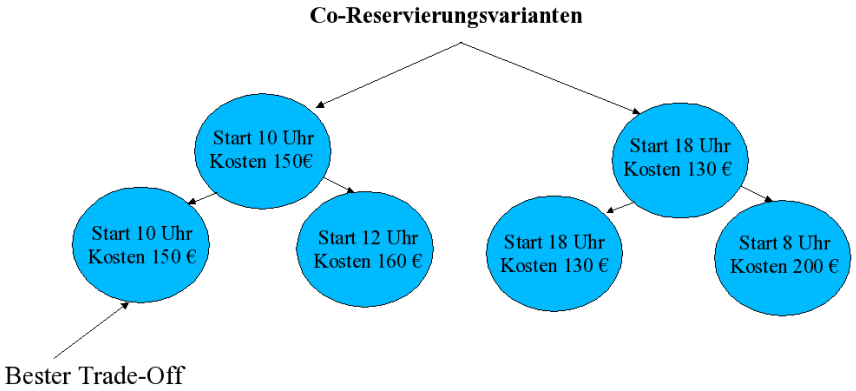


Abbildung 10: Sortierung nach bestem Trade-Off in Algorithmus 5

Ein- und Ausgabe des Algorithmus:
Eingabe des Algorithmus ist eine wie in den bisherigen Algorithmen aufgebaute Reservierungsanfrage (siehe Tabelle 1), ein Kostenfaktor k sowie ein Startzeitfaktor s. Die Faktoren k und s sind reelle Zahlen zwischen 0 und 1, mit denen Kosten- und Startzeitteilwert von Co-

Reservierungsvarianten gewichtet werden können. Die Optimierung einer Co-Reservierungsvariante nach Algorithmus 4 ist als Randfall *k=0, s=1* in Algorithmus 5 enthalten.
Das Ergebnis von Algorithmus 5 ist die Reservierung einer Co-Reservierungsvariante mit dem zusätzlichen Attribut für die zugeordneten Reservierungsgesamtkosten, es ist ein Tripel aus Startzeit, Gesamtkosten und der Ressourcenabbildung.

Algorithmusablauf:
Im Algorithmus wird nach der allgemeinen Vorgehensweise, die in Abschnitt 4.1 beschrieben wurde, vorgegangen.
Veränderungen bei den einzelnen Arbeitsschritten sind in Algorithmus 5 die folgenden:

Statusanfrage (Schritt 4 in Abbildung 5)
Die Teilkosten der Teilreservierungsvarianten werden im Statusanfrageschritt bei den lokalen Reservierungsdiensten erfragt. Ein Algorithmus für die Berechnung der Teilkosten wird in Abschnitt 4.7.3 vorgestellt.

Co-Reservierungsvarianten (Schritt 7 in Abbildung 5)
Aus den Teilreservierungsvarianten wird ein Variantenbaum erzeugt (siehe Abschnitt 4.2.2). Beim Aufbau des Variantenbaums wird in zwei Schritten vorgegangen:

Erstellung eines unsortierten Baums
Analog zum trivialen Algorithmus werden die Teilreservierungsvarianten als Knoten in den Variantenbaum hinzugefügt. Die zuvor bestimmten Teilkosten werden als Attribut im Knoten gespeichert.

Sortierung
In Algorithmus 5 muß der gesamte Baum nach einem aus Startzeit und Gesamtkosten kombinierten Wert sortiert werden. Dazu wird die Co-Reservierungsvariante mit dem besten Trade-Off aus Startzeit und Gesamtkosten im Variantenbaum so plaziert, daß sie bei einer Tiefensuche als erstes gefunden wird. Der Sortieralgorithmus wird im Detail in Abschnitt 4.7.4 beschrieben.

Reservierungsschritt (Schritt 8 in Abbildung 5)
Beim Reservieren mit dem Tiefensucheverfahren muß in Algorithmus 5 eine Sonderbehandlung nach einem Teilreservierungsfehlschlag durchgeführt werden, um die Sortierung des Baumes wiederherzustellen. In Abschnitt 4.7.5 wird dieser Teil des Algorithmus im Detail beschrieben.

4.7.3 Bestimmung von Teilreservierungsvarianten - Teilkostenbestimmung

Die Teilreservierungskosten werden im Statusanfrageschritt des generellen Schemas (siehe Abbildung 5), in dem auch die Auftragsbestandszeit und die belegten CPUs ermittelt werden, bestimmt.
Anders als im Deadline-Budget-Algorithmus von Abschnitt 4.7.1 variieren im vorliegenden Modell die Kosten für Teilreservierungen je nach Tageszeit für die Ressourcen. Jeder lokale Reservierungsdienst verfügt über eine Kostentabelle (Listing 6). Für jeden Wochentag kann zu den verschiedenen Tageszeiten ein anderer Stundensatz definiert werden. Beispielsweise kann ein höherer Stundensatz wochentags während der regulären Arbeitszeit vereinbart werden. Sei zum Beispiel *tabelle = [(MON, 0, 8, 1.0), (MON, 8, 18, 1.5), (MON, 18, 24, 1.0), ...]* eine Stundensatztabelle, dann kostet eine Stunde Rechenzeit montags zwischen Mitternacht und 8 Uhr morgens eine Geldeinheit, tagsüber 1.5 Geldeinheiten und abends wieder eine

```
1 -- Kostentabelle   [(Tag, Start, Ende, Preis pro Stunde)]
2 type Kostentabelle [(Int, Int, Int, Float)]
```

Listing 6: Kostentabelle

```
1   teilkosten start dauer cpus = cpus * (sum stdl_stundensaetze)
2   where
3     stdl_stundensaetze = map (lookup_stundensatz) [st, st + 3600 .. (st + dauer)]
```

Listing 7: Kostentabelle

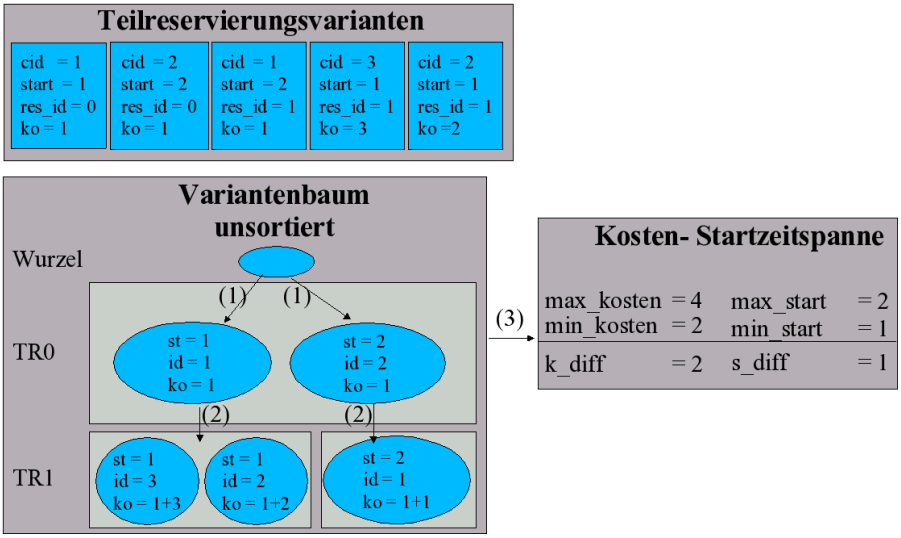
Geldeinheit. Mit einer einfachen Zählfunktion (Listing 7) werden die gesamten Teilkosten aus dem Produkt der benötigten CPUs und der Summe stündlich berechneter Stundensätze gebildet. Eine Teilreservierung für 10 CPUs zwischen 7:30 Uhr und 9 Uhr kostet montags im obigen Beispiel also $10 \cdot (1.0 + 1.5) = 25$ Geldeinheiten.

4.7.4 Erzeugung und Sortierung der Co-Reservierungsvarianten

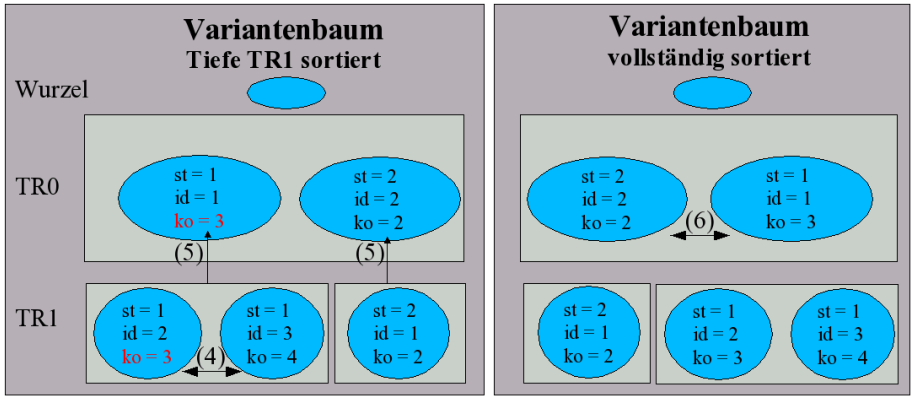
Aus der Liste der Teilreservierungsvarianten wird nun ein Baum für die Co-Reservierungsvarianten gebildet.
Der Baumaufbau wird in drei Schritten durchgeführt:

- Ein unsortierter Variantenbaum wird analog zu dem in Algorithmus 1 verwendeten erzeugt (siehe Abschnitt 4.2.2).
- Der Baum wird nach Maximal- und Minimalwerten der Sortierattribute analysiert.
- Eine Sortierung des Baumes wird vorgenommen.

In Abbildung 11(a) sind die ersten beiden Aufzählungspunkte dargestellt. Zuerst werden die Teilreservierungsvarianten für die erste Ressource (*res_id=0*) in den Baum eingefügt (1), anschließend rekursiv die jeweils zu den eingefügten Knoten passenden Teilreservierungsvarianten für die nächste Ressource (2). Die Teilreservierungsvarianten zur früheren Startzeit bilden den linken Ast, die anderen den rechten. Die Kosten der Elternknoten werden beim Baumaufbau an die Kinderknoten vererbt. Nach dem Aufbau des unsortierten Baumes haben die Werte für Gesamtkosten und Startzeiten in den Blättern potentiell unterschiedliche Spannweiten. Die Sortierung der Pfade nach dem besten Trade-Off der Startzeit- und Kostenwerte kann dann erst nach einer Normalisierung vorgenommen werden. Daher werden in Schritt (3) die maximalen und minimalen Werte für Startzeit und Gesamtkosten ermittelt. Aus den mit Kostenfaktor k gewichteten, normalisierten Gesamtkosten und der mit Startzeitfaktor s gewichteten, normalisierten Startzeit wird für jedes Blatt ein kombinierter Wert der zugehörigen Co-Reservierungsvariante berechnet. Der Baum aus Abbildung 11(b) entsteht, wenn der Kostenfaktor höher als der Startzeitfaktor bemessen ist, beispielsweise bei Kostenminimierung. In Schritt (4) erfolgt die Sortierung der Knotenlisten auf Ebene der Blätter. Die Gesamtkosten des Kindes mit bestem Wert werden zum Vaterknoten propagiert (5). Anschließend werden die Vaterknoten nach den propagierten Werten ihrer besten Kinder sortiert (6). Das Verfahren wird rekursiv fortgesetzt, bis die Wurzel erreicht ist.



(a) Erstellung des unsortierten Baums: Die Teilreservierungskosten werden auf tiefere Knoten vererbt (1,2). Anschließend werden Normierungsparmeter für Attribute bestimmt (3). Für jedes Blatt wird mit diesen aus Startzeit und Gesamtkosten ein kombinierter Wert berechnet.



(b) Rekursive „Bottom-Up“ Sortierung: Knoten werden auf Tiefe n nach bestem Wert sortiert (4). Der Beste propagiert seine Gesamtkosten an den Vaternknoten (5). Das Verfahren wird auf Tiefe n-1 fortgesetzt bis die Wurzel erreicht ist (6).

Abbildung 11: Mehrstufiger Variantenbaumaufbau in Algorithmus 5

Wertfunktion:
Ein Startzeit- oder Kostenteilwert beträgt 0, wenn alle Varianten gleiche Kosten oder Startzeiten haben. Für den Fall, daß Kosten und Startzeit von Co-Reservierungsvarianten tatsächlich variieren, gibt Gleichung 2 der Wert eines Knotens n bei der Sortierung an. Das Attribut *ko* eines Knotens enthält nach Schritt (4) die Gesamtkosten des Kindes mit niedrigstem Wert. Die *diff* Werte sind die in (3) analysierten Differenzbeträge, die jeweils durch den Abstand des maximalen vom minimalen Wert eines Attributs für die Co-Reservierungsvarianten berechnet wurden. Die Funktion *norm_kosten* gibt die zwischen 0 und 1 normierten Gesamtkosten der besten, von Knoten n aus möglichen, Co-Reservierungsvariante an. Die Funktion *norm_start* gibt analog die normierte Startzeit dieser Co-Reservierungsvariante an.

$$\begin{aligned} norm\_kosten(n) &= \frac{ko(n) - min\_kosten}{k\_diff} \\ norm\_start(n) &= \frac{st(n) - min\_start}{s\_diff} \end{aligned}$$

$$wert(n) = k \cdot norm\_kosten(n) + s \cdot norm\_start(n) \tag{2}$$

Mit dem Anfrageparametern k und s lassen sich individuell Kosten- und Startzeitteilwert des Knotenwertes steuern. Bei reiner Startzeitoptimierung mit $k = 0$ und $s = 1$ findet im Beispiel der Abbildungen 11(a) und 11(b) keine Umsortierung der Pfade statt. Bei $k < 0.5$ und $s = 1$ entspricht der linke Baum von Abbildung 11(b) bereits dem korrekt sortierten.

4.7.5 Reservierungsschritt

Der Reservierungsschritt ist in Algorithmus 5 dem der bisher betrachteten Algorithmen ähnlich. In Algorithmus 5 wird versucht weiterhin mit dem in Abschnitt 4.2.3 vorgestellten Tiefensucheverfahren zu reservieren.
Wenn keine Teilreservierungsfehlschläge auftreten, dann kann wie bei Algorithmus 1 die Co-Reservierungsvariante des ersten Pfades des Variantenbaums reserviert werden. Nach Konstruktion des Baumes sind darin die Knoten mit minimalen relativen Start- und Kostendifferenzen enthalten, also die Teilreservierungen der optimalen Co-Reservierungsvariante.
Nach einem Teilreservierungsfehlschlag muß in Algorithmus 5 geprüft werden, ob die partiell durchgeführte Co-Reservierungsvariante noch die beste mögliche ist. Der Abstiegsschritt der Tiefensuche wird dazu gegenüber den vorherigen Algorithmen leicht verändert, um dann nicht den Baum absuchen zu müssen.
Abstieg bei Tiefensuche:

- Falls alternative Pfade vom letzten reservierten Knoten *aktuell* möglich
  - aktualisiere Gesamtkosten *ko* von *aktuell* durch *ko* des besten alternativen Pfades
  - reserviere nächste Teilreservierungsvariante des besten Pfades
- sonst
  - reserviere nächste Teilreservierungsvariante des einzigen Pfades (*aktuell* muß bei Fehlschlag der nächsten Teilreservierung ebenfalls zurückgesetzt werden)

Beispiel ohne Umsortierung:
Abbildung 12 zeigt die Kostenstruktur eines Variantenbaums für eine aus zwei Teilen bestehende Co-Reservierung bei reiner Kostenoptimierung. Der Baum in Bild 1 der Abbildung ist

die Ausgangsposition. Nach erfolgreicher Reservierung von TR0 wird im Schritt von Bild 2 die Kosten der Wurzel aktualisiert und analog im Schritt von Bild 3 nach Erfolg von TR1 die Kosten des TR0 Knotens. Die Gesamtkosten der Variante sind nach Co-Reservierungserfolg weiterhin in TR1 enthalten (Bild 3).

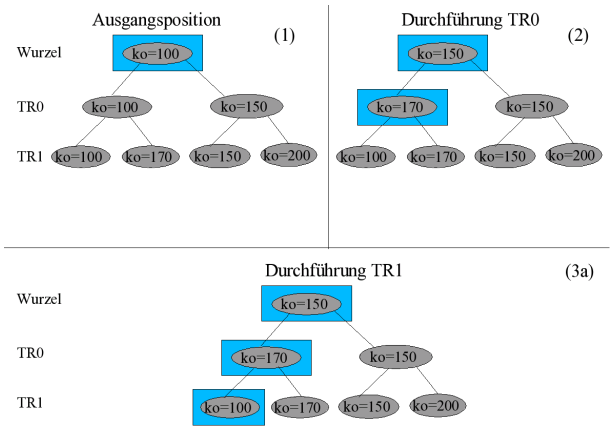


Abbildung 12: Kostenstruktur des Variantenbaums im Reservierungsschritt bei reiner Kostenoptimierung

Beispiel mit Umsortierung:
Ein Beispiel, in dem eine Umsortierung erforderlich ist, zeigt Abbildung 13. Ließe sich zum Beispiel die letzte Teilreservierung der 100 Geldeinheiten teuren Variante nicht reservieren, dann findet das Tiefensucheverfahren die 170 Geldeinheiten vor der 150 Geldeinheiten teuren Variante. Bei der Einsortierung wird auf die im Abstieg aktualisierten Gesamtkosten der nächsten Variante zurückgegriffen.

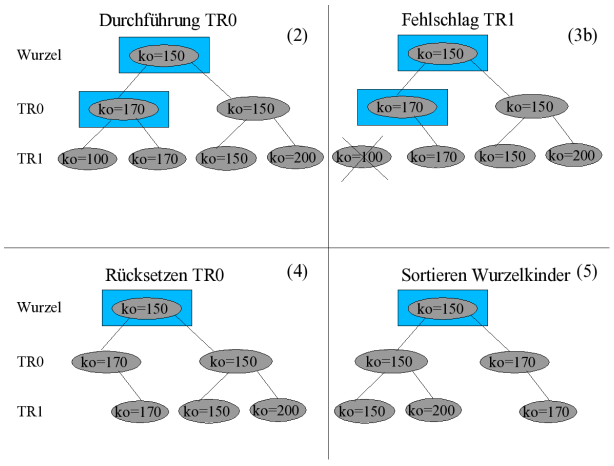


Abbildung 13: Kostenstruktur des Variantenbaums im Reservierungsschritt bei reiner Kostenoptimierung

Verhalten bei Teilreservierungsfehlschlag:
Sei *partiell* nun eine Liste des Wurzelknotens und der Knoten der partiell durchgeführten Co-Reservierungsvariante (in Abbildung 13 die umrahmten Knoten). Jeder Knoten aus *par-*

tiell enthält das lokale Minimum der Knotenwerte seines jeweiligen Unterbaums. Es hat nun irgendein Knoten k aus *partiell* den minimalen Wert. k ist somit ein Knoten der nächst besten Co-Reservierungsvariante. Alle Teilreservierungen der vormals besten Variante, deren Knoten im Baum tiefer als k liegen, müssen zurückgesetzt werden.
Im Beispiel von Abbildung 13 ist k der Wurzelknoten und TR0 muß zurückgesetzt werden. Sei *undo* nun die Teilliste der Knoten aus *partiell*, deren Tiefe größer als die von k ist. Bei der Rücksetzoperation von Knoten aus *undo* muß anders vorgegangen werden als bei kinderlosen Knoten. Jeder Knoten aus *undo* speichert eventuell noch mögliche Co-Reservierungsvarianten, die nicht verworfen werden dürfen.
Für jeden Knoten u aus *partiell*, beginnend mit dem nach dem Teilreservierungsfehlschlag zuletzt reservierten, ist die Rücksetzoperation in Algorithmus 5 die folgende:

- Durchführung der Co-Reservierungsvariante fortsetzen, wenn
  - $u = k$ oder
  - $Start(u) < Start(k)+Startzeittoleranz \wedge Kosten(u) < Kosten(k)+Kostentoleranz$
- u zurücksetzen
- u wird sortiert in die Kinderliste seines Vaters eingefügt.

Im Beispiel von Abbildung 13 wird die 170 Geldeinheiten teure Variante ans Ende der Kinderliste der Wurzel eingefügt. Die Konfiguration von Bild 5 wird nun wieder zur Ausgangsposition des Tiefensucheverfahrens. Die Schritte der Bilder 1 - 5 werden so lange wiederholt, bis die Tiefensuche terminiert.

4.7.6 Analyse

Algorithmus 5 wird in Kapitel 6.9 analysiert. Untersucht wurden verschiedene Kombinationen an Kosten- und Startzeitfaktoren für Anfragen eines an sonsten gleichen Reservierungs-Workloads. Wie erwartet, verschoben sich die reservierten Co-Reservierungsvarianten von den günstigsten zu den früheren Varianten, wenn der Kostenfaktor in den Anfragen gesenkt und der Startzeitfaktor erhöht wurde. Wegen der heterogenen Tarife der Ressourcen war eine Umverteilung der Reservierungslast zu den günstigen Ressourcen zu beobachten. Wenn Co-Reservierungen weit genug im voraus angefragt wurden, konnten Teilreservierungen die gesamten Ressourcen eines günstigen Clusters belegen. Lokale Jobs mußten dann, obwohl in dem Fall alle Teilreservierungen fair gestellt wurden, lange auf freie Ressourcen warten. Der Algorithmus sollte daher um eine Lastbegrenzung erweitert werden, die unabhängig vom Fairnesskriterium wirkt.

5 Vergleichsmethodik

Feitelson et al. beschreiben mit welchen Mitteln Scheduling-Algorithmen verglichen werden können [18]. Sie unterscheiden zwischen On- und Offline-Analysen. Bei einer Offline-Analyse sind Menge und Eigenschaften der zu verarbeitenden Jobs für den Scheduler im Vorfeld bekannt. In diesem Szenario sind analytische Methoden für die Berechnung des Ausführungsplanes sinnvoll. Bei Online-Analysen muß die Plazierung von Jobs ohne Wissen über deren Submit-Zeit durchgeführt werden. Dies entspricht dem Verhalten bei realen Schedulern. Ein Job wird übertragen, wartet eventuell eine gewisse Zeit, läuft und endet schließlich irgenwann. Während dieses Vorgangs treten dem Batchsystem auf die selbe Art und Weise andere Jobs bei. Wenn im Scheduling-Algorithmus ebenso noch Reservierungen beachtet werden, dann ist das analysierte System sehr dynamisch und mit analytischen Betrachtungen schwer analysierbar. Ein Kernaspekt des Auftragsbestandszeit-Algorithmus ist, daß die exakten Job-Laufzeiten im Vorfeld dem Batchsystem-Scheduler nicht bekannt sind. Dies macht es nahezu unmöglich analytisch zu beurteilen, welche Co-Reservierungsvarianten fair zu lokalen Jobs sind oder wie vielen Anfragen zu ihren Startzeiten genügend Ressourcen zur Verfügung stehen. Alternativ kann das dynamische Zusammenspiel von Job- und Reservierungsverarbeitung durch Analysen von Simulationen verglichen werden.

5.1 Simulationsumgebung

Ziel der Simulationen ist es, die in der Beschreibung der Algorithmen vermuteten Effekte von Co-Reservierungsanfragen zu bestätigen oder zu widerlegen. Dazu muß eine Simulationsumgebung bestimmten Anforderungen genügen: Es wird ein virtuelles Grid benötigt, in dem die wichtigsten Aspekte der realen Grid-Ressourcenverwaltung modelliert sind. Dazu gehört die Simulation von Batchsystemen und der GRS und LRS Diensten. Die simulierten Batchsysteme sollten einen realistischen Job-Workload verarbeiten und einen möglichst realistischen Scheduling-Algorithmus dafür verwenden. Außerdem müssen sie Vorausreservierungen für das Vornehmen von Teilreservierungen ermöglichen. Aus technischen Gründen ist in den Simulationen ein Zeitraffermechanismus erforderlich, da die zu simulierenden Zeiträume für aussagekräftige Untersuchungen mindestens ein bis zwei Wochen lang sein müssen. Im Anschluß an Simulationen muß auf detaillierte Informationen zu den verarbeiteten Co-Reservierungsanfragen und Batch-Jobs zu deren Analyse zurückgegriffen werden können.
Es existieren zahlreiche Arbeiten über Simulations-Werkzeuge, die bei der Erschaffung einer Simulationsumgebung hilfreich sein können. In Abschnitt 5.2 werden diese näher beschrieben und in Abschnitt 5.3 die Simulationsumgebung dieser Arbeit vorgestellt. In Abschnitt 5.4 wird detaillierter auf das Problem der Zeitbeschleunigung eingegangen.

5.2 Verwandte Arbeiten

Für die Simulation von Scheduling und Ressource-Management von Grid Umgebungen gibt es zahlreiche Werkzeuge [19, 20, 21, 22, 23, 24, 25].
In diesem Abschnitt wird untersucht, ob diese verwendet werden können, um darauf eine Simulationsumgebung aufzubauen. Microgrid [24] ist ein Emulator für Grid-Umgebungen. Eine detaillierte Emulation von Netzwerk-, CPU- und Speicherressourcen und Grid-Diensten ist mit Microgrid möglich. Beispielsweise ist die Entwicklung von Globus Anwendungen [6]

möglich, ohne daß Anpassungen der entwickelten Software in realen Grid-Umgebungen nötig sind. Grid-Sim [20] bietet beispielsweise eine Java-API und Sim-Grid eine C-API [21]. Grid-Sim erlaubt eine umfangreiche Modellierung von Benutzern und Ressourcen und bietet einfache Scheduling-Algorithmen für simulierte Batchsysteme. Für „time-shared"-Ressourcen liegt ein Round-Robin-Scheduler vor, für „space-shared"-Ressourcen ein FCFS-Scheduler. Vorausreservierungen lassen sich ebenfalls in Simulationen vornehmen. Grid-Sim wird beispielsweise für die Simulation des in Kapitel 4.7.1 vorgestellten Deadline-Budget-Algorithmus verwendet. In dieser Arbeit wird Grid-Sim nicht verwendet, da die LRS Dienste auf simulierten Batchsystemen aufsetzen müßten. Diese bieten zum einen nicht die volle Funktionalität realer Batchsysteme (zum Beispiel keinen Backfill-Algorithmus für Jobs) und zum anderen gibt es eine Alternative mit dem Maui-Scheduler, der lokalen Reservierungsdiensten ein virtuelles Batchsystem vorzutäuschen kann, daß sich exakt wie ein reales verhält. Außerdem muß bei der Verwendung von Rahmenwerken Einarbeitungszeit bei der Entwicklung der Simulationsumgebung eingeplant werden, die durch die schnelle Entwicklung einer eigenen Lösung eventuell nicht gerechtfertigt ist.
Es ist auch möglich eine eigene Simulationsumgebung um eine bereits existierende Komponente aufzubauen. Der Maui Scheduler kann die Scheduler verschiedener Batchsysteme (PBS, SGE) ersetzen [10, 25]. Er unterstützt Vorausreservierungen und eine Vielzahl an Job-Scheduling-Algorithmen. Er kann aber auch zu Simulationszwecken eingesetzt werden. In einem Simulationsmodus kann Maui Jobs und Reservierungen auf virtuellen Ressourcen verarbeiten lassen. Es ist für eine externe Komponente, die über Maui Reservierungen im Batchsystem vornimmt transparent, ob Maui ein echtes Batchsystem verwaltet oder ein virtuelles. Weiterhin protokolliert Maui verarbeitete Jobs in einer Historie. Er unterstützt verschiedene Formen der Zeitbeschleunigung. Der Aufwand eine eigene Batchsystem-Komponente zu implementieren wird, falls der Maui-Scheduler in der Simulationsumgebung verwendet wird, dem Entwickler abgenommen. Mit Diagnosewerkzeugen von Maui kann, wie in Kapitel 4.4.4 beschrieben, die Auftragsbestandszeit in Algorithmus 3 einfacher berechnet werden.
In Simulationen treten oft lange Zeiten keine Ereignisse ein (keine Jobs oder Reservierungen starten, enden oder werden übertragen). Es lohnt sich daher ein Zeitmodell für die Simulationsumgebung zu wählen, in der die Zeit sprungweise zu Ereignissen voranschreitet. Damit ließe sich das Problem der Zeitbeschleunigung in den durchzuführenden Simulationen lösen. Simjava [23] ist ein Rahmenwerk für Java mit dem dies möglich ist. Klassen, die von einer Sim_Entity Klasse erben, können Ereignisse generieren und ihrerseits auf bestimmte Ereignisse warten. Diskrete Zeitsprünge mit Simjava lassen sich allerdings nur dann realisieren, wenn alle in der Simulation beteiligten Komponenten darauf aufbauen. Gibt es jedoch eine nicht auf Simjava aufbauende externe Komponente, beispielsweise eine Batchsystem-Komponente mit dem Maui-Scheduler, dann kann diese keine Ereignisse verarbeiten oder erzeugen. In diesem Fall muß eine andere Lösung gefunden werden.

5.3 Aufbau des Simulators und Simulationsablauf

In dieser Arbeit wird Microgrid nicht verwendet, da nur die wesentlichen Effekte von Co-Reservierungsanfragen analysiert werden sollen und die Emulation eines Grids dafür überdimensioniert ist. Grid-Sim und Sim-Grid sind zwar geeignete Simulationsumgebungen, wegen der Einschränkungen und Anpassungen, die bei deren virtuellen Batchsystemen vorgenom-

men werden müßten, werden sie in dieser Arbeit nicht verwendet. Der Maui-Scheduler wird wegen der vielen beschriebenen Vorteile als Batchsystem-Komponente im Simulator verwendet. Da das ereignisgesteuerte Voranschreiten der Simulationszeit mit Simjava deshalb nur eingeschränkt möglich ist, wird auf dieses Werkzeug verzichtet.
Nun wird beschrieben aus welchen Teilen der Simulator besteht und wie sie miteinander interagieren. In Abbildung 14 sind diese dargestellt. Neben den aus den Algorithmen bekannten Komponenten: GRS, LRS und dem Scheduler wird ein Batchsystem-Manager (BM) benötigt, der in das Batchsystem während der Simulation Jobs überträgt. Ein Reservierungs-Manager (RM) unternimmt ähnliches für Co-Reservierungsanfragen, er erzeugt für diese GRS Instanzen, die die Anfrage verarbeiten. Die zu verarbeitenden Co-Reservierungsanfragen sind in einer Reservierungs-Workload-Datei formuliert und die zu verarbeitenden Jobs der Batchsysteme in Job-Workload-Trace-Dateien. Anfragen für Jobs und Co-Reservierungen sind jeweils nach Submit-Zeit geordnet, so daß die Manager-Komponenten eine nach der anderen bei Erreichen der Submit-Zeit verarbeiten lassen. Bei dem Ablauf des Co-Reservierungsalgorithmus wird gegenüber dem generellen Algorithmenschema von Abbildung 5 eine Vereinfachung durchgeführt. Die Schritte 2-3, in denen der Informationsdienst nach Cluster-Kandidaten für eine Anfrage befragt wird, entfällt in der Simulation. Alle simulierten Cluster sind für die Anfragen verfügbar. Die Reservierungsdienste loggen die für die Untersuchungen relevanten Daten für Co-Reservierungsanfragen; der Scheduler diejenigen für Jobs.

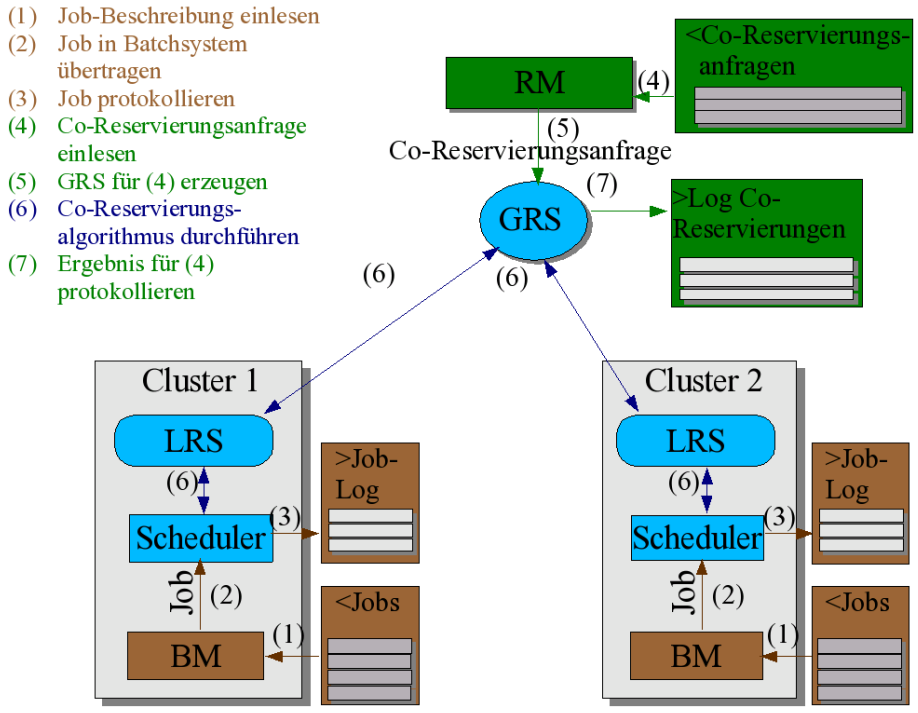


Abbildung 14: Architektur des Simulators und Simulationsverlauf

5.4 Simulationszeitsteuerung

In diesem Abschnitt wird beschrieben nach welchem Modell in dem Simulator die Simulationszeit voranschreitet.

In den in Kapitel 6 durchgeführten Simulationen werden etwa 1000 bis 2000 Jobs pro Batchsystem in einer Simulation verarbeitet. Wenn diese Jobs in ihrer natürlichen Dauer verarbeitet würden, dann benötigt eine Simulation etwa zwei Wochen. Es gibt verschiedene Möglichkeiten diese Zeit zu verkürzen. Eine Möglichkeit schneller zu simulieren ist es, einen Zeitraffer zu verwenden, so daß in jeder simulierten Komponente die Simulationszeit um einen festen Faktor beschleunigt wird. Eine Synchronisierung der Zeiten zwischen den Komponenten ist dann nicht nötig. Die Zeitraffertechnik kann jedoch die Ergebnisse der Simulationen verzerren, denn bei der Anfrageverarbeitung mit dem Co-Reservierungsalgorithmus vergeht, wenn die Anfrage ineffizient bearbeitet wird, viel natürliche Zeit. Frühe und späte Reservierungsversuche sehen nicht die selben Jobs und Reservierungen in den Batchsystemen wie sie bei Echtzeit zu beobachten wären. Ein weiterer Engpass im Zeitraffermodell ist der Batchsystem-Scheduler selbst. Er berechnet in periodischen Abständen einen Schedule und benötigt dafür einige Zeit. Er hat zu prüfen, ob Jobs fertig werden oder starten können und ob Reservierungen beginnen oder enden. Bei der Zeitbeschleunigung muß er diese Berechnungen häufiger durchführen und läuft Gefahr die definierte konstante Beschleunigung nicht beibehalten zu können. Die Verteilung der Simulator-Komponenten auf mehrere Rechner könnte den Effekt mindern. Ein effizienterer aber aufwändiger zu implementierender Ansatz der Zeitsteuerung ist mit einem ereignisorientierten diskreten Zeitmodell möglich. Die Idee ist, daß nur dann, wenn eine Anfrage bearbeitet wird, ein natürlicher Zeitfluss benötigt wird. Sonst können die Komponenten sprungweise von Job-Submit, -Start und -Ende und Reservierungs-Submit, -Beginn und -Ende die Zeit verarbeiten. Die Batchsystem-Scheduler müssen nun nicht mehr periodisch zu eventuell unbedeutenden Zeiten einen Schedule berechnen, sondern nur dann, wenn eines der beschriebenen Ereignisse eintritt. Die Anzahl der unnötig eingeplanten Scheduling Iterationen in einem Batchsystem läßt sich reduzieren, wenn nur zu den batchsystemeigenen Ereignissen die Zeit einstellt wird.
Weil jeder BM Jobs zu unterschiedlichen Zeiten in das zugehörige Batchsystem überträgt, die zu unterschiedlichen Zeiten starten und enden, macht es Sinn den BM die Zeitsteuerung ihrer Batchsystem-Komponente zu überlassen. Für eine korrekte Bearbeitung von Co-Reservierungsanfragen müssen in den Batchssystemen zu weiteren Zeiten Scheduling Iterationen eingeplant werden. Diese erhalten die BM von einer externen Zeitgeberkomponente. In Abbildung 15 sind diese zusätzlichen Ereignisse dargestellt. Der RM teilt dem Zeitgeber die Submit-Zeit (1) und das Ende der Bearbeitung durch den Co-Reservierungsalgorithmus (2) mit. Start- (3) und End-Zeit (4) von Co-Reservierungen werden ebenfalls zum Zeitpunkt (2) dem Zeitgeber übermittelt. Die Synchronisierung zur Submit-Zeit einer Co-Reservierungsanfrage ist nötig, um bei der Statusanfrage eine aktuelle Sicht der Batchsysteme zu erhalten. Die Ereignisse (3)-(4) sind nötig, damit der Scheduler die reservierten Ressourcen zu den angeforderten Zeiten auch tatsächlich belegt. Die Anfragebearbeitung einer Co-Reservierung dauert eine gewisse Zeit. Bei einem natürlichen Zeitfluss könnten nach einem Co-Reservierungs-Submit noch Jobs übertragen werden (5). Damit dies im diskreten Zeitmodell weiterhin möglich ist, fügt der Zeitgeber in kleinen periodischen Abständen weitere Haltepunkte ein (6). Der Zeitgeber wartet vor der Benachrichtigung der BM die Zeit bis zum nächsten Ereignis in realer Zeit ab und simuliert dadurch den natürlichen Zeitverlauf.

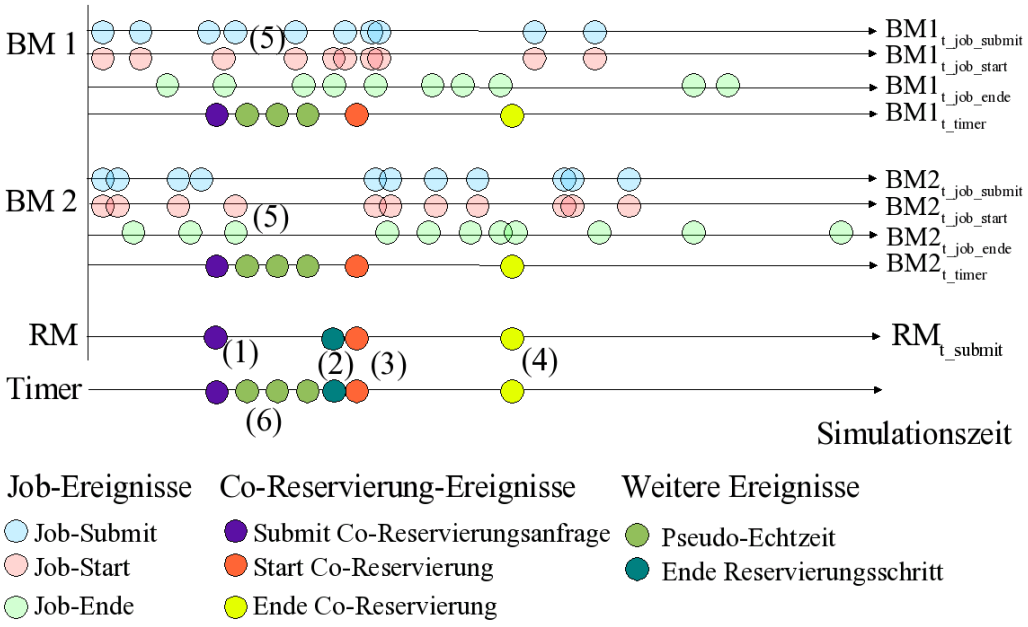


Abbildung 15: Ereignisgesteuertes Zeitmodell: Die Simulationszeit schreitet schrittweise zu Job-, Reservierungs- und Synchronisierungsereignissen voran.

6 Untersuchungen

In diesem Kapitel werden verschiedene Untersuchungen für die vorgestellten Algorithmen durchgeführt. In Abschnitt 6.1 werden die in den Simulationen untersuchten Kriterien vorgestellt. Anschließend werden Workload-Konfigurationen, die in Simulationen eingesetzt werden präsentiert. In Abschnitt 6.2 sind dies die verwendeten Job-Workloads und in Abschnitt 6.3 die eingesetzten Reservierungs-Workloads. Weil die Algorithmen jeweils schrittweise um zusätzliche Funktionalität erweitert wurden, lassen sich nicht in allen Untersuchungen alle Kriterien analysieren. Weiterhin werden die Algorithmen wegen der Vielzahl der Simulationsparameter und Untersuchungskriterien nur für bestimmte Kombinationen analysiert, bei denen es sinnvoll ist, sie zu untersuchen.
In Abschnitt 6.4 wird für einen stets gleich bleibenden Reservierungs-Workload der trivialen Algorithmus untersucht. Besonderes wichtig sind bei dieser Untersuchung die Auswirkungen unterschiedlich großer Auswahlmöglichkeiten bei Co-Reservierungsvarianten auf die Kriterien Reservierungseffizienz und Co-Reservierungsfehlschlagrate.
In Abschnitt 6.5 wird ebenfalls die Reservierungseffizienz untersucht, allerdings wird dort auf unterschiedliche Reservierungs-Workloads und den Vergleich des Trivialen- zum CPU-Zähl-Algorithmus eingegangen.
In Abschnitt 6.6 werden die unterschiedlichen Effekte durch die Fairness-Mißachtung zu wartenden Jobs beim CPU-Zähl-Algorithmus und die Fairness-Berücksichtigung beim Auftragsbestandszeit-Algorithmus analysiert. Untersucht werden die Kriterien: Co-Reservierungsfehlschlagrate, durchschnittliche Batchsystem-Auslastung, maximale Job-Verzögerung, durchschnittlicher Unfairness-Grad und der durchschnittliche Expansion-Faktor. In Abschnitt 6.7 wird untersucht, ob mit dem Backfill-Algorithmus gegenüber dem unoptimierten Auftragsbestandszeit-Algorithmus tatsächlich niedrigere Co-Reservierungsfehlschlagraten erreicht werden können. In Abschnitt 6.8 wird die durchschnittliche relative Startzeitdifferenz für Reservierungsanfragen mit startzeitoptimierenden Versionen von Algorithmus 2, 3 und Backfill-Optimierung (Algorithmus 4) verglichen. Zuletzt werden in Abschnitt 6.9 für Algorithmus 5 die Auswirkungen unterschiedlicher Kosten- und Startzeitaspekte für Anfragen auf die relative Start- und Kostendifferenz untersucht. Ebenso wird gezeigt, welche Auswirkungen heterogene Kosten der Ressourcen auf die Verteilung der Reservierungslast und die Verzögerung von Jobs haben.

6.1 Untersuchte Kriterien

In diesem Abschnitt werden die in den Simulationen untersuchten Kriterien beschrieben. Zunächst wird die Relevanz der untersuchten Kriterien für die beteiligten Interessengruppen erklärt. Anschließend folgt die Definition der Kriterien.
Die Interessen der beteiligten Gruppen sind:

Lokale Benutzer von Batchsystemen
Algorithmendesignziel: frühe Ausführung lokaler Jobs
Untersuchungskriterien: *maximale Job-Verzögerung*, *durchschnittlicher Unfairness-Grad*, *durchschnittlicher Expansion-Faktor*

Systemadministratoren von Clustern
Algorithmendesignziel: hohe Auslastung des Clusters
Untersuchungskriterien: *durchschnittliche Auslastung*

Grid-Benutzer schnelle, möglichst dem theoretischen Optimum nahe Reservierung einer Co-Reservierungsvariante
Untersuchungskriterien: *Relative Kostendifferenz*, *Relative Startzeitdifferenz*, *Co-Reservierungsfehlschlagrate*, *Reservierungseffizienz*

Die Untersuchungskriterien sind in den Simulationen folgendermaßen definiert:

Maximale Job-Verzögerung
Die maximalen Job-Verzögerung wird berechnet als Differenz der Job-Start- zur Job-Submit-Zeit.

Unfairness-Grad
Der Auftragsbestandszeit-Algorithmus wurde entworfen, um Co-Reservierungen erst nach wartenden Jobs starten zu lassen. Um dies zu prüfen, wird nun die Metrik Unfairness-Grad definiert. Sei bei Submit einer Co-Reservierungsanfrage a die Anzahl der wartenden Jobs in den verwendeten Batchsystemen *warte_vor*. Sei weiterhin *warte_noch* die Anzahl, der beim Start der Teilreservierungen von a immer noch wartenden Jobs von *warte_vor*. So wird der Unfairness-Grad von a definiert als:

$$\textit{Unfairness-Grad}_a = \frac{warte\_noch}{warte\_vor}$$

Sollten keine Jobs bei Submit von a warten, so ist der Unfairness-Grad von a gleich 0.

Durchschnittlicher Expansion-Faktor
Feitelson et al. untersuchten Performance-Metriken, mit denen Scheduling-Algorithmen hinsichtlich ihres Einflusses auf die Verzögerung von Jobs verglichen werden können [18]. Sie fanden heraus, daß die durchschnittliche Antwortzeit (Wartezeit + Ausführungszeit) der Jobs mit Einschränkungen eine geeignete Metrik ist, um Scheduling-Algorithmen bei gleichen Job-Workloads zu beurteilen. Sie bemängelten, daß bei der Analyse durchschnittlicher Antwortzeiten große Jobs bevorzugt werden, die in der Praxis seltener sind, jedoch wegen länger Wartezeiten einen größeren Einfluß auf die Durchschnittswerte haben. Zur Umgehung dieses Problems schlagen sie eine Normalisierung der Antwortzeit mittels eines Slowdown-Faktors vor, in dem die Antwortzeit durch die Ausführungszeit des Jobs geteilt wird. In dieser Arbeit wird ein ähnlicher Faktor, der Expansion-Faktor verwendet.
Er ist definiert als:

$$\textit{Expansion-Faktor} = \frac{Wartezeit + Laufzeit}{\textit{WallClock-Zeit}}$$

Der Expansion-Faktor ähnelt dem Slowdown-Faktor, da bei ihm die Antwortzeit durch die WallClock-Zeit des Jobs, also der Obergrenze der Ausführungszeit geteilt wird. Falls die WallClock-Zeiten groß genug sind, wird mit dem durchschnittlichen Expansion-Faktor eine gute Normalisierung der durchschnittlichen Antwortzeiten erreicht.

Durchschnittliche Auslastung
In den Simulationen werden ab einem fest definierten Zeitpunkt der Simulationszeit keine weiteren Job- und Reservierungs-Submits mehr zugelassen. Nach Ablauf der letzten dann noch laufenden Jobs und Co-Reservierungen endet die Simulation. Die Bestimmung der durchschnittlichen Batchsystem-Auslastung ist nur für die vor Submit-Stop gelaufenen Co-Reservierungen und Jobs repräsentativ, da in realen Systemen die

Workloads nie aufhören. Teilreservierungen und Jobs, die sich mit der Zeit Submit-Stop überschneiden werden nur anteilig bei der Berechnung der Auslastung berücksichtigt. Seien *cs_jobs* die Summe der anrechenbaren CPU-Sekunden von den Jobs, *cs_reservierungen* die Summe der anrechenbaren CPU-Sekunden von Teilreservierungen, *max_cpus* die maximale Anzahl von CPUs an einem Cluster und *t_stop* der Submit-Stop-Zeitpunkt. Dann wird die durchschnittliche Auslastung für das Batchsystem wie folgt berechnet:

$$Util = \frac{cs\_jobs + cs\_reservierungen}{max\_cpus \cdot t\_stop}$$

Co-Reservierungsfehlschlagrate

Mit der Co-Reservierungsfehlschlagrate wird angegeben wie groß der Anteil der fehlgeschlagenen Co-Reservierungsanfragen in einer Simulation ist. Der Wertebereich der Co-Reservierungsfehlschlagrate liegt zwischen 0 (keine Co-Reservierungsfehlschlag) und 1 (alle fehlgeschlagen).

Filterungsgrad und Undurchführbarkeitsrate

Ab Algorithmus 2 können bei der Bearbeitung einer Co-Reservierungsanfrage bereits vor dem Reservierungsschritt Teilreservierungsvarianten verworfen werden. Um Effekte, die dabei auftreten können, besser untersuchen zu können, werden zwei Hilfsmetriken entworfen.

Mit dem Filterungsgrad wird angegeben wie groß für ein Batchsystem c bei einer Co-Reservierungsanfrage a die Anzahl gefilterter Teilreservierungsvarianten *gefiltert* im Verhältnis zu den ungefilterten Teilreservierungsvarianten *ungefiltert* (Teilreservierungsvarianten nach Algorithmus 1) ist.

$$Filterungsgrad_{ac} = \frac{gefiltert}{ungefiltert}$$

Bei einem Filterungsgrad von 1 ist a bei c nicht durchführbar.

Die Undurchführbarkeitsrate gibt für c an, wie groß der Anteil der Co-Reservierungsanfragen ist, für die c keine Teilreservierungsvarianten bildet, bezogen auf die Anzahl aller Anfragen.

$$Undurchfuehrbarkeitsrate_c = \frac{Anzahl\ Anfragen\ mit\ Filterungsgrad\ 1}{Anzahl\ Co\text{-}Reservierungsanfragen}$$

Durchschnittliche Reservierungseffizenz

Jede Nachricht zwischen GRS und LRS bedeutet Aufwand, zum einen dauert die Nachrichtenübertragung eine gewisse Zeit und zum anderen muß der Batchsystem-Scheduler den Schedule prüfen. Um in einer Simulation festzustellen wie effizient eine Co-Reservierungsanfrage bearbeitet wird, kann das Verhältnis der Summe der durchgeführten Teilreservierungsaufträge und -rücksetzaufträge zu deren Gesamtanzahl berechnet werden. Am ineffizientesten wird eine Anfrage bearbeitet, wenn alle Co-Reservierungsvarianten ausprobiert werden müssen und Varianten jeweils bei der letzten Teilreservierung scheitern. Bei Algorithmus 5 kann ein Teilreservierungsfehlschlag bei der letzten Komponente dazu führen, daß der gesamte reservierte Pfad zurückgesetzt werden muß. Die inneren Knoten können daher zweimal pro Variante durchlaufen werden. Berechnung der Anzahl C der Co-Reservierungsvarianten kann mit Gleichung 1 durchgeführt werden (siehe Kapitel 3). Die maximale Nachrichtenanzahl N bei C Co-Reservierungsvarianten

und r Teilreservierungen berechnet sich für Algorithmus 5 daher als:

$$N = C + 2 \cdot C^{r-1}$$

Bei den anderen Algorithmen werden Teilreservierungen jeweils nur um eine Stufe zurückgesetzt und dann die nächste von dort aus mögliche Co-Reservierungsvariante probiert. Jede Kante zu inneren Knoten im Variantenbaum kann nur einmal durchlaufen werden. Die Anzahl der Knoten der Tiefe i des Variantenbaums C_i entsprechen der Anzahl der Co-Reservierungsvarianten einer nur i-teiligen Co-Reservierungsanfrage. Die maximale Nachrichtenanzahl berechnet sich daher als:

$$N = C + 2 \cdot \sum_{i=1}^{r-1} C_i$$

Werden im Reservierungsschritt bei einer Co-Reservierungsanfrage n Nachrichten benötigt, so wird die Effizienz der Anfragebearbeitung definiert durch:

$$Eff = 1 - \frac{n}{N}$$

Relative / normierte Startzeitdifferenz

In Algorithmus 4 und 5 können die Co-Reservierungsvarianten nach der Startzeit sortiert werden. Die relative Startzeitdifferenz gibt an, wie früh eine Co-Reservierungsvariante im Verhältnis zur frühesten und spätesten Co-Reservierungsvariante ist. Sei *min_start* die früheste und *max_start* die späteste mögliche Startzeit aller gebildeten Co-Reservierungsvarianten, dann ist die relative Startzeitdifferenz für eine Co-Reservierung mit Startzeit *start* definiert als:

$$Rel\_start = \frac{start - min\_start}{max\_start - min\_start}$$

Falls keine Startzeitunterschiede zwischen den Varianten bestehen sei *Rel_start* gleich 0.

Relative / normierte Kostendifferenz

In Algorithmus 5 können die Co-Reservierungsvarianten nach Kosten sortiert werden. Die relative Kostendifferenz gibt an, wie teuer eine Co-Reservierungsvariante im Verhältnis zur billigsten und teuersten Co-Reservierungsvariante ist. Sie ist definiert als:

$$Rel\_kosten = \frac{kosten - min\_kosten}{max\_kosten - min\_kosten}$$

Falls keine Kostenunterschiede zwischen den Varianten bestehen, sei *Rel_kosten* gleich 0.

6.2 Job-Workloads

In diesem Abschnitt wird beschrieben, welche Eigenschaften die von den Batchsystemen in den Simulationen zu verarbeitenden Jobs besitzen. Bei der Definition der Job-Workloads für die Batchsystem-Manager gibt es zwei Möglichkeiten: Die Jobs können nach einer zufälligen Verteilung synthetisch erzeugt werden oder es werden Historien von Jobs, die an realen Clustern liefen, als Workload-Vorlage benutzt. Feitelson et al. stellen Forschern verschiedene Job-Workload-Traces zur Verfügung [26]. Diese wurden für Analysen von Scheduling-Algorithmen von Super-Computing-Zentren für deren Rechner aufgezeichnet. Job-Workload-Traces sind

Historien von in Batchsystemen gelaufenen Jobs innerhalb eines bestimmten Zeitraums. In ihnen sind folgende Informationen über die Jobs enthalten:

Ressourcenanforderungen
angeforderte Prozessoren, Netzwerk, Betriebssystem etc.

Scheduling-Ereignisse
Übertragungszeit, Startzeit, Endzeit, Erfolgsstatus der Ausführung etc.

Ausführungsumgebung
Warteschlange, belegte Prozessoren

Verwendete Job-Workload-Traces:
Grundlage für die verwendeten Batch-Workloads in den Simulationen sind Teilmengen des SDSC Blue Workload-Trace [26]. Dieser Job-Workload-Trace ist mit Job-Spuren von 3 Jahren besonders lang. Er ist daher geeignet, um jedem Batchsystem-Manager während der Simulationsläufe einen anderen Ausschnitt zur Verfügung zu stellen. Der SDSC Cluster besteht aus 144 Knoten mit je 8 Prozessoren. Einige Jobs mit ungültigen Werten (z.B. negativen Laufzeiten) werden aus dem Eingabe-Workload-Trace der BM entfernt. Alle Jobs haben die gleiche Priorität und laufen in einer einzigen Warteschlange. Weiterhin wird eine Optimierung verwendet, damit die Simulation schneller verarbeitet werden kann. Es werden Jobs mit einem CPU-Bedarf von weniger als 128 CPUs zusammengelegt, wenn sie die gleiche WallClock-Zeit teilen und ihre Submit-Zeiten nicht länger als eine Stunde auseinander liegen. Die Anzahl der Jobs und die Zeit zur Verarbeitung einer Simulation reduziert sich durch dieses Verfahren um etwa 30 Prozent.
Eine Job-Simulation wurde durchgeführt, auf die sich in den Untersuchungen später bezogen wird. Die Batchsysteme verarbeiten Jobs, deren Submit-Zeiten kleiner als zwei Wochen sind. Referenzwerte zu den Anzahlen verarbeiteter Jobs, den durchschnittlichen Batchsystem-Auslastungen und Expansion-Faktoren sind in den Tabellen 6(a) bis 6(c) zu sehen.
Im Durchschnitt werden für die Jobs der Eingabe-Workload-Traces bei jedem BM unterschiedlich viele CPUs angefordert. In Abbildung 16 wird gezeigt, wie viel Last Jobs der unterschiedlichen Größen-Kategorien bei den Batchsystemen in der Job-Simulation durchschnittlich erzeugten.

BM 0	BM 1	BM 2	BM 3	BM 4	BM 5	BM 6
2226	1026	2167	1829	1955	1771	2068

(a) Anzahl der Jobs in 2 Wochen Simulationszeit

BM 0	BM 1	BM 2	BM 3	BM 4	BM 5	BM 6
68	65	72	84	82	68	85

(b) Durchschnittliche prozentuale Auslastung der Batchsysteme

BM 0	BM 1	BM 2	BM 3	BM 4	BM 5	BM 6
2	1.73	2.04	1.63	2.4	1.13	5.92

(c) Durchschnittlicher Expansion-Faktor

BM 0	BM 1	BM 2	BM 3	BM 4	BM 5	BM 6
0.23	0.45	0.37	0.40	0.36	0.36	0.32

(d) Durchschnittliche WallClock-Genauigkeit der Jobs in 2 Wochen Simulationszeit

Tabelle 6: Referenzwerte untersuchter Kriterien in der Job-Simulation

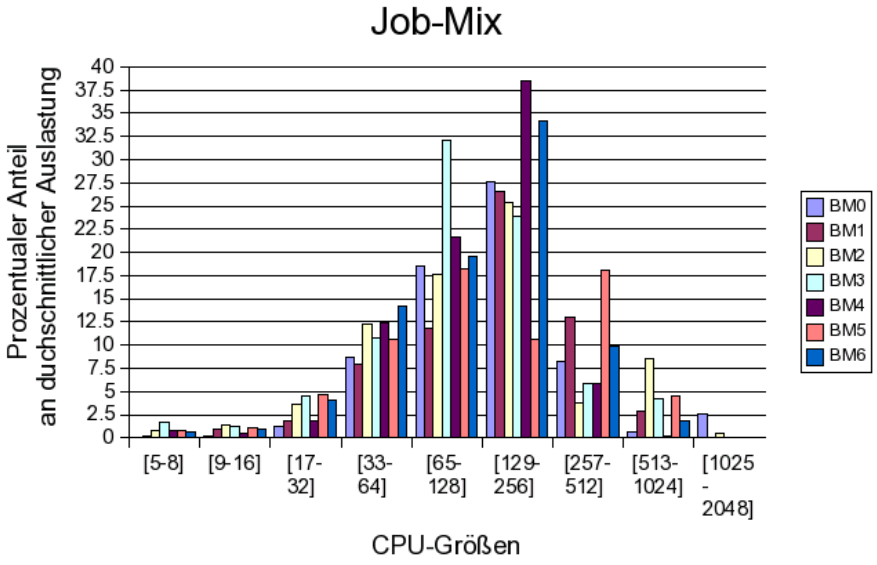


Abbildung 16: Aufschlüsselung der durchschnittlichen Auslastung

6.3 Reservierungs-Workloads

Bisher gibt es wenig bekannte Untersuchungen zu Co-Reservierungsalgorithmen [27], daher sind keine Co-Reservierung-Workload-Traces, wie sie für die Simulation der Batch-Jobs verwendet werden, bekannt. Co-Reservierungs-Workloads werden deshalb aus Job-Workloads abgeleitet.
In den Simulationen werden Workloads mit Anfragen unterschiedlich langer Vorausbuchzeit und Startzeitspanne untersucht. Nun wird die Zusammensetzung von drei in den Untersuchungen verwendeten Reservierungs-Workloads beschrieben.

Reservierungs-Workload 1:

Co-Reservierungsanfragen

Die Auswahl einer Zufallszahl aus einer stetigen Gleichverteilung entschied, ob ein Job eines Job-Workload-Traces als Vorlage für eine Co-Reservierungsanfrage diente. Aus dem Job-Workload für BM 0 wurden somit, bei einer Auswahlchance von eins zu zehn, 213 Jobs zufällig ausgewählt. Es seien nun *wc* die WallClock-Zeit einer Job-Vorlage, *cpus* die angeforderten CPUs des Jobs und *random* für jede zu konstruierende Anfrage eine zufällige Zahl zwischen 0 und 1 [3].
Die Anfrageparameter werden wie folgt definiert:

Dauer *dauer*	*wc*
Ressourcenliste *res*	[*cpus*, $min(1160, round(0.7 \cdot cpus + 0.6 \cdot random \cdot cpus)]$
Früheste Startzeit *f_start*	$random \cdot$ *Job-Submit-Stop*
Späteste Endzeit *deadline*	min (start + dauer + 86400, $start + dauer + 6 \cdot wc$)
Submit-Zeit	$max(0, f\_start - 3 \cdot wc)$

Bemerkungen:
1160 ist die CPU-Obergrenze jedes simulierten Clusters, der Job-Submit-Stop ist nach 2 Wochen. Die Deadline ist begrenzt, damit Reservierungen möglichst in der Zeit vor dem Job-Submit-Stop laufen.

Eigenschaften des gesamten Reservierungs-Workload

Theoretisch erzeugbare Gesamtlast durch alle Teilreservierungen	1 Millarde CPU-Sekunden (gerundet)
ø Startzeitspanne	Mittelwert und Standardabweichung 60000 Sekunden
ø Größe einer Teilreservierung	Mittelwert 80 CPUs und Standardabweichung 120 CPUs

Ein einzelnes Batchsystem würde etwa 10 Tage benötigen, um bei maximaler Auslastung alle Teilreservierungen der Co-Reservierungsanfragen zu verarbeiten. In Abbildung 17 ist eine Häufigkeitsverteilung der Submit-Zeiten von Co-Reservierungsanfragen dargestellt. Gezeigt wird die Anzahl von Submits, die in einem Zeitraum von acht Stunden starten. Abbildung 18(a) zeigt eine Verteilung der WallClock-Zeiten der verwendeten Job-Schablonen. Diese sind die Vorlage für Startzeitspanne, Dauer und Vorausbuchzeit

[3] Reservierungs-Workload 1,2,3 bezeichnen in den Simulationen immer die selben Workloads.

der Reservierungsanfragen. Abbildung 18(b) zeigt eine Verteilung der CPUs der ersten Teilreservierung von Co-Reservierungsanfragen.

Reservierungs-Workload 2 / kurze Vorausbuchzeit:
Bei sonst gleichen Anfrageparametern liegt die Submit-Zeit von Anfragen nur $\frac{wc}{2}$ vor f_start.

Reservierungs-Workload 3 / kurze Vorausbuchzeit und Startzeitspanne:
Bei sonst gleichen Anfrageparametern liegt die Submit-Zeit von Anfragen nur $\frac{wc}{2}$ vor f_start und die späteste Endzeit ist $min(start + dauer + 86400, start + dauer + 3 \cdot wc)$.

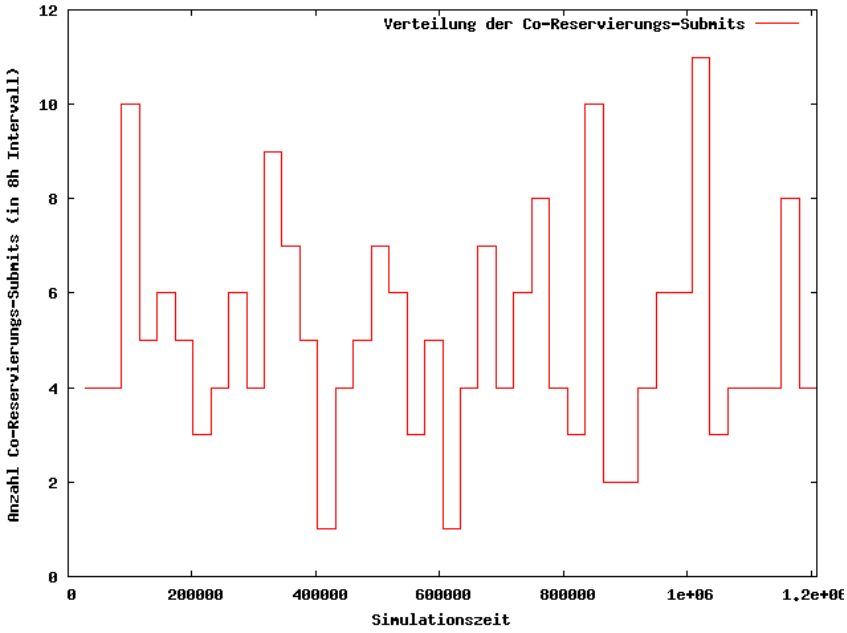
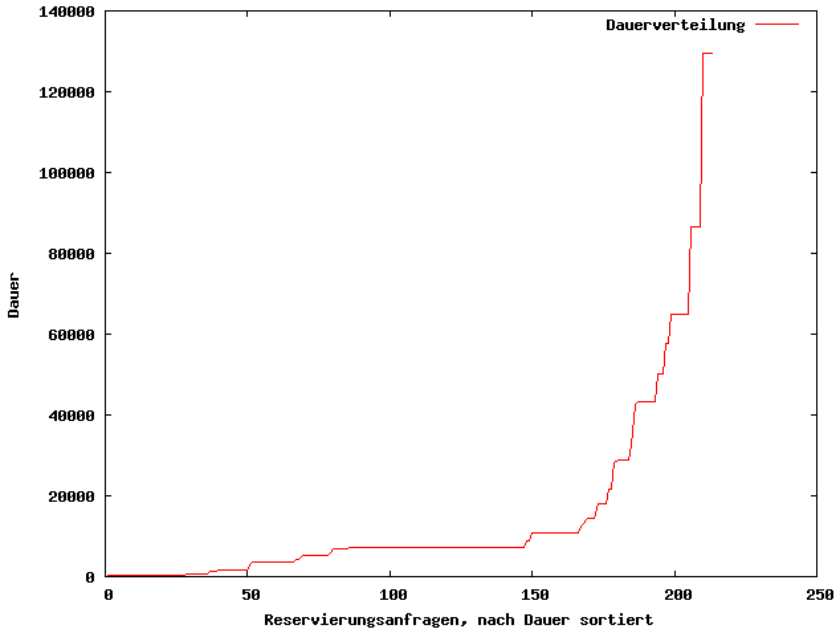
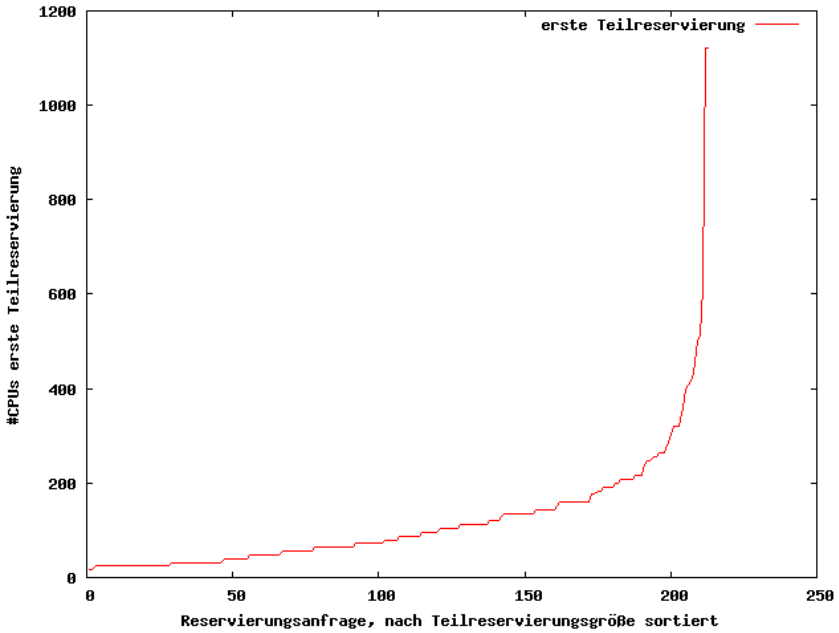


Abbildung 17: Verteilung der Co-Reservierungsanfrage-Submits auf die Simulationszeit (Reservierungs-Workload 1). Anfragen wurden jeweils zu achtstündigen Intervallen zugerechnet.



(a) Verteilung Reservierungsdauer / WallClock-Zeiten der Job-Schablonen in den Reservierungs-Workloads 1-3



(b) Verteilung der CPU-Anzahlen der ersten Teilreservierung in den Reservierungs-Workloads 1-3

Abbildung 18: Eigenschaften der Teilreservierungen

6.4 Untersuchung: Benötigte Anzahl an Co-Reservierungsvarianten

6.4.1 Ziel

Wie viele Co-Reservierungsvarianten bei einer Anfragebearbeitung gebildet werden sollten, ist für Algorithmus 1 noch zu klären. Die Co-Reservierungsvariantenanzahl ist von verschiedenen Faktoren abhängig, von denen nicht alle durch die Algorithmen beeinflußt werden können. Algorithmen müssen Co-Reservierungsanfragen des Klienten mit dessen definierten Vorgaben für die Startzeitspanne ausführen. Weiterhin ist die Anzahl der Cluster-Kandidaten bei der Bearbeitung von dem Ergebnis einer Informationsdienstanfrage abhängig. In den vorgestellten Algorithmen lassen sich durch die Parameter *minsize* und *max_zeitfenster* (siehe Kapitel 4.2.1) für eine Teilreservierung unterschiedlich viele Startzeiten pro Cluster-Kandidat erzeugen und damit die Anzahl der Co-Reservierungsvarianten steuern. Anhand dieser beiden Parameter wird der Trade-Off zwischen Erfolgswahrscheinlichkeit und Reservierungseffizenz für verschiedene Anzahlen von Cluster-Kandidaten und unterschiedliche Reservierungs-Workloads für Algorithmus 1 untersucht.

Erwartungen:

Eine unterschiedlich große Anzahl an Co-Reservierungsvarianten kann in den Simulationen zwei Effekte haben. Zum einen ist bei vielen Varianten die Chance größer eine Anfrage erfolgreich reservieren zu können, zum anderen steigt damit aber auch die Anzahl der Varianten an, die potentiell erfolglos versucht werden können. Bei Simulationen mit einem Reservierungs-Workload, in dem Co-Reservierungsanfragen kürzere Vorausbuchzeiten und dichtere Startzeiten haben, ist mit einer höheren Co-Reservierungsfehlschlagrate und einer niedrigeren Reservierungseffizienz zu rechnen. Je früher Co-Reservierungen angefordert werden und je kleiner die Startzeitspanne der Anfrage ist, desto häufiger treten vermutlich viele Kollisionen mit anderen Teilreservierungen und Jobs auf.

6.4.2 Durchführung

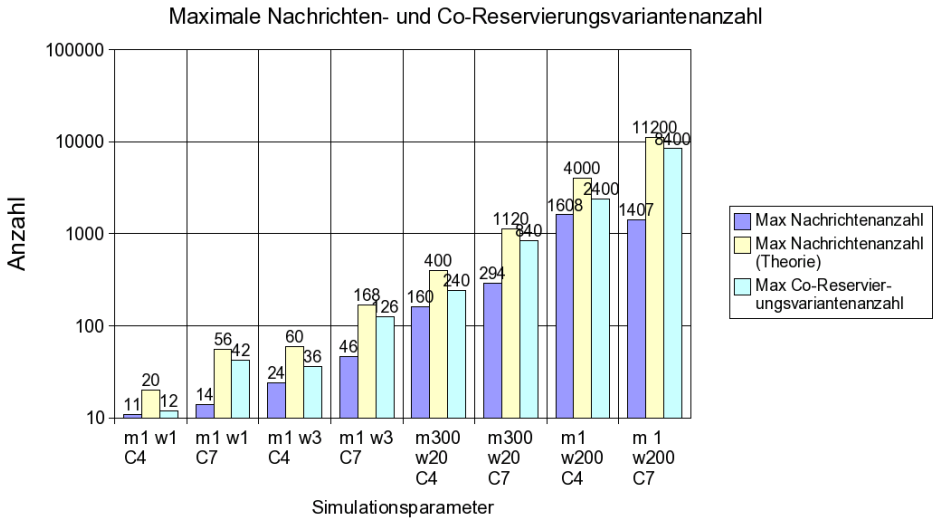
Es werden Simulationsreihen mit vier und sieben Cluster-Kandidaten und Reservierungs-Workload 1 und 3 durchgeführt. Jede Reihe besteht aus vier Simulationen, in denen unterschiedliche Startzeitanzahlen pro Cluster-Kandidat durch Variation von *minsize* und *max_zeitfenster* bei der Anfragebearbeitung getestet werden. In Reservierungs-Workload 1 haben Anfragen größere Startzeitspannen und längere Vorausbuchzeiten zur frühesten Startzeit als in Reservierungs-Workload 3. Auswirkungen von unterschiedlichen Fristen von Co-Reservierungsanfragen können somit durch die Wahl der beiden Workloads untersucht werden. Tabelle 7 zeigt die in den Simulationen verwendeten Parameter in einem Überblick.

Parameter	untersuchte Wert(e)
verwendeter Algorithmus	Algorithmus 1
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workload	Reservierungs-Workload 1 und 3 (Abschnitt 6.3)
(minsize, max_zeitfenster)	[(1,1),(1,3),(300,20),(200,1)]
Anfrage bearbeitende Cluster	[4,7]
untersuchte Kriterien	Reservierungseffizienz (Nachrichtenanzahl), Co-Reservierungsfehlschlagrate

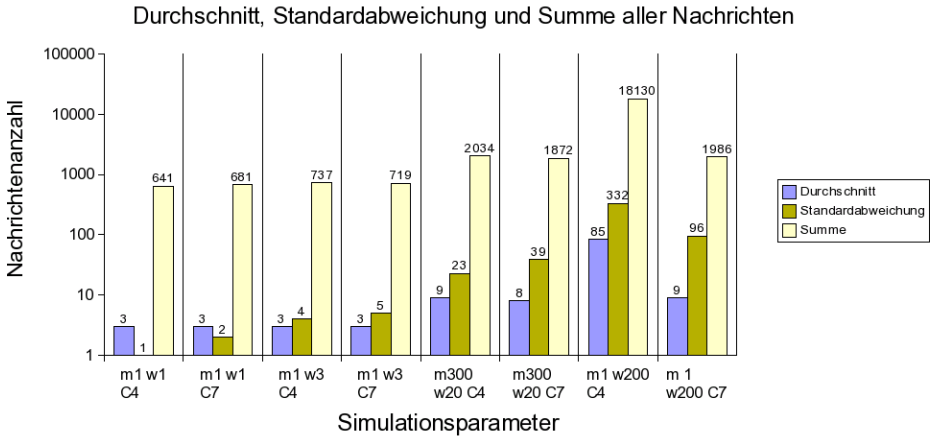
Tabelle 7: Simulationsparameter in Abschnitt 6.4

6.4.3 Resultate

Simulationen mit Reservierungs-Workload 1:
In Abbildung 19(a) ist zu erkennen, daß bei sieben Cluster-Kandidaten das Verhältnis maximal benötigter Nachrichten zu theoretisch möglichen Nachrichten günstiger ausfällt als bei vier Cluster-Kandidaten. Außerdem wurde nur ein Bruchteil der theoretisch möglichen Nachrichten bei der Bearbeitung der ineffizientesten Anfragen benötigt. In Abbildung 19(b) ist zu sehen, daß Anfragebearbeitungen im Schnitt bereits nach wenigen Schritten terminieren. Lediglich in der Simulation mit 200 Startzeiten und 4 Cluster-Kandidaten benötigt die Anfragebearbeitung überdurchschnittlich viele Nachrichten. In der Simulation mit 200 Startzeiten und 7 Cluster-Kandidaten wurden über drei viertel aller versandten Nachrichten durch nur eine einzige Co-Reservierungsanfrage verursacht. Abbildung 19(c) zeigt, daß bei vier Cluster-Kandidaten es mehr fehlgeschlagene Co-Reservierungsanfragen gibt als bei sieben. Außer bei Simulationen mit zeitlich unflexiblen Anfragen sind die Fehlschlagraten dann jedoch nahezu gleich und niedrig. Für Simulationen mit Reservierungs-Workload 1 macht es kaum einen Unterschied, ob 3 oder 200 Startzeiten pro Cluster-Kandidat betrachtet wurden.



(a) Es sind die Nachrichtenanzahlen der ineffizientesten Anfragebearbeitungen pro Simulation dargestellt. Die Maximalwerte liegen deutlich unter den theoretisch möglichen.



(b) Es sind die durchschnittlichen Nachrichtenanzahlen aller Anfragebearbeitungen pro Simulation dargestellt sowie die Standardabweichung und Summe dieser. Anfragen lassen sich in den meisten Fällen effizient bearbeiten.

m1 w1 C4	m1 w1 C7	m1 w3 C4	m1 w3 C7	m300 w20 C4	m300 w20 C7	m1 w200 C4	m1 w200 C7
0.12	0.06	0.06	0.01	0.07	0.02	0.06	0

(c) Die Co-Reservierungsfehlschlagraten in den Simulationen.

Abbildung 19: Simulationen mit Reservierungs-Workload 1, m=minsize, w=max_zeitfenster und C=Anzahl der Cluster-Kandidaten

Simulationen mit Reservierungs-Workload 3:
Abbildung 20(a) zeigt analog zu Abbildung 19(a) die maximalen Anzahlen für Nachrichten und Co-Reservierungsvarianten für den Reservierungs-Workload mit kürzeren Vorausbuchzeiten und Startzeitspannen der Anfragen. Die ineffizientesten Anfragebearbeitungen benötigten nicht mehr Nachrichten als bei Reservierungs-Workload 1, dafür stieg im Durchschnitt die Anzahl an Nachrichtenübertragungen pro Co-Reservierungsanfrage leicht an (Abbildung 20(b)). Auch die Co-Reservierungsfehlschlagraten erhöhten sich bei Reservierungs-Workload 3.

6.4.4 Erklärungen zu den untersuchten Kriterien

Durchschnittliche Reservierungseffizienz:
Eine Erklärung für die hohe durchschnittliche Reservierungseffizienz und niedrige Co-Reservierungsfehlschlagrate in den Simulationen ist, daß die möglichen Startzeiten der Anfragen im Durchschnitt sehr weit in der Zukunft lagen, wo noch keine Ressourcen vergeben waren. Dies trifft insbesondere auf Simulationen mit Reservierungs-Workload 1 zu. Bei Simulationen mit Reservierungs-Workload 3 waren die Vorausbuchzeiten und Startzeitspannen der Anfragen kürzer und somit stiegen dort wie erwartet die Anzahlen der im Durchschnitt benötigten Nachrichtenübertragungen an. Eine weitere Erklärung für die generell sehr niedrige Anzahl an Nachrichtenübertragungen ist die, daß in Algorithmus 1 Co-Reservierungsvarianten im Variantenbaum nicht nach Startzeit sortiert sind. Ein Teil der Analyse bleibt daher spekulativ. Nach Teilreservierungsfehlschlägen werden schnell Varianten zu anderen, zufällig angeordneten Startzeiten ausprobiert, die die blockierenden Reservierungen und Jobs umgehen. Bei vereinzelten Ressourcenknappheiten ist eine Anfrage deshalb trotzdem mit wenigen Nachrichten durchführbar. Insbesondere bei den Simulationen mit 8400 Co-Reservierungsvarianten kann eine andere Plazierung der Teilreservierungen stärkere Auswirkungen auf die Anzahl der bei der Anfragebearbeitung benötigten Nachrichten haben.

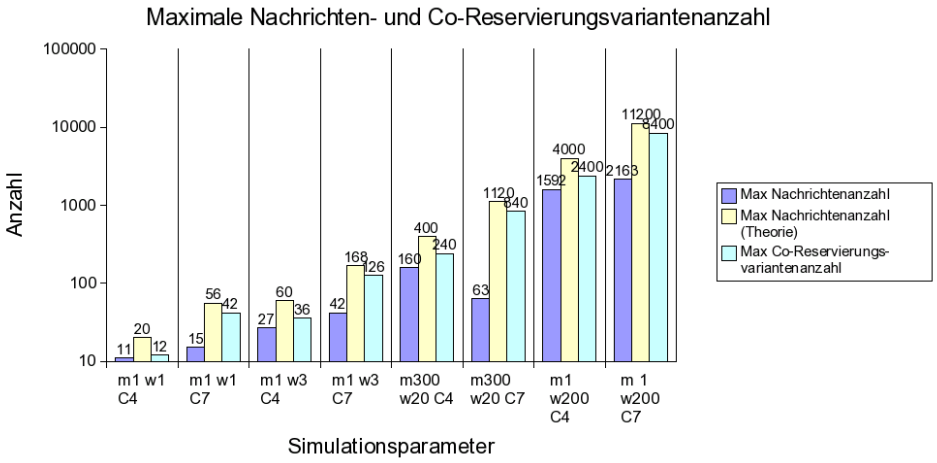
Maximale Anzahl von Nachrichtenübertragungen:
Nur ein Bruchteil der theoretisch möglichen Anzahlen an Nachrichten wurden bei Anfragebearbeitungen in den Simulationen im schlimmsten Fall benötigt. Der Hauptgrund dafür ist die Durchführungsreihenfolge der Teilreservierungen. Die vorgestellten Co-Reservierungsalgorithmen führen immer zuerst die größeren Teilreservierungen durch. Wenn bei einer Anfragebearbeitung mehrere Cluster-Kandidaten die größere Teilreservierung aufnehmen könnten, dann sehr wahrscheinlich auch die kleinere. Der Reservierungsschritt terminiert in diesem Fall nach wenigen Versuchen. Nur wenn sich die Auslastungslage in den Batchsystemen während der Reservierungsversuche stets ungünstig ändert, ist es möglich, daß tatsächlich alle Nachrichten (und Co-Reservierungsvarianten) ausprobiert werden müssen. Dies ist sehr unwahrscheinlich, denn dazu muß zunächst bei einem der Cluster-Kandidaten die größere Teilreservierung glücken und alle Teilreservierungen mit dem geringerem Ressourenbedarf bei den anderen Cluster-Kandidaten fehlschlagen. Anschließend muß jeweils einer der Cluster-Kandidaten, bei dem ein Fehlschlag der kleinen Teilreservierung auftrat, eine große Teilreservierung vornehmen können. Derjenige Kandidat, der die große Teilreservierung erfolgreich durchführte, sollte dann eine kleinere nicht mehr vornehmen können. Bei Anfragen mit zwei Teilreservierungen wird somit schlimmstenfalls nur bei einem Cluster-Kandidaten zur zweiten Ebene im Variantenbaum abgestiegen. Dies erklärt, warum die

maximale Nachrichtenanzahl in den Simulationen nur grob linear ansteigen. Nur in jeweils einem viertel oder siebentel aller Fälle wird das aufwendigere Testen des zweiten Teils der Co-Reservierungsvariante durchgeführt.

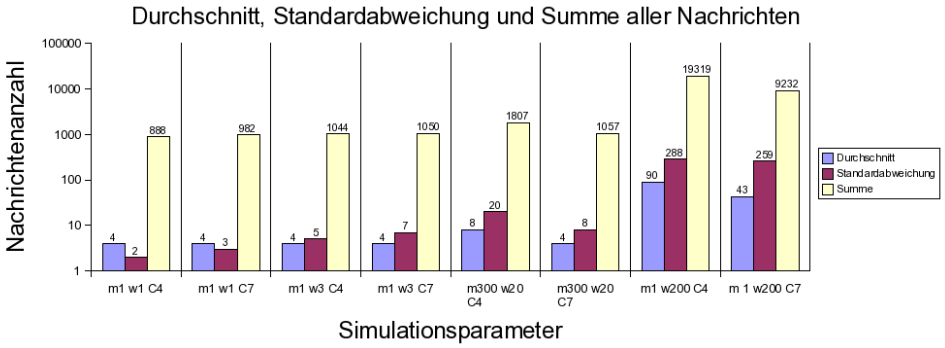
Co-Reservierungsfehlschlagrate:
Bei Simulationen mit sieben Cluster-Kandidaten waren die Co-Reservierungsfehlschlagraten erwartungsgemäß niedriger als bei nur vier Cluster-Kandidaten, denn dort standen mehr Wahlmöglichkeiten pro betrachteter Startzeit für jede Teilreservierung zur Verfügung. Außerdem war dort auch die auf die Cluster verteilte Reservierungslast geringer als bei vier Cluster-Kandidaten.

6.4.5 Schlußfolgerungen

In dieser Untersuchung reichten bereits drei Startzeiten pro Cluster-Kandidat, um niedrige Co-Reservierungsfehlschlagraten zu erzielen. Die betrachteten Reservierungs-Workloads waren jedoch balanciert, es traten keine stoßweisen Anforderungen von Co-Reservierungsanfragen auf. Weiterhin filtern ab Algorithmus 2 die Algorithmen einen Teil der Teilreservierungsvarianten und Algorithmen die Sortiervorgaben beachten benötigen eine Grundlage zur Sortierung. In den weiteren Simulationen werden daher die Parameter *minsize=300, max_zeitfenster=20* verwendet. Eine weitere Optimierungsidee ist, *max_zeitfenster* von der Anzahl der gefundenen Cluster-Kandidaten abhängig zu machen. Bei großen Anzahlen an Cluster-Kandidaten genügen bei jedem wenige Teilreservierungsvarianten. Solange der Variantenbaum nicht zur Berechnung einer optimierten Durchführungsreihenfolge für Co-Reservierungsvarianten herangezogen wird, ist folgende Optimierung einfach umzusetzen. Es werden mehrere Variantenbäume separat für jede Startzeit schrittweise erzeugt. Baumaufbau und Reservierungsschritt wechseln einander ab, bis eine Co-Reservierungsvariante für eine Anfrage reserviert werden kann. Der Baum wird somit nur für die Varianten erzeugt, die tatsächlich durchprobiert werden müssen.



(a) Es sind die Nachrichtenanzahlen der ineffizientesten Anfragebearbeitungen pro Simulation dargestellt. Bei kürzeren Vorausbuchzeiten und Startzeitspannen unterscheiden sich die maximalen Nachrichtenanzahlen nicht (Vgl. Abbildung 19(a)).



(b) Es sind die durchschnittlichen Nachrichtenanzahlen aller Anfragebearbeitungen pro Simulation dargestellt sowie die Standardabweichung und Summe dieser. Auch bei kürzeren Vorausbuchzeiten mit kürzerer Startzeitspanne lassen sich Anfragen meistens effizient bearbeiten.

m1 w1 C4	m1 w1 C7	m1 w3 C4	m1 w3 C7	m300 w20 C4	m300 w20 C7	m1 w200 C4	m1 w200 C7
0.3	0.11	0.08	0.05	0.1	0.03	0.07	0.02

(c) Die Co-Reservierungsfehlschlagraten in den Simulationen. Gegenüber Reservierungs-Workload 1 sind leicht erhöhte Werte festzustellen.

Abbildung 20: Simulationen mit Reservierungs-Workload 3, m=minsize, w=max_zeitfenster und C=Anzahl der Cluster-Kandidaten

6.5 Untersuchung: Reservierungseffizienz bei CPU-Zähl-Algorithmus

6.5.1 Ziel

Einige Anfragen in Simulationen mit Algorithmus 1, die im vorigen Abschnitt untersucht wurden, benötigten sehr viele Nachrichtenübertragungen (siehe Abbildung 20(a)). Ziel dieser Untersuchung ist es festzustellen, ob sich mit dem zweiten Algorithmus, die Anzahl der maximalen Nachrichtenübertragungen in einer Simulation reduzieren lassen.

Erwartung:
Da der zweite Algorithmus vor dem Reservierungsschritt die belegten CPUs in den Batchsystemen der Cluster-Kandidaten zählt, sollten Teilreservierungsfehlschläge wegen der Teilreservierungsvariantenfilterung nur sehr selten mit diesem Algorithmus auftreten.

6.5.2 Durchführung

Verglichen werden Algorithmus 1 und 2 in Simulationen mit Reservierungs-Workload 3, da dort in der ersten Untersuchung im Durchschnitt mehr Nachrichten bei Anfragebearbeitungen nötig waren. Tabelle 8 zeigt die in den Simulationen verwendeten Parameter.

Parameter	untersuchte Wert(e)
verwendete Algorithmen	Algorithmus 1 und 2
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workload	Reservierungs-Workload 3 (Abschnitt 6.3)
minsize	300
max_zeitfenster	20
Anfrage bearbeitende Cluster	4
untersuchtes Kriterium	Reservierungseffizienz

Tabelle 8: Simulationsparameter in Abschnitt 6.5

6.5.3 Resultate

Abbildung 21 zeigt die Anzahl der Teilreservierungsfehlschläge und -rücksetzungen für Co-Reservierungsanfragen (Y-Achse) im zeitlichen Verlauf zu deren Submit-Zeiten (X-Achse). Es ist zu sehen, daß beim trivialen Algorithmus für einige Anfragen viele Teilreservierungsfehlschläge auftreten, bei Algorithmus 2 jedoch nicht. Für Co-Reservierungsanfrage (1) werden für 7200 Sekunden 160 und 192 CPUs benötigt. Bei Algorithmus 1 wurden beispielsweise 30 Teilreservierungen beim ersten Cluster-Kandidaten (BM 0), 34 beim zweiten, 27 beim dritten und 33 beim vierten für diese Anfrage versucht durchzuführen. Jeweils eine Teilreservierung konnte vier mal am Anfang der Startzeitspanne beim ersten Cluster-Kandidaten durchgeführt werden und weitere acht mal jeweils beim dritten Cluster am Ende der Startzeitspanne. Die Anfrage schlug fehl, weil zu keinem Zeitfenster beide Teilreservierungen durchführbar waren.

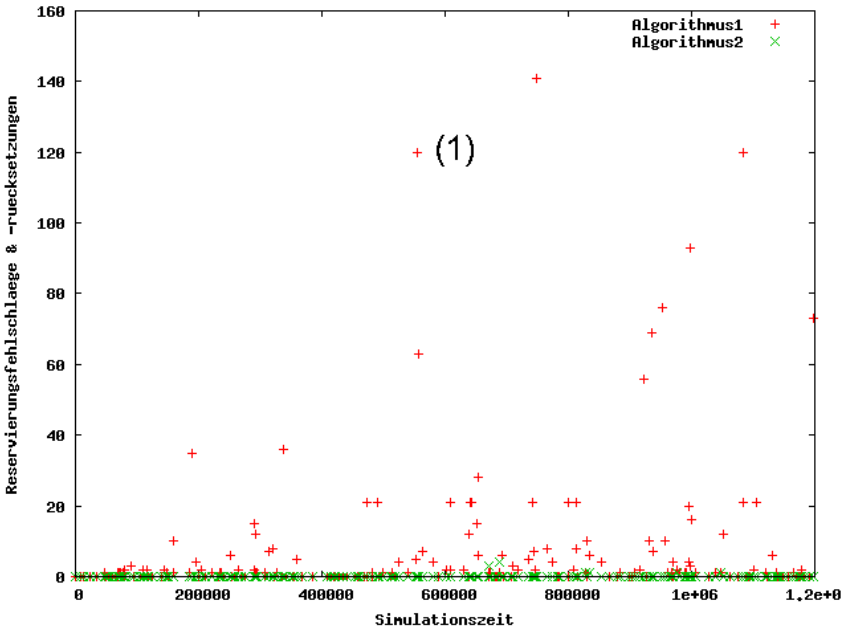


Abbildung 21: Dargestellt ist der Vergleich der Anzahlen fehlgeschlagener Teilreservierungen und -rücksetzungen für die Simulationen mit Algorithmus 1 und 2. Die Nachrichtenanzahlen der Anfragebearbeitungen sind zu den Submit-Zeiten dargestellt. Bei Algorithmus 2 gelingt die Co-Reservierung meist sofort.

Abbildung 22 zeigt die Anzahlen der CPUs, die jederzeit für die nächsten 7200 Sekunden in den Batchsystemen frei sind. Dargestellt ist das Zeitintervall, in dem die Startzeiten von Co-Reservierungsanfrage (1) liegen. Die Abbildung zeigt eine Sicht der Ressourcenverfügbarkeit, die Algorithmus 2 nach allen Statusanfragen erhalten hätte. Es ist zu sehen, daß zu keiner Zeit für beide Teilreservierungen zwei Cluster-Kandidaten mit genügend freien Ressourcen verfügbar sind. Bei BM 0 werden die Ressourcen über die gesamte dargestellte Zeitspanne hinweg durch drei laufende Jobs benötigt. Die Abnahme der freien CPUs ab der Simulationszeit 562000 wird durch eine beginnende Teilreservierung einer anderen Co-Reservierungsanfrage verursacht. Ansteigende Treppen bedeuten frei werdende Ressourcen, die durch das Ende von Teilreservierungen anderer Anfragen oder das Ende von Jobs verursacht werden können. In Tabelle 9 sind die Summen der in den Simulationen benötigten

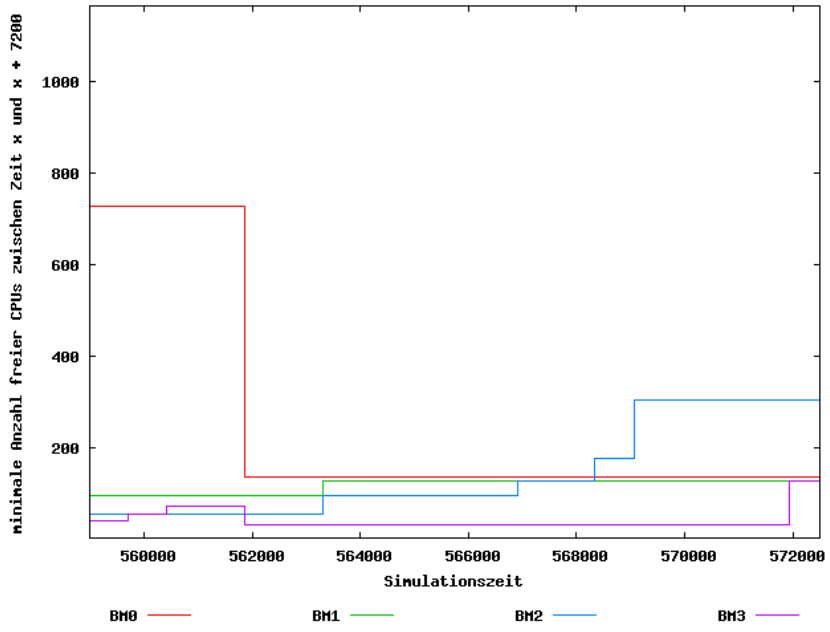


Abbildung 22: Dargestellt sind die Anzahlen an freien CPUs in den Batchsystemen, die für die Dauer von Co-Reservierungsanfrage (1) frei sind. Der X-Achsenausschnitt entspricht Zeiten zwischen frühester und spätester Startzeit. (1) wurde ineffizient bearbeitet, weil nur am Anfang (BM 0) und Ende (BM 2) jeweils für eine der Teilreservierungen genügend Ressourcen frei waren.

Nachrichten für die Verarbeitung aller Co-Reservierungsanfragen dargestellt. Je eine Statusanfrage pro Teilreservierung und Cluster-Kandidat wurde pro Co-Reservierungsanfrage bei Algorithmus 2 zu den Teilreservierungs- und -rücksetznachrichten hinzugezählt. Die Nachrichtenanzahlen sind bei beiden Simulationen ausgeglichen, bei Algorithmus 2 gibt es jedoch nur eine geringe Streuung der Nachrichtenanzahlen pro Anfrage.

6.5.4 Erklärung der Resultate

Im Beispiel von Co-Reservierungsanfrage (1) wurde deutlich, warum es sich lohnt die Ressourcenverfügbarkeit vor der Durchführung von Teilreservierungen zu prüfen. Algorithmus 2 hätte für Co-Reservierungsanfrage (1) nicht eine Co-Reservierungsvariante durchführen müssen,

	Algorithmus 1	Algorithmus 2
Nachrichtenanzahl	2016	2089

Tabelle 9: Summe aller Nachrichten für Simulationen mit Algorithmus 1 und 2 bei Reservierungs-Workload 3

da bei der vorgefundenen Auslastungslage der Batchsysteme keine einzige Aussicht auf Erfolg hatte. Weil Algorithmus 1 diese Heuristik nicht beachtet, wurden die weiteren Versuche durchgeführt. Diese Beobachtung erklärt die niedrige Anzahl der Teilreservierungsfehlschläge und -rücksetzungen für Anfragenbearbeitungen mit Algorithmus 2. Anfragen, wie z.B. Co-Reservierungsanfrage (1), für die die Batchsysteme über längere Zeiten zu hoch ausgelastet sind, werden mit Algorithmus 1 ineffizient bearbeitet. Es werden für diese immer mindestens $max\_zeitfenster \cdot$ *Cluster-Kandidaten* Teilreservierungsversuche durchgeführt.

Ableitung für andere Algorithmen:

Bei Anfragen mit vielen Teilreservierungen und vielen Cluster-Kandidaten müssen bei Algorithmus 1 eventuell sehr häufig Co-Reservierungen fast vollständig durchgeführt werden, bevor der Mangel an freien Ressourcen für die letzte Teilreservierung festgestellt werden kann. Die Beachtung von Mindestabständen zwischen möglichen Startzeiten und die Vorteile einer optimierten Reservierungsreihenfolge der Teile reichen dann nicht aus, um effiziente Co-Reservierungen auch in diesen Fällen sicher zu stellen. Die Zählung der CPUs bringt für Anfragen, bei denen zu keiner Zeit genügend Ressourcen für alle Teile verfügbar sind, eine drastische Senkung der benötigten Nachrichtenanzahlen.

Die Vorabprüfung der Ressourcenverfügbarkeit sollte deshalb Bestandteil weiterer Algorithmen sein, um auch diese Anfragen effizient bearbeiten zu können. Da im Statusanfrageschritt bei den folgenden Algorithmen noch mehr als die Verfügbarkeit geprüft wird, lassen sich diese Aufgaben zusammen durchführen. Der Nachteil der zusätzlichen Kommunikation vor dem Reservierungsschritt verliert dann an Bedeutung.

6.6 Untersuchung: Beachtung und Mißachtung des Fairnesskriteriums

6.6.1 Ziele

Bei der Anfragebearbeitung mit dem Auftragsbestandszeit-Algorithmus (Algorithmus 3) wird zusätzlich auf Fairness zu lokalen Jobs geachtet. Die Teilreservierungsvarianten, deren Startzeiten vor der Auftragsbestandszeit liegen, werden nicht zulassen. Ziel ist es zu prüfen, ob Algorithmus 3 Co-Reservierungen tatsächlich erst nach wartenden Jobs starten läßt. Auswirkungen von Algorithmus 3 auf die Batchsystem-Auslastungen und die Job-Verzögerungen sowie die Erfolgschancen der Anfragen werden ebenfalls untersucht.

Erwartungen:
Es ist zu erwarten, daß Co-Reservierungsanfragen bei Algorithmus 3 im Durchschnitt einen niedrigeren Unfairness-Grad haben als bei Algorithmus 2. Bei der Bearbeitung von Anfragen mit Algorithmus 3 werden die potentiell unfairen Teilreservierungsvarianten verworfen. Als Nachteil dieser Vorgehensweise wird vermutlich häufiger der Fall eintreten, daß Cluster-Kandidaten gar keine Teilreservierungsvarianten bilden können. Wenn die Anzahl der Cluster-Kandidaten, die Teilreservierungsvarianten bilden, geringer ist, als die Anzahl der Teilreservierungen, dann scheitert die Anfrage. Demnach ist bei Algorithmus 3 mit einer erhöhten Co-Reservierungsfehlschlagrate zu rechnen.
Die Filterungsgrade haben Auswirkungen auf die Berücksichtigungswahrscheinlichkeit eines Cluster-Kandidaten bei der Auswahl einer Co-Reservierungsvariante. Bei einem hohen Filterungsgrad für einen Cluster-Kandidaten existieren wenige Co-Reservierungsvarianten und bei einem niedrigen Filterungsgrad existieren viele. Somit steigt und sinkt die Auswahlwahrscheinlichkeit eines Cluster-Kandidaten vermutlich mit dessen Filterungsgrad. Teilreservierungen werden vermutlich eher bei Cluster-Kandidaten mit niedrigen Auftragsbestandszeiten durchgeführt, weil dort niedrigere Filterungsgrade für die Anfragen zu erwarten sind. Die in der Job-Simulation niedrig ausgelasteten Batchsysteme werden deshalb vermutlich mehr Reservierungslast tragen. Wegen der erwarteten höheren Co-Reservierungsfehlschlagrate werden vermutlich auch die maximalen Wartezeiten und die durchschnittlichen Expansion-Faktoren sinken, insbesondere bei den Batchsystemen mit hohen durchschnittlichen Auftragsbestandszeiten.

6.6.2 Durchführung

Vier Cluster-Kandidaten können in den Simulationen Anfragen bearbeiten. Untersucht werden Simulationen mit den Reservierungs-Workloads 1, 2 und 3. Von Workload 1 zu 3 werden die größten Vorausbuchzeiten der Anfragen kürzer, so daß in den Simulationen vermutlich Effekte unterschiedlich hoher Auftragsbestandszeiten betrachtet werden können.
Tabelle 10 gibt eine Übersicht über die Parameter in den durchgeführten Simulationen.

6.6.3 Resultate und Erklärungen

In diesem Abschnitt folgen abwechselnd Resultate und Erklärungen zu den untersuchten Kriterien.

Resultate zur Co-Reservierungsfehlschlagrate:
Alle fehlgeschlagenen Co-Reservierungsanfragen scheiterten in den Simulationen, weil für

Parameter	untersuchte Wert(e)
verwendete Algorithmen	Algorithmus 2 und 3
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workloads	Reservierungs-Workloads 1-3 (Abschnitt 6.3)
minsize	300
max_zeitfenster	20
Anfrage bearbeitende Cluster	4
untersuchte Kriterien	Co-Reservierungsfehlschlagrate durchschnittliche Batchsystem-Auslastung maximale Job-Verzögerung Unfairness-Grad von Co-Reservierungsanfragen durchschnittlicher Expansion-Faktor

Tabelle 10: Simulationsparameter in Abschnitt 6.6

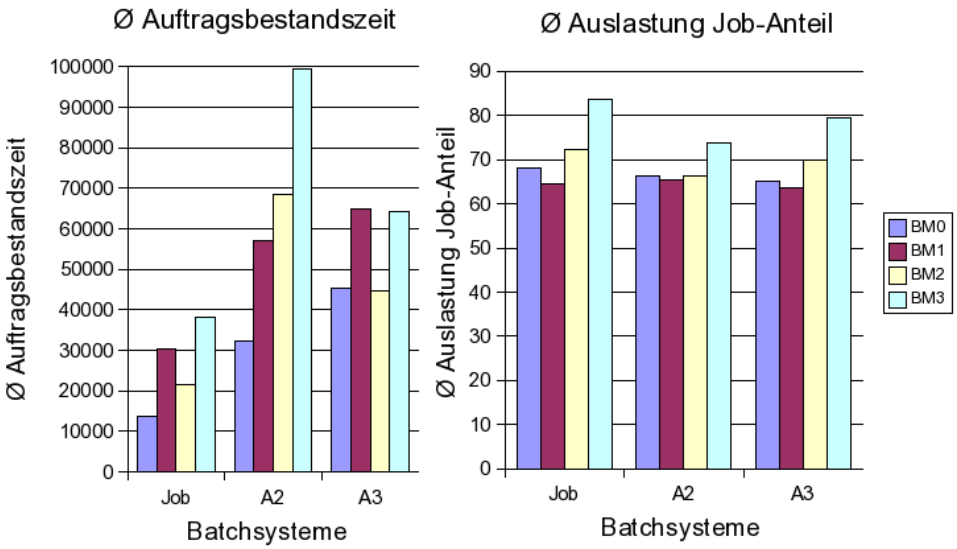
sie keine Co-Reservierungsvarianten gebildet wurden. In Tabelle 11 sind die Co-Reservierungsfehlschlagraten und die Hilfsmetriken durchschnittlicher Filterungsgrad und die Undurchführbarkeitsrate dargestellt. Folgende Resultate ließen sich beobachten: bei langen Startzeitspannen (Reservierungs-Workload 1 und 2) wurden mit Algorithmus 2 93 Prozent aller Anfragen reserviert. Bei Algorithmus 3 scheiterten stets über die Hälfte der Anfragen, bei Anfragen in Reservierungs-Workload 3 sogar 75 Prozent. Bei Simulationen mit Algorithmus 3 war die Undurchführbarkeitsrate von Co-Reservierungsanfragen bei den Cluster-Kandidaten um etwa 0.5 größer als bei Simulationen Algorithmus 2. Die Co-Reservierungsfehlschlagraten stiegen in dem Maße an wie bei Cluster-Kandidaten die Undurchführbarkeitsrate anstieg.

Erklärungen:
Die Filterung der Teilreservierungsvarianten wegen des Auftragsbestandszeitkriteriums ist für die hohen Anteile undurchführbarer Anfragen bei Algorithmus 3 verantwortlich. Da das Auftragsbestandszeitkriterium bei Algorithmus 2 nicht beachtet werden mußte, war dort die Undurchführbarkeitsrate geringer. Bei den hohen Undurchführbarkeitsraten konnten bei Algorithmus 3 deshalb auch häufiger keine Co-Reservierungsvarianten mehr gebildet werden, so daß dort mehr Anfragen scheiterten.

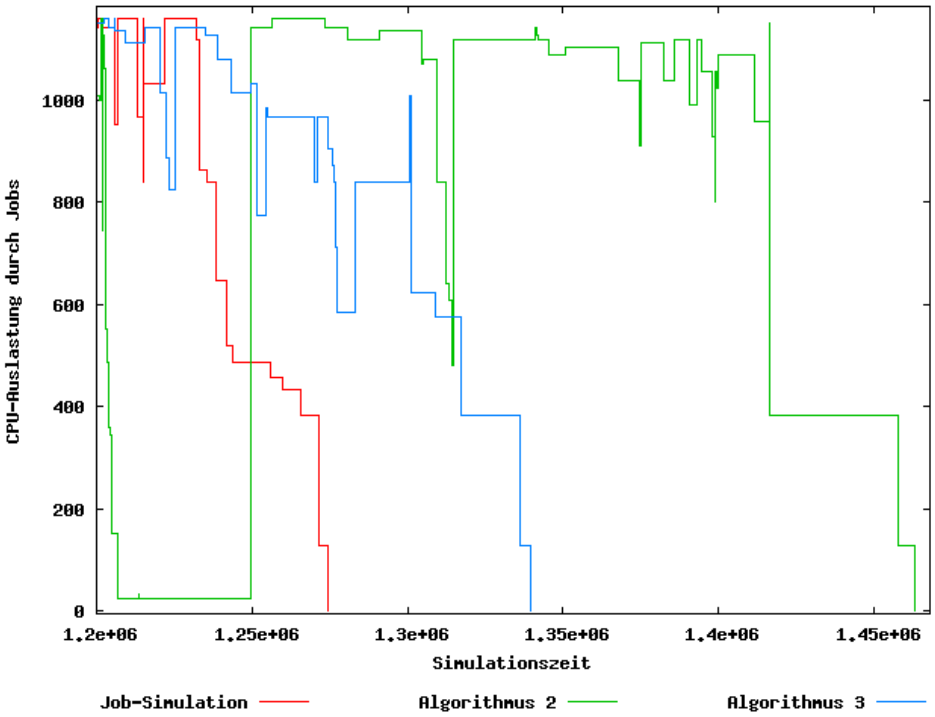
Algorithmus	BM 0	BM 1	BM 2	BM 3
Reservierungs-Workload 1				
ø Filterungsgrad A2	0.20	0.18	0.17	0.18
Undurchführbarkeitsrate A2	0.14	0.16	0.14	0.16
Co-Reservierungsfehlschlagrate A2	0.07			
ø Filterungsgrad A3	0.68	0.72	0.63	0.73
Undurchführbarkeitsrate A3	0.59	0.64	0.54	0.62
Co-Reservierungsfehlschlagrate A3	0.51			
Reservierungs-Workload 2				
ø Filterungsgrad A2	0.32	0.31	0.24	0.31
Undurchführbarkeitsrate A2	0.21	0.22	0.17	0.22
Co-Reservierungsfehlschlagrate A2	0.07			
ø Filterungsgrad A3	0.71	0.83	0.78	0.87
Undurchführbarkeitsrate A3	0.66	0.74	0.64	0.75
Co-Reservierungsfehlschlagrate A3	0.60			
Reservierungs-Workload 3				
ø Filterungsgrad A2	0.34	0.30	0.31	0.41
Undurchführbarkeitsrate A2	0.29	0.25	0.23	0.33
Co-Reservierungsfehlschlagrate A2	0.11			
ø Filterungsgrad A3	0.84	0.87	0.86	0.88
Undurchführbarkeitsrate A3	0.78	0.80	0.77	0.78
Co-Reservierungsfehlschlagrate A3	0.75			

Tabelle 11: Durchschnitt der Filterungsgrade und Undurchführbarkeitsraten für Simulationen mit Reservierungs-Workload 1-3 (R1 - R3) und Algorithmen 2 und 3 (A2, A3) bei den Batchsystemen (BM 0 bis 3)

Batchsystem-Auslastung und Auftragsbestandszeit:
Abbildung 23(a) zeigt die durchschnittliche Auslastung und die durchschnittliche Auftragsbestandszeit der Batchsysteme in den Simulationen mit Reservierungs-Workload 3 für Algorithmen 2 und 3 und für die reine Job-Simulation. In der Simulation mit Algorithmus 3 ist der Anteil der durchschnittlichen Batchsystem-Auslastung durch Jobs annähernd der in der Job-Simulation aufgetretene. In der Simulation mit Algorithmus 2 sank der Job-Anteil an der durchschnittlichen Auslastung gegenüber der Job-Simulation, in Batchsystem 3 am deutlichsten mit über 10 Prozent. Gleichzeitig stieg in Batchsystem 3 die durchschnittliche Auftragsbestandszeit auf über einen Tag. Dieser Effekt trat auf, weil viele Jobs bis weit nach Ende des Job-Submit-Stops (1209600 Sekunden) verzögert wurden. Weil sie in der Auslaufphase der Simulation liefen, wurden sie nicht mehr in der Auslastungsberechnung berücksichtigt. Abbildung 23(b) zeigt dies an den Verläufen der durch Jobs verursachten Batchsystem-Auslastung. Die Y-Achse zeigt die durch Jobs belegten CPUs, die X-Achse die Simulationszeit. In der Simulation mit Algorithmus 2 liefen Jobs auch 3 Tage nach dem Submit-Stop noch. Eine große Reservierung lief dort außerdem kurz nach Ende der Simulationszeit und verzögerte diese Jobs zusätzlich. Bei Algorithmus 3 und der Job-Simulation endete der letzte Job schon nach der Hälfte bzw. einem Drittel der Auslaufzeit von Algorithmus 2.



(a) Dargestellt ist die durchschnittliche Auftragsbestandszeit und der durchschnittliche prozentuale Anteil der Job-Last an der gesamten Batchsystem-Auslastung. Bei Algorithmus 3 tritt eine kleinere Streuung der Auftragsbestandszeit auf und mehr Jobs wurden vor Submit-Stop (bei 1209600 Sekunden) verarbeitet.



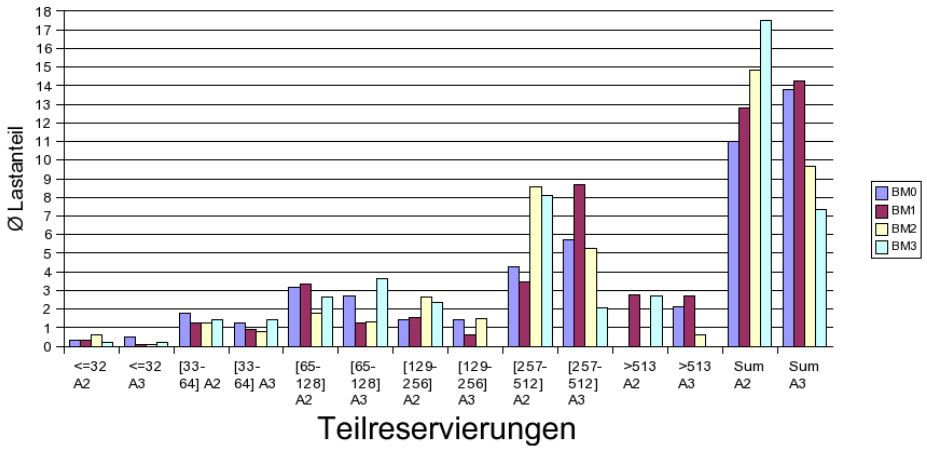
(b) Job-Anteil an den Batchsystem-Auslastungen von Batchsystem 3 in den Auslaufphasen der Simulationen

Abbildung 23: Durchschnittliche Batchsystem-Auslastungen und Auslastungen zum Simulationsende

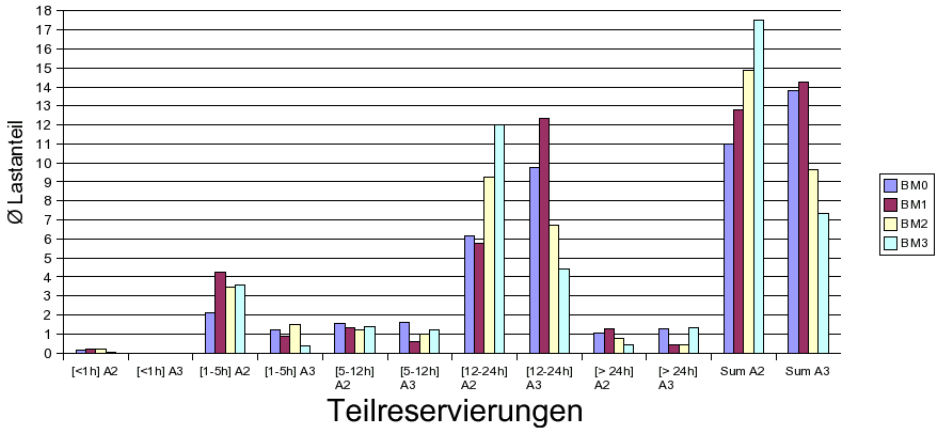
Erklärungen:
Die bei Algorithmus 3 sehr viel früher auslaufenden Jobs profitierten von der Auftragsbestandszeitfilterung. Algorithmus 3 ließ wegen der hohen Auftragsbestandszeit durch vielen wartenden Jobs keine Teilreservierungen zu. Zusammengezählt ergäbe die durchschnittliche Auslastung der Job-Simulation und der Reservierungsanteil von Algorithmus 2 bei Reservierungs-Workload 3 einen Wert von über 100 Prozent. Selbst Auslastungswerte von 100 Prozent lassen sich in der Praxis bei Workloads mit Mehr-Prozessor-Jobs nicht erreichen. Wenn wartende Jobs zu viele Prozessoren anfordern oder die Jobs mit Vorausreservierungen kollidieren, bleiben oft einige CPUs ungenutzt. Deshalb dauerte es in der Simulation mit Algorithmus 2 so lange, bis auch der letzte Job endete. In realen Umgebungen gibt es keinen Submit-Stop, so daß die in der Simulation auslaufenden Jobs noch viel länger verzögert werden würden. Besonders bei Algorithmus 2 könnte dies zu sehr langen Wartezeiten führen.

Batchsystem-Auslastung (Verteilung der Reservierungslast):
Die Verteilungen der Reservierungslasten für die Algorithmen 2 und 3 sind in den Abbildungen 24(a) und 24(b) für die Simulationen mit Reservierungs-Workload 3 zu sehen. Die Y-Achsen zeigen die Anteile, die Teilreservierungen prozentual an den durchschnittlichen Batchsystem-Auslastungen hatten. Die Teilreservierungen sind in Abbildung 24(a) nach der Anzahl der angeforderten CPUs und in Abbildung 24(b) nach deren Dauer gruppiert. Sowohl große, als auch lange dauernde Teilreservierungen wurden bei Algorithmus 2 in Batchsystem 3 vorgenommen. Auch insgesamt mußte damit das Batchsystem mit der durchschnittlich höchsten Auslastung durch Jobs bei Algorithmus 2 die höchste Reservierungslast tragen. Bei Algorithmus 3 trugen die Batchsysteme 2 und 3, die eine hohe durchschnittliche Auslastung durch Jobs hatten, insgesamt weniger Reservierungslast. Batchsystem 3 trug sogar nur noch etwa die Hälfte der Reservierungslast, die anderen Batchsysteme 10 bis 30 Prozent mehr. Der Großteil der gesamten Reservierungslast wurde durch wenige Co-Reservierungen verursacht. Jeweils nur 10 Teilreservierungen verursachen die hohe Last in der Kategorie von 12 bis 24 Stunden bei A2 und BM3 sowie A3 und BM1. Die Auswirkungen der Auftragsbestandszeit sind eher in den Kategorien mit geringerer Dauer festzustellen, da dort höhere Filterungsgrade für die Cluster-Kandidaten bei Anfragen erreicht werden. Bei Co-Reservierungsanfragen der Dauerkategorie von einer bis fünf Stunden verteilte sich die Reservierungslast bei Algorithmus 3 so, wie dies anhand ihrer durchschnittlichen Auftragsbestandszeiten zu erwarten war. So hatten BM1 und BM3 mit hohen durchschnittlichen Auftragsbestandszeiten auch weniger Reservierungslast in dieser Kategorie als BM0 und BM2.

Erklärungen:
Batchsystem 3 mußte trotz der hohen Auslastung durch Jobs bei Algorithmus 2 viel Reservierungslast tragen. Dies liegt an den lange andauernden Co-Reservierungen, die ihre frühesten Zeitfenster vermutlich nach laufenden Jobs hatten. Damit würde der Algorithmus alle Cluster-Kandidaten gleichwertig bei der Co-Reservierungsvariantenwahl behandeln, da jeder gleich viele Teilreservierungsvarianten ermöglicht. Bei Anfragen mit Algorithmus 3 gilt dies für die lange andauernden Co-Reservierungen ebenso, da viele Startzeiten nach der Auftragsbestandszeitgrenze liegen. Daher trug Batchsystem 1 hier trotz der überdurchschnittlich hohen durchschnittlichen Auftragsbestandszeit viel Reservierungslast. Bei kurzen Co-Reservierungen richtete sich die Verteilung mehr an den unterschiedlichen Filterungsgraden aus, die durch die unterschiedlichen Auftragsbestandszeiten verursacht wurden. Der erwartete Lastverteilungseffekt trat ein.



(a) nach CPU-Größen gruppiert



(b) nach Dauer gruppiert

Abbildung 24: Prozentuale durchschnittliche Batchsystem-Auslastung (Reservierungs-Anteil)

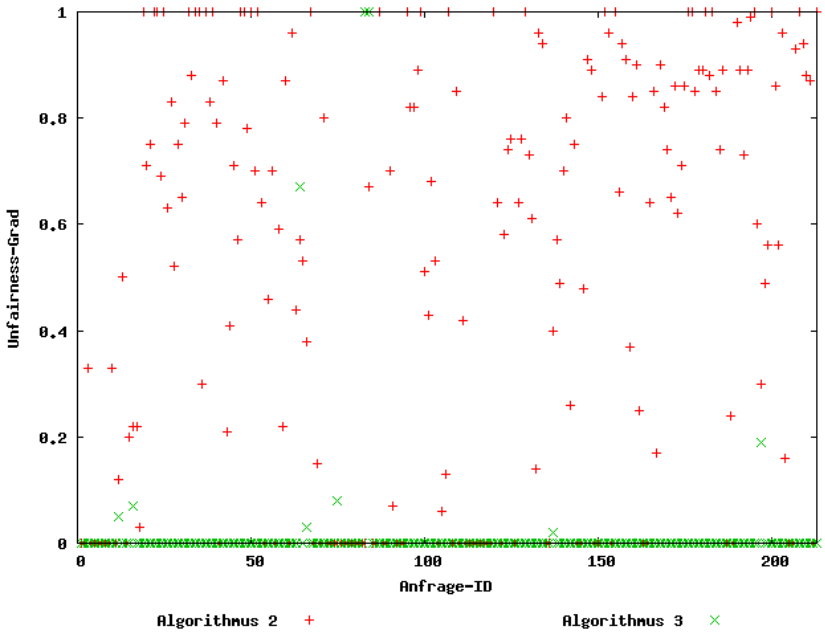
Maximale Job-Verzögerung:
In Tabelle 12 sind die Wartezeiten der am längsten wartenden Jobs in den Simulationen mit Reservierungs-Workload 3 dargestellt. Der am längsten wartende Job startete bei Algorithmus 2 erst nach fast sechs Tagen, bei Algorithmus 3 nach drei Tagen.

	Algorithmus 2	Algorithmus 3
Längster wartender Job	499859 sec	261616 sec
ø der 10 am längsten wartenden Jobs	337874 sec	219572 sec

Tabelle 12: Längste Wartezeiten von Jobs bei Simulationen mit Reservierungs-Workload 3 und Algorithmus 2 und 3

Erklärungen:
Wie bereits an dem auslaufenden Workload in Abbildung 23(b) zu sehen war, stauen sich bei Algorithmus 3 weniger wartende Jobs. Co-Reservierungsanfragen werden bei Algorithmus 3 bei hohen Auftragsbeständen häufiger abgelehnt, anstatt dessen können wartende Jobs schneller verarbeitet werden. Algorithmus 2 beachtet dies nicht, deshalb traten dort lange Wartezeiten auf.

Unfairness-Grad:
Abbildung 25(a) zeigt die Unfairness-Grade der Anfragen bei Reservierungs-Workload 3. Bei Algorithmus 2 wird die Fairness zu wartenden Jobs häufig verletzt, bei Algorithmus 3 nahezu immer eingehalten. Mittelwerte und Standardabweichungen der durchschnittlichen Unfairness-Grade der Simulationen sind in Abbildung 25(b) dargestellt. Bei Reservierungs-Workload 3 und Algorithmus 2 startet durchschnittlich jeder zweite bei Reservierungs-Submit wartende Job erst nach der Teilreservierung. Algorithmus 3 beachtet die Fairness in fast allen Fällen. In einer Extra-Simulation (Algorithmus 3*) mit Reservierungs-Workload 3, bei denen in der Auftragsbestandszeitberechnung Job-Laufzeiten nicht mit der WallClock-Genauigkeit abgeschätzt wurden, wurde die Fairness stets eingehalten. Die Co-Reservierungsfehlschlagrate lag bei Algorithmus 3* gegenüber Algorithmus 3 jedoch um 35 Prozent höher.



(a) Unfairness-Grad der Anfragen bei Reservierungs-Workload 3

Reservierungs-Workload 1	Mittelwert	Standardabweichung
Algorithmus 2	0.33	0.34
Algorithmus 3	0.02	0.09
Reservierungs-Workload 2	Mittelwert	Standardabweichung
Algorithmus 2	0.43	0.38
Algorithmus 3	0.01	0.06
Reservierungs-Workload 3	Mittelwert	Standardabweichung
Algorithmus 2	0.50	0.40
Algorithmus 3	0.01	0.11
Algorithmus 3*	0.00	0.00

(b) Durchschnitt und Standardabweichung der Unfairness-Grade: Nicht alle wartenden Jobs werden bei Algorithmus 3 fair behandelt. Die Simulation Algorithmus 3* wurde mit pessimistischer Berechnung der Auftragsbestandszeit durchgeführt (WallClockzeiten wurden statt geschätzter Job-Laufzeiten verrechnet).

Abbildung 25: Unfairness-Grade in den Simulationen

Erklärungen:
Bei Reservierungs-Workload 3 liegen die Vorausbuchzeiten von Co-Reservierungsanfragen durchschnittlich den Submit-Zeiten am nächsten, deshalb ist bei Algorithmus 2 der Unfairness-Grad dort am höchsten. Bei Algorithmus 3 konnte in allen Simulationen die Fairness nicht für jede Anfrage eingehalten werden, entweder führten zwischen Submit und Start der Co-Reservierung übertragene Jobs zu einer zusätzlichen Verzögerung der wartenden Jobs oder die Laufzeitabschätzungen wurden für sie zu niedrig abgeschätzt.

Durchschnittlicher Expansion-Faktor:
In Abbildung 26 sind die durchschnittlichen Expansion-Faktoren der Simulationen mit Reservierungs-Workload 3 für verschiedene CPU-Anzahl-Kategorien der Jobs dargestellt. In allen Simulationen waren die Expansion-Faktoren kleinerer Jobs geringer als die der größeren. In den Batchsystemen 0 und 1, die bei Algorithmus 3 mehr Reservierungslast trugen, wurden Jobs im Durchschnitt stärker verzögert als bei Algorithmus 2. Ihre durchschnittlichen Expansion-Faktoren stiegen um etwa 50 Prozent. In den Batchsystemen 2 und 3, die weniger Reservierungslast zu tragen hatten, sanken hingegen die durchschnittlichen Expansion-Faktoren.

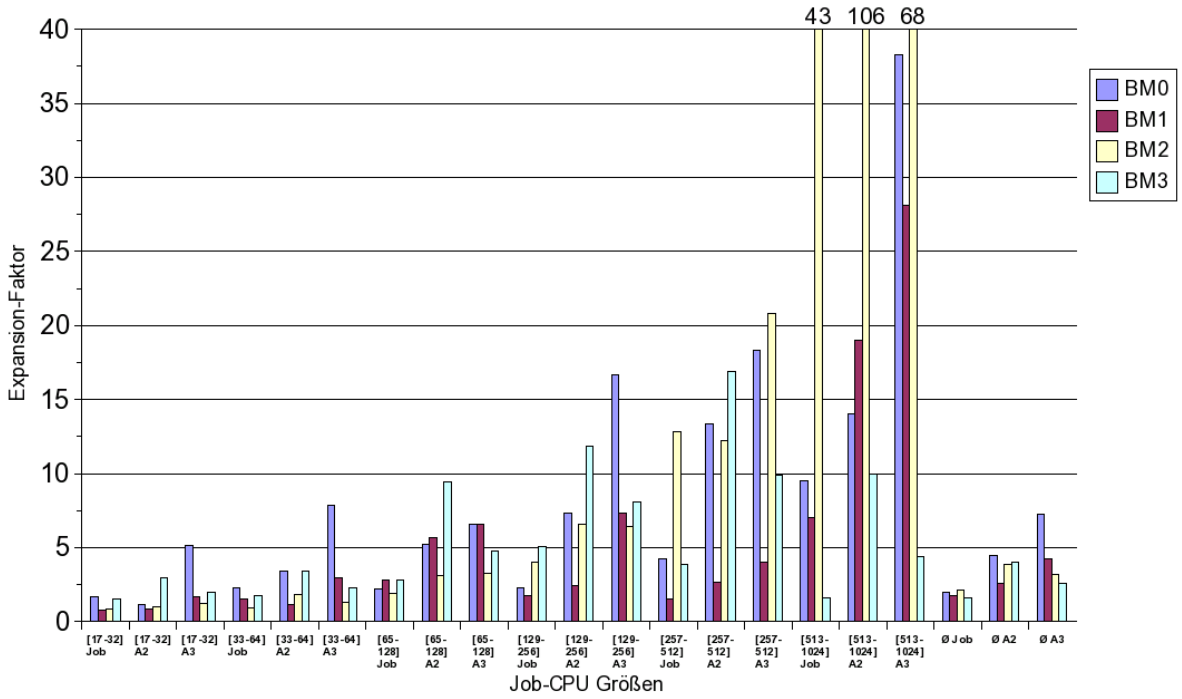


Abbildung 26: Expansion-Faktoren bei Job-Simulation, Reservierungs-Workload 3 für Algorithmus 2 und 3

Erklärungen:
Eine Ursache für niedrigere Expansion-Faktoren kleinerer Jobs ist, daß diese öfter als Backfill-Jobs eingeplant werden konnten und größere dadurch verzögert wurden. Größere Jobs benötigen oft eine spezielle Reservierung durch den Batchsystem-Scheduler, um überhaupt ausgeführt werden zu können. In den Simulationen konnte jeweils nur ein wartender Job eine

solche Reservierung erhalten. Anderen wartenden Jobs konnten stets durch Jobs mit geringerem Ressourcenbedarf und Teilreservierungen ein Teil ihrer benötigten Ressourcen wieder entzogen werden, so daß diese weiter verzögert wurden. Generell führte das Sinken der Reservierungslast durch die höhere Co-Reservierungsfehlschlagrate bei Algorithmus 3 dazu, daß Jobs weniger verzögert wurden und die durchschnittlichen Expansion-Faktoren abnahmen, da die Jobs öfter die Ressourcen nutzen konnten, die sonst Teilreservierungen vorbehalten waren.
In der Simulation mit Algorithmus 3 traten jedoch Effekte auf, die bei Batchsystem 0 und 1 gegenteiliges verursachten. Die Reservierungslastumverteilung von den Batchsystemen mit hohem Job-Anteil an der durchschnittlichen Batchsystem-Auslastung zu denen mit niedrigerem Anteil ist zum Teil dafür verantwortlich. Teilreservierungen entzogen den wartenden Jobs von BM0 häufiger ihre angeforderten Ressourcen. Nur weil BM0 in der Job-Simulation eine niedrigere durchschnittliche Auslastung als BM2 und BM3 hatte, war es nicht unbedingt geeignet, um die zusätzliche Reservierungslast aufzunehmen. Snell et al. behaupten, daß ein proportionaler Zusammenhang zwischen durchschnittlicher Auslastung und dem Verhältnis der durchschnittlichen Job-Größe zur totalen Ressourcenanzahl eines Batchsystems besteht [4]. Batchsysteme, die eher kleinere Jobs als größere haben, können die Ressourcen effizienter nutzen. Denn bei großen Jobs reicht die verbleibende Anzahl freier CPUs oft nicht aus, um die minimalen CPU-Anforderungen des kleinsten wartenden Jobs zu erfüllen. Eine zusätzliche Last führt demnach bei Batchsystemen mit größeren Jobs statt zu einer höheren Auslastung eher zu einer höheren Job-Verzögerung. Die Batchsysteme 0 und 1 besitzen im Vergleich zu den anderen Batchsystemen eher größere Jobs (Abbildung 16). Ihre hohen durchschnittlichen Expansion-Faktoren bei Algorithmus 3 können durch diese These erklärt werden.
Eine weitere Überlegung ist, daß sich bei Algorithmus 3 der Auftragsbestand in den Batchsystemen durch spätere Startzeiten nicht vermindert, sofern diese trotzdem durchgeführt werden können. Zu einem späteren Zeitpunkt kann eine höhere Job-Last im Batchsystem vorhanden sein und die Teilreservierung zu dieser Zeit einen negativeren Einfluß haben als zur einer früheren Zeit.

6.6.4 Schlußfolgerungen

Die Beachtung der Job-Fairness ist ein wichtiger Aspekt für die Co-Reservierungsalgorithmen. Ohne deren Berücksichtigung könnten sie die Scheduling-Regeln der Batchsysteme zu Ungunsten der lokalen Benutzer umgehen. Die Akzeptanz eines Co-Reservierungsdienstes, der nur den CPU-Zähl-Algorithmus verwendet, wäre daher sehr gering. Die grobe Abschätzung der WallClock-Genauigkeiten genügte in den meisten Fällen, um die Fairness bei der Anfragebearbeitung zu allen wartenden Jobs zu gewährleisten. Der Mechanismus von Algorithmus 3 kann noch an einigen Stellen verbessert werden. Ein Ziel weiterer Algorithmen könnten Maßnahmen zur Senkung der Co-Reservierungsfehlschlagrate sein, die bei Algorithmus 3 in den Simulationen sehr hoch war. Eine weitere Untersuchung könnte verschiedene Skalierungen der WallClock-Ungenauigkeit bei der Auftragsbestandszeitberechnung betrachten. Es könnte tiefergehend untersucht werden, bei welcher Skalierung welche Trade-Offs zwischen Erfolgswahrscheinlichkeit der Co-Reservierungen und dem Anteil unfair behandelter wartender Jobs möglich sind. Ebenso kann ein besserer Lastverteilungs- und Zugangskontrollmechanismus entwickelt werden, mit dem die durchschnittlichen Job-Wartezeiten in den

Batchsystemen besser berücksichtigt werden. Denkbar ist eine generelle Zugangskontrolle, mit der Anfragen unabhängig von deren Vorausbuchzeit angenommen oder abgelehnt werden können. Außerdem könnten bei Batchsystemen mit eher kleineren Jobs Teilreservierungen bevorzugt werden.

6.7 Untersuchung: Co-Reservierungsfehlschlagrate beim Backfill-Algorithmus

6.7.1 Ziel

In der vorangegangenen Untersuchung wurde festgestellt, daß viele Co-Reservierungsanfragen mit Algorithmus 3 nicht erfolgreich verliefen (Tabelle 11 Seite 69). In den für Algorithmus 3 durchgeführten Simulationen schlugen mehr als die Hälfte aller Anfragen fehl, obwohl die Batchsysteme nur etwa zu durchschnittlich 80 Prozent ausgelastet waren.
In dieser Untersuchung wird überprüft, ob sich die Co-Reservierungsfehlschlagraten mit dem Backfill-Algorithmus (siehe Kapitel 4.5) bei den selben Reservierungs-Workloads tatsächlich senken lassen. Außerdem wird untersucht, welche Eigenschaften zusätzlich durchgeführte Co-Reservierungen haben.

Erwartung:
Ein Teil der ungenutzten Kapazitäten in den Batchsystemen hätte möglicherweise durch Backfill-Reservierungen genutzt werden können. Daher sind geringere Co-Reservierungsfehlschlagraten in den Simulationen mit dem Backfill-Algorithmus zu erwarten.

6.7.2 Durchführung

Die Simulationen mit dem Auftragsbestandszeit-Algorithmus aus Abschnitt 6.6 werden mit dem Backfill-Algorithmus durchgeführt. Tabelle 13 zeigt die verwendeten Parameter der durchgeführten Simulationen.

Parameter	untersuchte Wert(e)
verwendete Algorithmen	Algorithmus 3 mit und ohne Backfill-Reservierungen
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workloads	Reservierungs-Workload 1-3 (Abschnitt 6.3)
minsize	300
max_zeitfenster	20
Anfrage bearbeitende Cluster	4
untersuchte Kriterien	Co-Reservierungsfehlschlagrate CPU-Anzahl und Dauer von Teilreservierungen

Tabelle 13: Simulationsparameter in Abschnitt 6.7

6.7.3 Resultate

Die Eigenschaften, der bei dem Backfill-Algorithmus zusätzlich durchgeführten Co-Reservierungsanfragen, sind in Tabelle 14 dargestellt. Die Mittel- und Maximalwerte sowie die

Standardabweichungen der CPU-Anzahl- und die Dauer-Verteilung sind darin enthalten. Außerdem sind die Anzahlen der zusätzlich durchführbaren Co-Reservierungsanfragen und die Unterschiede der Co-Reservierungsfehlschlagraten des Backfill-Algorithmus im Vergleich zu Algorithmus 3 dargestellt.
Mehrstündige Co-Reservierungen mit Teilreservierungen von über 100 CPUs konnten beim Backfill-Algorithmus zusätzlich zu den bei Algorithmus 3 durchgeführten vorgenommen werden. Die Co-Reservierungsfehlschlagrate sank bei dem Backfill-Algorithmus in jeder Simulation um mindestens 12 Prozent gegenüber Algorithmus 3. Einige Anfragen konnten beim Backfill-Algorithmus nicht mehr reserviert werden.

	Mittelwert	Standardabweichung	Maximum
CPUs erste Teilreservierung	100	65	264
Dauer	2990	4050	12600
Anfragen nur Algorithmus 3	6		
Anfragen nur Backfill-Algorithmus	29		
$\Delta Rate$	13%		

(a) Reservierungs-Workload 1

	Mittelwert	Standardabweichung	Maximum
CPUs erste Teilreservierung	78	48	176
Dauer	3265	5090	18000
Anfragen nur Algorithmus 3	6		
Anfragen nur Backfill-Algorithmus	27		
$\Delta Rate$	12%		

(b) Reservierungs-Workload 2

	Mittelwert	Standardabweichung	Maximum
CPUs erste Teilreservierung	104	153	888
Dauer	5700	8965	42600
Anfragen nur Algorithmus 3	3		
Anfragen nur Backfill-Algorithmus	34		
$\Delta Rate$	15%		

(c) Reservierungs-Workload 3

Tabelle 14: Eigenschaften beim Backfill-Algorithmus zusätzlich durchgeführter Co-Reservierungsanfragen, $\Delta Rate$ = Prozentuale Senkung der Co-Reservierungsfehlschlagrate bei Backfill-Algorithmus

Tabelle 15 zeigt den Vergleich der Undurchführbarkeitsraten der Simulationen. Wenn Backfill-Reservierungen möglich waren, stieg der Anteil von Co-Reservierungsanfragen, zu denen die Cluster-Kandidaten bei Statusanfragen Teilreservierungsvarianten zuließen. Bei den Batch-

systemen 0 und 1 wurden beim Backfill-Algorithmus für etwa 10% der vormals undurchführbaren Anfragen wieder Teilreservierungsvarianten gebildet.

Reservierungs-Workload 1				
	BM 0	BM 1	BM 2	BM 3
Undurchführbarkeitsrate A3	0.59	0.64	0.54	0.62
Undurchführbarkeitsrate A3 BF	0.48	0.57	0.52	0.63
Reservierungs-Workload 2				
Undurchführbarkeitsrate A3	0.66	0.74	0.64	0.75
Undurchführbarkeitsrate A3 BF	0.54	0.63	0.59	0.65
Reservierungs-Workload 3				
Undurchführbarkeitsrate A3	0.78	0.80	0.77	0.78
Undurchführbarkeitsrate A3 BF	0.68	0.71	0.76	0.75

Tabelle 15: Undurchführbarkeitsrate der Batchsysteme (BM 0 bis 3) für Simulationen mit Algorithmus 3 und Backfill-Algorithmus (A3, A3 BF)

6.7.4 Erklärungen zu den untersuchten Kriterien

Die Senkungen der Co-Reservierungsfehlschlagraten beim Backfill-Algorithmus werden durch die niedrigeren Undurchführbarkeitsraten verursacht. Eine Anfrage ist bei dem Backfill-Algorithmus bereits durchführbar, wenn zwei Cluster-Kandidaten Teilreservierungsvarianten zu Backfill-Startzeiten für jeweils eine der beiden Teilreservierungen zulassen. Beim Auftragsbestandszeit-Algorithmus fehlschlagende Anfragen können deshalb beim Backfill-Algorithmus erfolgreich verlaufen.
Die beim Backfill-Algorithmus zusätzlich durchgeführten Anfragen wurden vermutlich in den Batchsystemen 0 und 1 vorgenommen. Die niedrigen Undurchführbarkeitsraten deuten darauf hin, daß sie mehr Backfill-Reservierungen als die anderen beiden Batchsysteme erlaubten. Sie hatten einen aus größeren Jobs bestehenden Job-Mix, der das Backfilling begünstigte.
Bei den Simulationen mit dem Backfill-Algorithmus konnten einige Anfragen nicht mehr durchgeführt werden. Zwei Effekte können dafür verantwortlich sein: Entweder wurden die benötigten Ressourcen aufgrund ungünstiger Plazierungen von anderen Co-Reservierungen belegt oder die Reihenfolgen der Job-Ausführungen änderten sich für sie ungünstig. Im schlimmsten Fall konnten die Plazierungen von Jobs und Teilreservierungen dazu führen, daß für die Anfragen keine Co-Reservierungsvarianten gebildet wurden (siehe Untersuchung von Abschnitt 6.5).

6.7.5 Schlußfolgerungen

Backfill-Reservierungen sind eine geeignete Maßnahmen die Co-Reservierungsfehlschlagrate zu senken. Eine weitere Senkung der Co-Reservierungsfehlschlagrate ist möglich, wenn die Ermittlung von Backfill-Startzeiten genauer durchgeführt wird. Statt wie im Backfill-Algorithmus Startzeiten nur bis zum nächsten Job- oder Reservierungs-Ende zu betrachten, könnte dazu der Schedule auch einige Iterationen weiter berechnet werden (Schedule-Darstellung

siehe Abbildung 8). Falls auch nach weiteren Iterationen noch genügend Ressourcen für eine Teilreservierung verbleiben, können die Backfill-Startzeiten ausgedehnt und entsprechend mehr Teilreservierungsvarianten zugelassen werden.

6.8 Untersuchung: Qualität der Startzeitoptimierung

6.8.1 Ziel

Untersucht wird nun Algorithmus 4, der die Reservierungsanfragen versucht so früh wie möglich durchzuführen.
Bei genügend langer Vorausbuchzeit kann durch Algorithmus 4 meistens die früheste Startzeit reserviert werden, weil dann noch keine anderen Reservierungen und Jobs die Ressourcen beanspruchen. Interessanter ist zu untersuchen, wie häufig Algorithmus 4 die früheste Startzeit von Anfragen reservieren kann, deren frühester Start unmittelbar nach dem Submit liegt und welche durchschnittlichen relativen Startzeitverzögerungen möglich sind.
Modifikationen des CPU-Zähl-Algorithmus und des Auftragsbestandszeit-Algorithmus können als Vergleich herangezogen werden, um die Effizienz der Startzeitoptimierung zu prüfen. Die Dauer und CPU-Anzahlen von Teilreservierungen, die direkt nach Submit beginnen, werden ebenfalls untersucht. Damit soll herausgefunden werden, für welche Anfragen es sich lohnt frühe Startzeiten zu probieren.

Erwartungen:
Wahrscheinlich lassen sich Backfill-Reservierungen nur für kurze und wenig CPUs benötigende Anfragen durchführen, da zu den frühesten Startzeiten die Ressourcen zu häufig durch Jobs benötigt werden und Backfill-Reservierungen nur möglich sind, wenn sie vor dem nächsten Scheduling-Ereignis in den Batchsystemen abgelaufen sind. Die Wahrscheinlichkeit für eine Anfrage eine Backfill-Reservierung durchführen zu können, hängt unter anderem von der Teilreservierungsanzahl ab und der Anzahl der verfügbaren Cluster-Kandidaten. Zu erwarten ist, daß bei Anfragen mit wenigen Teilreservierungen und vielen Cluster-Kandidaten durch Algorithmus 4 mehr Anfragen zu frühen Startzeiten reserviert werden können.

6.8.2 Durchführung

In dieser Untersuchung werden zwei Reservierungs-Workloads verwendet, die von Reservierungs-Workload 1 abgeleitet sind. Der eine verwendet zweiteilige Anfragen, der andere dreiteilige Anfragen [4]. Bei dem Workload mit dreiteiligen Anfragen wird jede dritte Anfrage der Vorlage von Reservierungs-Workload 1 entfernt, damit eine etwa gleiche Reservierungslast in den Simulationen verteilt wird.
Die früheste Startzeit, die Dauer und die Ressourcenparameter von Anfragen werden in den verwendeten Workloads übernommen. Submit-Zeiten und und späteste Endzeiten werden so angeglichen, daß alle Co-Reservierungsanfragen jeweils eine Startzeitspanne von sechs Stunden haben und zur frühesten Startzeit gestellt werden [5]. Diese Workloads sollten einerseits durchführbare Anfragen erlauben und andererseits frühe Startzeiten für die Anfragen ermöglichen, so daß ein unterschiedliches Verhalten der Algorithmen auftreten kann.

[4] Die dritte Teilreservierung ist ein Duplikat der zweiten

[5] Anfragen werden 30 Sekunden vor der frühesten Startzeit gestellt, damit den Algorithmen die früheste Startzeit bei Beginn des Reservierungsschritts noch zur Verfügung steht und die Zeit nicht bereits verstrichen ist.

Es werden pro Algorithmus drei Simulationen durchgeführt. Variiert werden die Parameter: Anzahl der für die Anfragebearbeitung verfügbarer Cluster-Kandidaten *#Cluster*, Anzahl der Co-Reservierungsanfragen *#Anzahl* und Anzahl der Teilreservierungen von Anfragen *#TR*.
Für startzeitoptmimierende Versionen des CPU-Zähl-Algorithmus, Auftragsbestandszeit-Algorithmus und Backfill-Algorithmus (Algorithmus 4) werden zunächst die Co-Reservierungsfehlschlagraten und durchschnittlichen relativen Startzeitdifferenzen untersucht[6]. Die Co-Reservierungsfehlschlagraten werden ebenfalls analysiert, da fehlgeschlagene Anfragen auf generell zu frühe Startzeiten hindeuten. Anschließend wird in einer zweiten Teiluntersuchung analysiert, welche Dauer und CPU-Anzahlen Teilreservierungen haben, die unmittelbar nach der Anfrage beginnen.
In Tabelle 16 sind die Simulationsparameter in einer Übersicht dargestellt.

Parameter	untersuchte Wert(e)
verwendete Algorithmen	startzeitoptimierende Algorithmen 2, 3 und Backfill
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workloads	Modifikation Reservierungs-Workload 1 (Abschnitt 6.3) Anfragen werden erst bei f_start übertragen 6 h Startzeitspanne Reihe 1,2: #Anzahl = original ; Reihe 3: #Anzahl = $2/3 \cdot$ original Reihe 1,2: #TR = 2 ; Reihe 3 #TR = 3
minsize	300
max_zeitfenster	20
Anfrage bearbeitende Cluster	Reihe 1: #Cluster=4; Reihe 2,3: #Cluster=7
untersuchte Kriterien	Co-Reservierungsfehlschlagrate durchschnittliche relative Startzeitdifferenz Zusammensetzung frühester Teilreservierungen

Tabelle 16: Simulationsparameter in Abschnitt 6.8

6.8.3 Resultate: Startzeitdifferenzen und Co-Reservierungsfehlschlagrate

In Abbildung 27 ist ein Protokollausschnitt der Co-Reservierungen von Simulationen mit *#Cluster=4, #TR=2* zu sehen. Die Vorausbuchzeiten durchgeführter Co-Reservierungsanfragen sind zu den Submit-Zeiten dargestellt. Die zweitkleinste Auftragsbestandszeit der Cluster ist für die beiden Algorithmen, in denen die Auftragsbestandszeit berücksichtigt wird, eingezeichnet. Die Co-Reservierungsanfragen von Algorithmus 3 ohne Backfilling konnten nur zu Zeiten vor 980000 laufen, da sonst immer die Auftragsbestandszeit die letzte mögliche Startzeit überschritt. Algorithmus 2 konnte die Co-Reservierungsanfragenen immer zu den Zeiten reservieren, zu denen in zwei Batchsystemen das erste Mal genügend Ressourcen für die jeweiligen Teile frei waren. Co-Reservierungen des Backfill-Algorithmus starteten

[6]Bei der Berechnung der relativen Startzeitdifferenz wird sich für alle Algorithmen auf den f_start Parameter der Anfrage bezogen, nicht auf die früheste Startzeit der gebildeten Co-Reservierungsvarianten.

entweder zur frühesten Startzeit oder zur ersten Startzeit nach der Auftragsbestandszeit, wenn kein Backfilling möglich war. Die beiden Co-Reservierungen des Backfill-Algorithmus um 980000 und 1000000 starten früher als beim CPU-Zähl-Algorithmus. Ohne Backfill-Startzeiten hätten die beiden Co-Reservierungsanfragen nicht durchgeführt werden können. Eine Erklärung für die späteren Startzeiten bei Algorithmus 2 ist, daß er mehr Co-Reservierungen zuließ, so daß zu den beiden frühesten Startzeiten gerade keine freie Ressourcen mehr verfügbar waren. Die Tabellen 17(a) bis 17(c) zeigen die in den Simulationen ermittelten

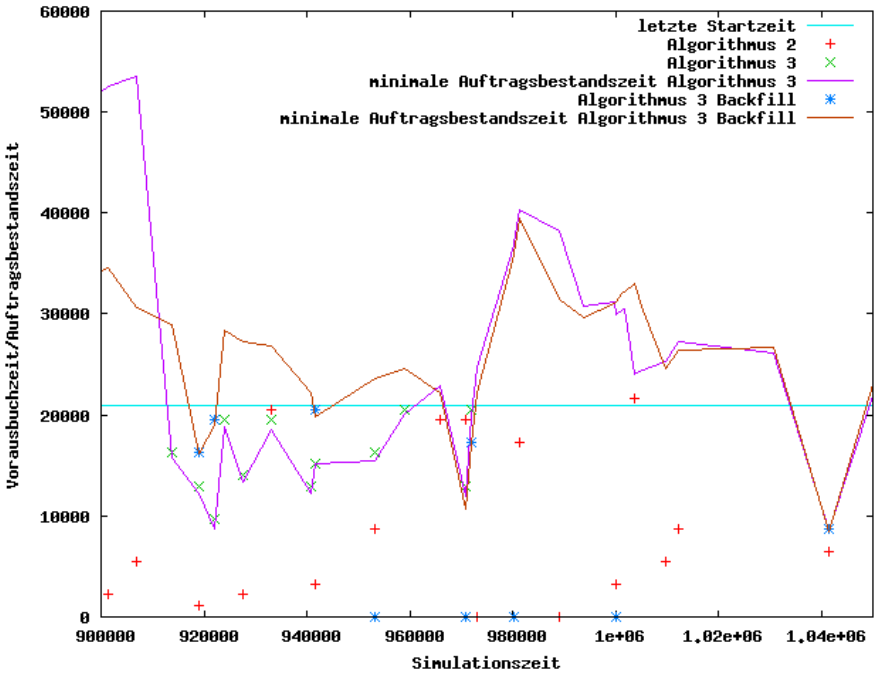


Abbildung 27: Vorausbuchzeiten von Anfragen zur Submit-Zeit dargestellt: Simulationen mit #Cluster=4, #TR=2 (Simulationsreihe 1)

Werte für die Co-Reservierungsfehlschlagrate und die durchschnittliche relative Startzeitdifferenz für jede verwendete Kombination von Algorithmus und Reservierungs-Workload. Für alle Simulationsreihen fällt auf, daß der Backfill-Algorithmus frühere Startzeiten durchführen konnte als Algorithmus 3 und Algorithmus 2 noch früher reservierte. Je mehr Cluster den Anfragen zur Verfügung standen und je weniger Teilreservierungen vorgenommen werden mußten, desto geringer waren die Co-Reservierungsfehlschlagrate und die durchschnittliche relative Startzeitdifferenz. Der Anstieg der Anzahl der Cluster-Kandidaten von vier auf sieben führte beim Backfill-Algorithmus zu einer Senkung der Co-Reservierungsfehlschlagrate um 19 Prozent, bei Algorithmus 3 um nur sechs Prozent und bei Algorithmus 2 zum fast vollständigen Verschwinden von Co-Reservierungsfehlschlägen.

6.8.4 Erklärungen: Startzeitdifferenzen und Co-Reservierungsfehlschlagrate

Bei Algorithmus 2 ist die Regel für das Verwerfen von frühen Teilreservierungsvarianten am lockersten und bei Algorithmus 3 am strengsten. Daher sind die durchschnittlichen relati-

Kriterium	Algorithmus 2	Algorithmus 3	Algorithmus 3 Backfill
Co-Reservierungsfehlschlagrate	0.11	0.53	0.52
ø Relative Startzeitdifferenz	0.23	0.52	0.30

(a) Simulationsreihe 1: #Cluster=4, #TR=2

Messkriterium	Algorithmus 2	Algorithmus 3	Algorithmus 3 Backfill
Co-Reservierungsfehlschlagrate	0.01	0.47	0.33
ø Relative Startzeitdifferenz	0.04	0.47	0.25

(b) Simulationsreihe 2: #Cluster=7, #TR=2

Messkriterium	Algorithmus 2	Algorithmus 3	Algorithmus 3 Backfill
Co-Reservierungsfehlschlagrate	0.03	0.60	0.53
ø Relative Startzeitdifferenz	0.08	0.53	0.29

(c) Simulationsreihe 3: #Cluster=7, #TR=3

Tabelle 17: Durchschnittliche relative Startzeitdifferenzen und Co-Reservierungsfehlschlagraten

ven Startzeitdifferenzen und die Co-Reservierungsfehlschlagraten an diese Reihenfolge gekoppelt. Eine Erklärung für die Unterschiede der Co-Reservierungsfehlschlagraten zwischen Simulationsreihe 1 und 2 ist, daß durch den Backfill-Algorithmus nun noch mehr Co-Reservierungsanfragen zu Backfill-Startzeiten eingeplant werden können, während Algorithmus 3 lediglich von der größeren Auswahl an Cluster-Kandidaten mit eventuell niedrigeren Auftragsbestandszeiten profitiert. Weil bei Algorithmus 2 keine Auftragsbestandszeiten beachtet werden müssen, weisen dessen Simulationen die niedrigsten Co-Reservierungsfehlschlagraten und die niedrigsten durchschnittlichen relativen Startzeitdifferenzen auf. In Simulationsreihe 3 ist eine Teilreservierung zusätzlich zu plazieren, daher sinkt die Chance auf Backfill-Startzeiten und die Wahrscheinlichkeit steigt, daß Teilreservierungsvarianten für die dritte Ressource verworfen werden.

6.8.5 Resultate: Früheste Startzeiten

Nun wird untersucht, für welche Co-Reservierungsanfragen in den Simulationen früheste Startzeiten reserviert werden konnten. In Abbildung 28 wird dies für die drei Simulationsreihen aufgeschlüsselt. Dargestellt sind für die Simulation mit Algorithmus 2 (A2) und den Backfill-Algorithmus (BF) die Teilreservierungsanzahlen der Anfragen, die zur frühesten Startzeit (f_start Parameter der Anfrage) reserviert wurden. Die Teilreservierungen sind nach CPU-Größen und ihrer Dauer gruppiert. Algorithmus 3 wird nicht dargestellt, da für dessen Anfragen keine frühesten Starts erreicht wurden. In Abbildung 27 war dies bereits zu erkennen, denn damit Teilreservierungen die früheste Startzeit erhielten, müßten die Batchsysteme leer sein.
Beim Backfill-Algorithmus liefen in einigen Kategorien mit kurzen Teilreservierungen (< 1h)

mehr Teilreservierungen, als bei Algorithmus 2 aber auch in den Kategorien bis fünf Stunden kommen noch Backfill-Reservierungen vor.
In der ersten und dritten Simulationsreihe konnten insgesamt 20 Prozent aller Anfragen durch den Backfill-Algorithmus zur frühesten möglichen Startzeit reserviert werden, in der zweiten waren es 30 Prozent. Für Algorithmus 2 konnten in den Simulationen 1-3 25, 70 und 50 Prozent aller Anfragen zur frühesten möglichen Startzeit reserviert werden.

6.8.6 Erklärungen: Früheste Startzeiten

In allen Simulationen mußten, um eine früheste Startzeit für eine Anfrage reservieren zu können, genügend freie Ressourcen nach den CPU-Zählungen für sie vorhanden sein. Beim Backfill-Algorithmus konnten weniger Anfragen die früheste Startzeit nutzen, je länger ihre Teilreservierungen dauerten, denn jede endende Teilreservierung und jeder endende Job verhinderte die Bildung von Backfill-Reservierungen in den zugehörigen Batchsystemen.

6.8.7 Schlußfolgerungen

Die Beachtung der Fairness-Bedingungen wirken sich negativ auf den Erfolg von Reservierungen mit frühen Startzeiten aus. Für Anfragen mit kurzer Dauer können sich der Reservierungsversuche zu frühen Startzeiten bei Algorithmus 4 dennoch lohnen. Die Startzeitoptimierung kann auf zwei Arten verbessert werden. Zum einen ist es möglich, daß ein weiter optimierter Backfill-Algorithmus mehr frühe Startzeiten als Backfill-Startzeiten erkennt, zu denen die Auftragsbestandszeit ignoriert werden darf. Zum anderen kann die Auftragsbestandszeit niedriger bemessen werden, so daß Co-Reservierungen auf Kosten geringerer Fairness zu wartenden Jobs früher starten können.

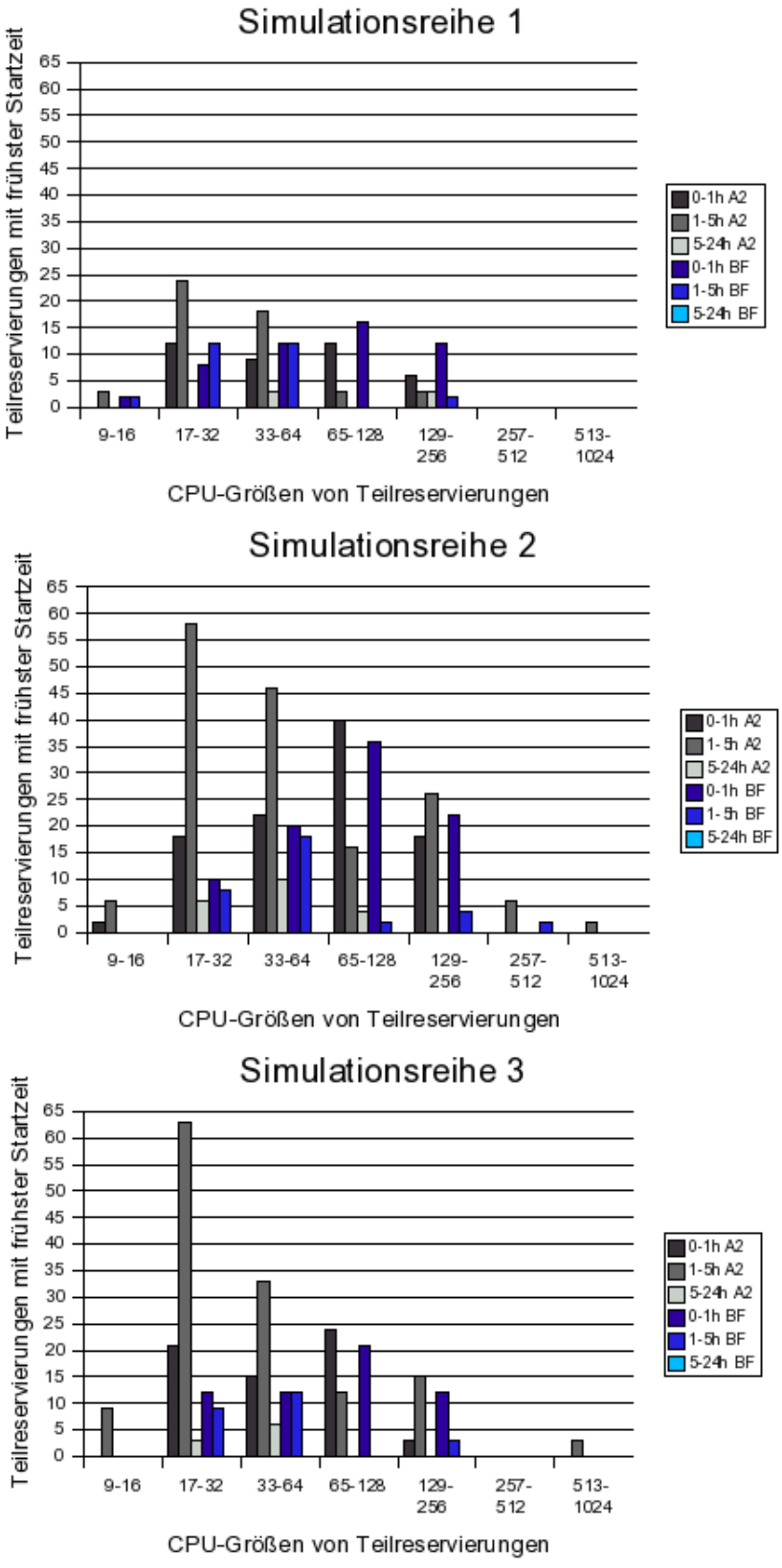


Abbildung 28: Anzahl der Teilreservierungen mit frühester Startzeit

6.9 Untersuchung: Kombinierte Sortierkriterien

In dieser Untersuchung wird Algorithmus 5 in zwei Schritten analysiert. Simulierte Cluster haben in beiden Teiluntersuchungen heterogene Kostentabellen. Als erstes werden die Auswirkungen unterschiedlicher Startzeit- und Kostenfaktoren von Co-Reservierungsanfragen auf die durchschnittliche Startzeit- und Kostendifferenz der Co-Reservierungen untersucht. Dann werden für einen kostenminimierenden Algorithmus die Einflüsse der Co-Reservierungen auf die durchschnittlichen Batchsystem-Auslastungen und Expansion-Faktoren untersucht.

6.9.1 Untersuchung: Start- und Kostendifferenzen

Die Entwicklung eines realistischen Kostenmodells ist nicht Gegenstand dieser Untersuchung. Vielmehr geht es darum zu zeigen, welche Effekte bei der Optimierung von Anfragen nach Kosten und Startzeit auftreten. In den Simulationen haben die Cluster tageszeitabhängige Kosten. Bei jedem Cluster wird die Stundensatztabelle in eine Tages- und Nachtphase eingeteilt. Tagsüber wird für eine CPU-Stunde das Doppelte der Nachtzeit berechnet. Außerdem bieten zwei Cluster ihre Ressourcen an Wochenenden nur noch zum halben Preis der anderen an. Sonst sind die Stundensätze jedes Clusters gleich groß. Für verschiedene Reservierungs-Workloads, in denen Anfragen unterschiedliche Startzeit- und Kostenfaktoren haben, werden die durchschnittlichen relativen Start- und Kostendifferenzen und die durchschnittlichen Gesamtkosten bestimmt.

Erwartungen:
Die Co-Reservierungsanfragen lassen sich in vier Gruppen unterteilen:

- **a)** Werktag: Der Beginn der Startzeitspanne einer Anfrage liegt in der Tageszeit, das Ende in der Nachtzeit, es widersprechen sich Startzeit- und Kostenminimierungsziel.
- **b)** Werktag: Die Überschneidung befindet sich bei Beginn der Tageszeit, die Optimierungsziele ergänzen sich.
- **c)** Werktag: Die Anfrage liegt komplett in der Tages- oder Nachtzeit. Weil die Kosten fix sind, ist die kombinierte Optimierung eine reine Startzeitminimierung.
- **d)** Wochenende: Teilreservierungsvarianten werden bei den günstigen Clustern bevorzugt. Sind zu deren frühen Startzeiten die Ressourcen belegt, dann widersprechen sich Startzeit- und Kostenminimierungsziel.

Bei reiner Kostenminimierung ist zu erwarten, daß von den gebildeten Co-Reservierungsvarianten von Anfragen eine mit im Schnitt mittiger Startzeit und den stets günstigsten Kosten reserviert wird. Wenn gleichzeitig nach frühesten Startzeiten optimiert wird, dann werden bei Anfragen der Typen **b** und **c** neben den kostengünstigsten auch die frühesten Varianten reserviert. Bei höheren Werten des Startzeitfaktors der Anfragen ist mit früheren Startzeiten auch bei Anfragen der Typen **a** und **d** zu rechnen. Gleichzeitig werden bei diesen vermutlich nicht mehr die kostengünstigsten gebildeten Co-Reservierungsvarianten durchgeführt werden können, so daß die durchschnittlichen Gesamtkosten steigen.

Durchführung:
Die Simulationsparameter sind Tabelle 18 zu entnehmen.

Parameter	untersuchte Wert(e)
verwendeter Algorithmus	Algorithmus 5
Batch-Job Workloads	gleiche Job-Workloads wie bei der Job-Simulation Abschnitt 6.2
Reservierungs-Workloads	Reservierungs-Workload 1 (Abschnitt 6.3)
minsize	300
max_zeitfenster	20
Anfrage bearbeitende Cluster	7
Kostentabellen	Alle Cluster: 8-18h 2 Geldeinheiten, 18-8h 1 Geldeinheit, Cluster 1,3: Wochenende halber Preis
Kosten- und Startzeitfaktoren	[(k=1, s=0), (k=0.9, s=0.1), (k=0.8, s=0.2), (k=0.7, s=0.3), (k=0.6, s=0.4), (k=0, s=1)]
untersuchte Kriterien	durchschnittliche Gesamtkosten durchschnittliche relative Startzeitdifferenz durchschnittliche relative Kostendifferenz

Tabelle 18: Simulationsparameter in Abschnitt 6.9

Resultate:
Tabelle 19(a) zeigt für die durchgeführten Simulationen zu welchen Tageszeiten Co-Reservierungen bevorzugt plaziert wurden und wieviel sie durchschnittlich kosteten. In allen Simulationen gab es mehr Co-Reservierungen, die vollständig innerhalb der Nachtzeit liefen, als vollständig zur Tageszeit. Bei reiner Kostenminimierung liefen nur 20 Prozent der Anfragen vollständig innerhalb Tageszeit, über die Hälfte der Anfragen hingegen vollständig zur Nachtzeit. Der Anteil der vollständig nachts liegenden Anfragen verringerte sich bei höheren Startzeitfaktoren. Von 100 auf 90 Prozent Kostenfaktor ist eine stärkere Verringerung der Nachtreservierungen zu beobachten als zwischen 70 und 60 Prozent. Dort stagnieren die Nachtreservierungen. Bei reiner Startzeitminimierung gab es mehr mit der Tages- und Nachtzeit überlappende Reservierungen. Die durchschnittlichen Gesamtkosten stiegen bei Erhöhung des Startzeitfaktors. Im Durchschnitt kostete eine startzeitminimierte Anfrage doppelt so viel wie eine kostenminimierte. Tabelle 19(b) zeigt die Werte der durchschnittlichen relativen Kosten- und Startzeitdifferenzen für die durchgeführten Simulationen. Die Resultate der ersten und zweiten Zeile beziehen sich auf Mittelwerte von sich mit der ersten Wochenendzeit überlagernden Co-Reservierungsanfragen. Für die ersten drei Simulationen konnte immer die günstigste Variante reserviert werden. Bei der Simulation mit Startzeitfaktor 0 lag die durchschnittliche relative Startzeitdifferenz der Anfragen, wie erwartet, mit 0.58 etwa in der Mitte. Die Berücksichtigung von 10 Prozent Startzeitfaktor führte auch bei der durchschnittlichen relativen Startzeitverzögerung zu einer deutlichen Differenz gegenüber der reinen Kostenminimierung. Erst bei 40 Prozent Startzeitfaktor war eine weitere Senkung der durchschnittlichen relativen Startzeitdifferenz zu beobachten. In den Simulationen mit weniger als 30 Prozent Startzeitfaktor wurde immer die günstigste Co-Reservierungsvariante durchgeführt. Bei 30 und 40 Prozent Startzeitfaktor wurden auch leicht teurere Varianten reserviert, insbesondere am preislich stärker variierenden Wochenende. Bei Startzeitminimierung wurde für Anfragen immer die frühest mögliche Startzeit reserviert, preislich lagen sie im Durchschnitt bei knapp 40 Prozent relativer Kostendifferenz.

Kostenfaktor Startzeitfaktor	k = 1 s = 0	k = 0.9 s = 0.1	k = 0.8 s = 0.2	k = 0.7 s = 0.3	k = 0.6 s = 0.4	k = 0 s = 1
ø Gesamtkosten	8698	8846	8872	9428	10440	15935
Reservierungen tags	0.21	0.37	0.36	0.38	0.33	0.28
Reservierungen nachts	0.53	0.50	0.45	0.44	0.44	0.41

(a) Durchschnittliche Gesamtkosten, Anteil vollständig zur Tages- und Nachtzeit gelaufener Co-Reservierungen an allen erfolgreichen Co-Reservierungen

Kostenfaktor Startzeitfaktor	k = 1 s = 0	k = 0.9 s = 0.1	k = 0.8 s = 0.2	k = 0.7 s = 0.3	k = 0.6 s = 0.4	k = 0 s = 1
1. Wochenende ø rel. Startzeitdifferenz	0.59	0.27	0.32	0.27	0.25	0.00
1. Wochenende ø rel. Kostendifferenz	0.00	0.00	0.00	0.02	0.05	0.19
ø rel. Startzeitdifferenz	0.58	0.27	0.27	0.26	0.22	0.00
ø rel. Kostendifferenz	0.00	0.00	0.00	0.01	0.03	0.39
σ rel. Startzeitdifferenz	0.37	0.36	0.36	0.35	0.34	0.00
σ rel. Kostendifferenz	0.00	0.00	0.00	0.04	0.09	0.43

(b) Durchschnitt und Standardabweichung der relativen Kosten- und Startzeitdifferenzen

Tabelle 19: Zeitliche Plazierung der Co-Reservierungen, durchschnittliche Gesamtkosten und Durchschnitt und Standardabweichung der relativen Kosten- und Startzeitdifferenzen

Erklärungen:
Die sprunghafte Senkung der durchschnittlichen relativen Startzeitdifferenz von den Simulationen mit 0 zu 10 Prozent Startzeitfaktor ist auf die Anfragen zurückzuführen, für die Co-Reservierungsvarianten homogene Kosten hatten. Statt der durchschnittlichen Startzeit konnte auf diese Weise die früheste ausgewählt werden. Die Anfragen des ersten Wochenendes der Simulation mit 80 Prozent Kostenfaktor zeigen höhere durchschnittliche relative Startzeitdifferenzen. Eine Erklärung dafür ist, daß Anfragen bei den günstigen Clustern zu reservieren versucht wurden. Wegen der erhöhten Reservierungslast bei ihnen konnten für sie aber erst spätere Startzeiten genutzt werden.
Der Anstieg der durchschnittlichen Gesamtkosten bei Erhöhung des Startzeitfaktors muß nicht automatisch mit höheren durchschnittlichen relativen Kostendifferenzen verbunden sein. Denn bei der Durchführung früherer Co-Reservierungsvarianten kann es vorkommen, daß für spätere Anfragen die günstigen Co-Reservierungsvarianten vorab gefiltert werden und diese nur Varianten mit höheren Gesamtkosten bilden. Ab der Simulation mit k=0.7 und weniger sind die Anstiege der durchschnittlichen Gesamtkosten darauf zurückzuführen, daß von den gebildeten Co-Reservierungsvarianten tatsächlich teurere durchgeführt wurden, um gleichzeitig frühere Startzeiten zu erreichen. Dies ist auch an den höheren Standardabweichungen der relativen Kostendifferenzen zu erkennen. Je größer der Startzeitfaktor im Verhältnis zum Kostenfaktor ist, desto teurere Co-Reservierungsvarianten durften durchgeführt werden.
Die geringen Veränderungen der durchschnittlichen relativen Kostendifferenzen sind darauf zurückzuführen, daß die Reservierungslast auf die sieben Cluster verteilt insgesamt sehr

niedrig war und für Co-Reservierungsanfragen nur wenige Kostengruppen bei Co-Reservierungsvarianten gebildet wurden. Die langsamen Veränderungen der Standardabweichungen für relativen Kosten- und Startzeitdifferenzen sind ebenfalls durch diesen Effekt erklärbar.

Schlußfolgerungen:
Kombinierte Optimierungen für Co-Reservierungsanfragen sind mit Algorithmus 5 möglich. Für Anfragen können, je nach Wahl der Optimierungsfaktoren, Co-Reservierungsvarianten mit dem besten Trade-Off aus frühester Startzeit und geringsten Kosten durchgeführt werden.

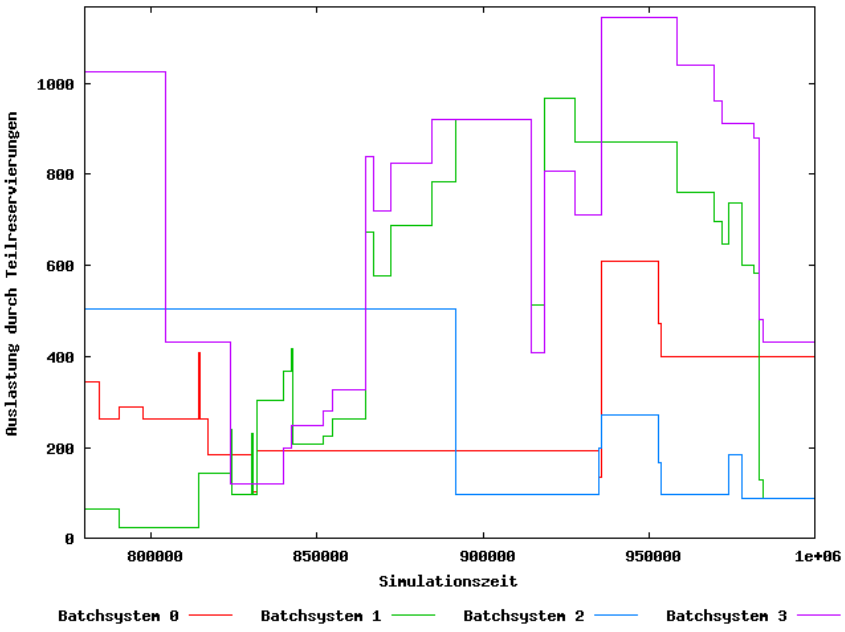
6.9.2 Untersuchung: Batchsystem-Auslastung und Expansion-Faktor

In dieser Untersuchung soll festgestellt werden, welche Auswirkungen heterogene Reservierungskosten auf die einzelnen Batchsysteme haben. Die durchschnittliche Auslastung und die durchschnittlichen Expansion-Faktoren der Jobs werden untersucht.

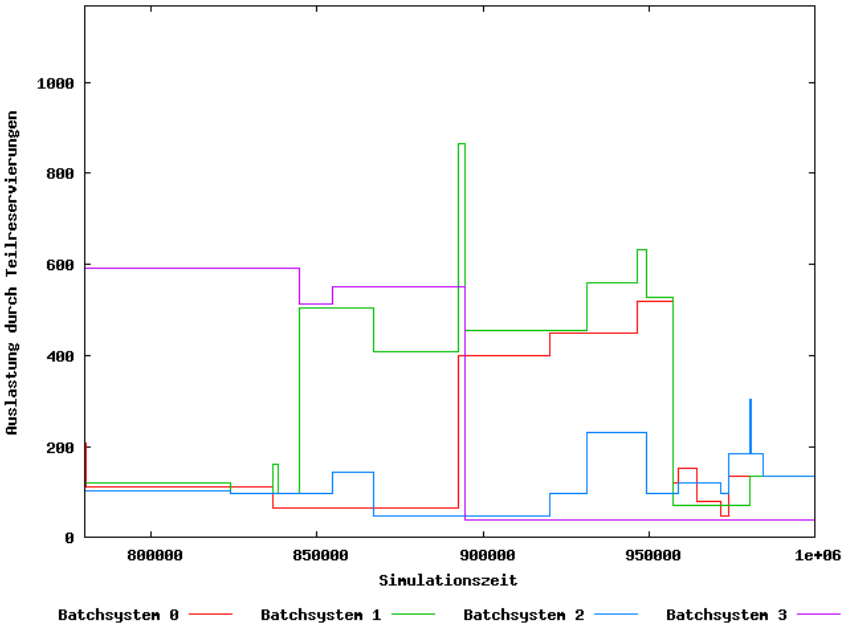
Erwartungen:
Bei Clustern mit niedrigen Kosten ist eine Häufung der Teilreservierungen zu erwarten, ebenso eine stärkere Verzögerung der Jobs, die höhere durchschnittliche Expansion-Faktoren verursachen.

Durchführung:
Eine Simulation wird mit homogenen Kosten durchgeführt (Backfill-Algorithmus), eine weitere mit Algorithmus 5 bei reiner Kostenminimierung. Der Anschaulichkeit wegen werden nur 4 Cluster-Kandidaten simuliert, ansonsten werden wieder die Parameter von Tabelle 18 verwendet.

Resultate:
Abbildung 29(a) zeigt die Batchsystem-Auslastungen zur ersten Wochenendzeit bei Kostenoptimierung. Dargestellt ist nur der Reservierungsanteil an den Auslastungen. Batchsystem 1 und 3, die niedrigere Kosten haben, verarbeiten überdurchschnittlich viel Reservierungslast. Batchsystem 0 und 2 verarbeiten erst zur Zeit 940000 Teilreservierungen, nachdem die günstigen Cluster dazu nicht mehr in der Lage sind (1160 CPUs sind maximal verfügbar).
In Abbildung 29(b) ist zu sehen, daß die günstigen Cluster zu dieser Zeit, wenn keine Kosten minimiert werden müssen, vergleichsweise wenig Reservierungslast verarbeiten. Im Zeitrahmen zwischen Beginn der Wochenendzeit (800000 sek) und Ende (975000 sek) starten im Szenario mit Kostenminimierung 16 Teilreservierungen, im Szenario ohne Optimierung sind es nur 7.



(a) Bei Kostenoptimierung mit Algorithmus 5



(b) Ohne Kostenoptimierung (Backfill-Algorithmus)

Abbildung 29: Verlauf der Batchsystem-Auslastungen (nur Reservierungsanteil)

Tabelle 20 zeigt die durchschnittlichen prozentualen Batchsystem-Auslastungen und die Anteile die Jobs und Teilreservierungen an diesen haben. Die durchschnittlichen Expansion-Faktoren und Auftragsbestandszeiten sind ebenfalls in Tabelle 20 dargestellt. In Tabelle 20(a) sind die Anzahlen der Teilreservierungen, die in den Batchsystemen liefen, noch einmal nach CPU-Bedarf aufgeschlüsselt.

Simulation	Kostenminimierung					keine Optimierung				
Batchsystem	0	1	2	3	Mittel	0	1	2	3	Mittel
`#cpus[16-32]`	16	11	16	1	11	10	8	11	5	8
`#cpus[33-64]`	19	19	15	15	17	21	10	23	12	16
`#cpus[65-128]`	23	19	23	13	19	21	13	19	11	16
`#cpus[129-256]`	19	18	12	7	14	15	12	16	11	13
`#cpus[257-512]`	6	7	6	6	6	6	5	7	7	6
`#cpus[513-1024]`	2	0	1	3	1	1	3	0	2	1
Summe	85	74	74	45	69	74	51	76	49	62

(a) Anzahl der Teilreservierungen pro Batchsystem

Simulation	Kostenminimierung					keine Optimierung				
Batchsystem	0	1	2	3	Mittel	0	1	2	3	Mittel
Job	64	63	71	71	67	65	63	70	76	68
Reservierungen	10	14	9	19	13	13	14	11	12	12
Job+Reservierungen	74	77	80	90	80	78	77	81	88	81
Expansion-Faktor	3.96	2.56	3.65	5.87	4.15	5.07	3.18	3.38	3.72	3.96
Auftragsbestandszeit	34462	70178	50297	114654	67397	44945	68355	52518	88980	63699

(b) Durchschnittliche Batchsystem-Auslastungen, Expansion-Faktoren und Auftragsbestandszeiten

Tabelle 20: Übersicht: Verteilung der Co-Reservierungen bei heterogenen Kosten

Erklärungen:
Die erhöhte Reservierungslast der günstigen Cluster ist auch an den Durchschnittswerten der Batchsystem-Auslastung in Tabelle 20 zu sehen. Im durch Jobs hoch ausgelasteten Batchsystem 3 ist in beiden Simulationen die durchschnittliche Auslastung etwa bei 90 Prozent. Allerdings wird diese bei dem kostenminimimierenden Algorithmus mehr durch Teilreservierungen verursacht als durch Jobs. Dies führt zu einer Erhöhung der Job-Verzögerung der verdrängten Jobs und erklärt den Anstieg der durchschnittlichen Expansion-Faktoren um 60 Prozent.
Ebenfalls wirkt sich die zusätzliche Last durch Teilreservierungen auf die Auftragsbestandszeit der günstigen Cluster aus. Diese werden von jeder Anfrage, die den günstigen Zeitraum nutzt, als Favorit betrachtet. Wie bereits in Abbildung 29(a) zu sehen war, werden bevorzugt Anfragen bei ihnen reserviert. Dadurch erhöht sich die Auftragsbestandszeit der günstigen Batchsysteme. Batchsystem 1 verarbeitet sehr viel mehr kleine und mittlere Teilreservierungen als im Szenario ohne Kostenminimierung. Wenn Reservierungsanfragen eine niedrige späteste Startzeit haben, dann müssen sie wegen des Auftragsbestandszeitkriteriums auch auf weniger günstige Cluster ausweichen. Dies ist eine Erklärung, warum in Batchsystem 3 weniger Teilreservierungen laufen, und Batchsystem 0 trotz der im Schnitt höheren Kosten mehr Teilreservierungen verarbeitet. Um in Batchsystem 3 Teilreservierungen unterbringen

zu können, sind oft Vorausbuchzeiten von über einem Tag nötig. Die Kopplung von Dauer und Vorausbuchzeit von Anfragen im Reservierungs-Workload ist eine Erklärung, warum in Batchsystem 3 eher längere, höhere Last verursachende Teilreservierungen laufen. Daher hat Batchsystem 3 trotz der geringeren Anzahl an Teilreservierungen bei Kostenminimierung mit durchschnittlich 20 Prozent den größten Anteil der Reservierungslast zu tragen.

Schlußfolgerungen:
Die Verwendung heterogener Kosten führt zu einer Umverteilung der Reservierungslast zu günstigen Clustern. Es sind eventuell zusätzliche Mechanismen nötig, um die Reservierungslast zu balancieren. Besonders bei großen Vorausbuchzeiten der Co-Reservierungsanfragen könnten lokale Batch-Jobs in günstigen Batchsystemen sehr lange verzögert werden.

7 Zukünftige Arbeiten

Es sind noch weitere Erweiterungsmöglichkeiten zu den vorgestellten Co-Reservierungsalgorithmen denkbar. Vier mögliche Erweiterungen werden in diesem Kapitel vorgestellt. In Abschnitt 7.1 wird ein Ansatz für eine allgemeine Reservierungslastbegrenzung bei den lokalen Reservierungsdiensten vorgestellt. In Abschnitt 7.2 wird ein Ansatz vorgestellt, mit dem die Job-Verteilung der Batchsysteme im Co-Reservierungsalgorithmus berücksichtigt werden kann. In Abschnitt 7.3 wird eine Erweiterung des Algorithmenschemas von Kapitel 4.1 vorgestellt, in der Neuaushandlungen für bereits reservierte Co-Reservierungsvarianten möglich sind. Die Idee für diesen Ansatz ist, daß zu späteren Zeiten möglicherweise bessere Co-Reservierungsvarianten zu einer Anfrage gefunden werden können, als zum ersten Anfragezeitpunkt. Ausfälle von lokalen Reservierungsdiensten wurden in den vorgestellten Algorithmen nicht betrachtet. In Abschnitt 7.4 wird eine Erweiterung vorgeschlagen, die diese berücksichtigt.

7.1 Algorithmus Reservierungslaststeuerung

Für die fünf vorgestellten Algorithmen gibt es keine Möglichkeit für einen Cluster Anzahl oder Last der Teilreservierungen zu begrenzen. Co-Reservierungsanfragen mit genügend langer Vorausbuchzeit sind immer durchführbar. Snell et al. stellten fest, daß die gemeinsame zeitliche und räumliche Nutzung der Ressourcen durch Vorausreservierungen und lokale Jobs nur bis zu einem bestimmten Anteil an Reservierungslast sinnvoll ist [4]. Bei weiterem Anstieg des Reservierungsanteils sinkt die durchschnittliche Batchsystem-Auslastung.
Im Umkehrschluß wurde festgestellt, daß, wenn eine zeitliche oder räumliche Trennung in der Ressourcenverwaltung benutzt wird, bei niedrigem Anteil der Vorausreservierungen an der durchschnittlichen Auslastung die Ressourcen verschwendet werden. Bei niedriger Job- oder Reservierungslast sind die Ressourcen nicht durch die jeweils andere Nutzungsmöglichkeit verwendbar. Ebenso hat, durch die Partitionierung der Ressourcen für einerseits Jobs und andererseits Reservierungen, der Scheduler weniger Optimierungsmöglichkeiten durch Backfilling.
Eine denkbare Erweiterung der Co-Reservierungsalgorithmen ist die Einführung einer Reservierungsquota bei lokalen Reservierungsdiensten. Mit einer Quota kann ein Mittelweg zwischen uneingeschränkter Reservierbarkeit des Clusters und der Sperrung der Ressourcen für Teilreservierungen beschrieben werden.
Bei den lokalen Reservierungsdiensten könnte eine zusätzliche Prüfung der Teilreservierungsvarianten vorgenommen werden:

- für jede Teilreservierungsvariante bestimme den erwarteten Reservierungsgrad für den Fall der Durchführung
- lehne die Variante ab, wenn der Reservierungsgrad die Quota überschreitet

Bei der Ermittelung des Reservierungsgrades kann ähnlich vorgegangen werden wie beim CPU-Zähl-Algorithmus (Abschnitt 4.3). Definieren könnte man ihn als Verhältnis der durchschnittlich reservierten CPU-Anzahl während der Reservierungsdauer zur gesamten CPU-Anzahl des Clusters.
Das erwartete Resultat der Berücksichtigung von Quotas ist, daß Teilreservierungen gegebenenfalls auf andere Cluster ausweichen müssen, bei denen die Durchführungen noch tole-

riert werden. Ein Administrator könnte nun die Quota experimentell so konfigurieren, daß die höchst mögliche durchschnittliche Auslastung durch lokale Jobs und Co-Reservierungen erreicht werden kann. Als Nebeneffekt könnte zu Wartungsarbeiten am Cluster der Co-Reservierungsdienst abgeschaltet werden, in dem die Quota für Teilreservierungen frühzeitig auf null herabgesetzt wird.
Eine noch weitergehende Variante zur Steuerung der Reservierungslast ist denkbar. Falls keine weiteren Sortiervorgaben für Anfragen gegeben werden, könnten Co-Reservierungsvarianten, anstatt unsortiert, nach aufsummierten Reservierungsgraden von Teilreservierungsvarianten im Variantenbaum sortiert werden. Bei Clustern, die noch keine oder kaum Reservierungslast tragen, könnten somit Reservierungen gezielter durchgeführt werden als bisher.

7.2 Algorithmus Job-Verteilung

Ein Ansatz die durchschnittliche Job-Verzögerung zu senken, ist die Job-Verteilung bei der Anfragebearbeitung zu berücksichtigen:
Die Ressourcen des Batchsystems können besser genutzt werden, wenn überwiegend kleine Jobs verarbeitet werden. Eine Schlußfolgerung daraus ist, daß Teilreservierungen bei gleicher Verfügbarkeit der Ressourcen zuerst bei Clustern durchgeführt werden sollten, die in der letzten Zeit überwiegend kleine Jobs verarbeitet haben. Auswahlbevorzugungen können mit dem in Algorithmus 5 vorgestellten Schema berücksichtigt werden. Das Sortierkriterium ist in diesem Fall ein charakteristisches Job-Verhältnis zwischen großen und kleinen Jobs. Dieses Verhältnis könnte bei Statusanfragen ähnlich wie die Teilkosten bei Algorithmus 5 zurückgegeben werden. Anschließend ließe sich im Variantenbaum ein durchschnittliches Job-Verhältnis für jede Co-Reservierungsvariante ermitteln und könnte als zusätzliches Sortierkriterium berücksichtigt werden.

7.3 Algorithmus Neuaushandlung

Rui Min und Muthucumaru Maheswaran untersuchten ebenfalls Co-Reservierungsalgorithmen für CPU-Ressourcen [27]. In ihren Algorithmen werden Prioritäten und Nutzenfunktionen für Co-Reservierungsanfragen unterstützt. Sie verwendeten einen Scheduler, der durchgeführte Co-Reservierungsanfragen nach ihrer Aushandlung weiter überwacht. Höher priorisierte Reservierungsanfragen können die Ressourcen einer niedrigeren wieder entziehen und eine erneute Aushandlung für diese erzwingen. Ebenso stellen sie einen Algorithmus vor, der nach jedem Scheduling-Ereignis die Anfragen nach ihrer Nutzenfunktion neu bewertet und gegebenenfalls umsortiert. Gegenüber den in dieser Arbeit vorgestellten Algorithmen hat diese Vorgehensweise den Vorteil, daß sich weiter optimierte Co-Reservierungsvarianten, als den anfänglich verfügbaren, durchführen lassen.
In Simulationen fanden sie heraus, daß bei dem Algorithmus, der bereits durchgeführte Anfragen umsortiert, häufigere Ablehnungen von neu gestellten Anfragen mit längerer Reservierungsdauer auftreten. Dieser Nachteil war so schwerwiegend, daß sich der Gesamtnutzen für alle Anfragen im System verschlechterte. Ebenso vermindern neuaushandelbare Anfragen die Planungssicherheit des Benutzers, der eine solche Anfrage stellt. Dieser weiß nie genau, zu welchem Zeitpunkt die Anfrage bewilligt wird, so lange die Startzeit der zuletzt neu ausgehandelten Variante nicht erreicht ist.
In einem Modell ohne globalen Scheduler, der Co-Reservierungsanfragen mit verdrängten

Teilreservierungen wieder in einen konsistenten Zustand überführt, könnte man neuaushandelbare Reservierungen folgendermaßen berücksichtigen. Im Co-Reservierungsalgorithmus könnten zwei Typen von Reservierungsanfragen unterstützt werden, neu verhandelbare und einmalig verhandelbare. Neu verhandelbare Anfragen dürfen keine nicht verhandelbaren Anfragen verdrängen. Ebenso könnten sie mit höheren ökonomischen Kosten verrechnet werden, da sie einmalig verhandelbare Reservierungen behindern können oder weil diese zusätzliche Eigenschaft aus ökonomischen Gründen bezahlt werden sollte.
Wenn ein Benutzer nun eine einmalig verhandelbare Anfrage stellt, wird weiterhin nach der allgemeinen Vorgehensweise von Kapitel 4.1 reserviert.
Eine einfache Methode neu aushandelbare Anfragen nach unserem Modell durchzuführen, könnte folgendermaßen umgesetzt werden:
Für neu verhandelbare Co-Reservierungen wird in periodischen Abständen durch einen die Anfrage überwachenden Agenten eine identische Anfrage durchgeführt. Anschließend wird die schlechtere beider reservierten Co-Reservierungsvarianten zurückgesetzt. Der Agent beendet die Testschritte, wenn die Startzeit der zuletzt gültigen Co-Reservierungsvariante erreicht ist.

7.4 Algorithmus Robustheit

Die vorgestellten Co-Reservierungsalgorithmen bieten Klienten Nutzungsgarantien für die angeforderten Ressourcen, sofern eine Anfrage einmal bewilligt ist und keines der beteiligten Systeme ausfällt. Ein weitergehender Ansatz ist nun auch Systemausfälle im Co-Reservierungsalgorithmus zu berücksichtigen. Dazu kann klientenseitig ein Agent installiert werden, der bis zum Start der Co-Reservierung die Verfügbarkeit der Ressourcen weiterhin prüft. Wenn eine reservierte Ressource über längere Zeit nicht erreicht werden kann, dann kann versucht werden die nicht verfügbare Teilreservierung zur selben Zeit an einem anderen Cluster nachzureservieren oder die gesamte Anfrage gegebenenfalls neu auszuhandeln.

8 Zusammenfassung

In dieser Arbeit wurden fünf Co-Reservierungsalgorithmen vorgestellt, die die Vorausreservierung von CPU-Ressourcen an mehreren Clustern für den gleichen Zeitraum erlauben. Sie bieten den Klienten eines Grid-Reservierungsdienstes die Möglichkeit High-Level-Anfragen zu stellen, in denen von der Ressourcenauswahl und der konkreten Reservierungsstartzeit abstrahiert wird. Sie wenden verschiedene Strategien an, um effizient Co-Reservierungen zu den Anfragen durchzuführen.
Mit dem ersten Algorithmus wurde ein einfacher Algorithmus vorgestellt, der aus der Co-Reservierungsanfrage einen Suchraum von Teilreservierungsvarianten erzeugt, in der die flexiblen Attribute variiert werden. Dann kombiniert er passende Teilreservierungsvarianten miteinander zu Co-Reservierungsvarianten, von denen anschließend eine zufällige davon zu reservieren versucht wird. Dieses Verfahren erweist sich als einfach und effizient, wenn die Vorausbuchzeiten von Reservierungsanfragen groß genug sind.
Der zweite Algorithmus führt Statusanfragen bei lokalen Reservierungsdiensten der Ressourcen durch. Vor dem Reservierungsschritt wird dort gezählt, ob genügend CPUs frei sind. Je mehr Varianten für Teilreservierungen ein Cluster-Kandidat bietet, desto mehr lohnt sich die CPU-Zählung. Sehr wahrscheinlich fehlschlagende Reservierungsversuche werden somit bereits im Vorfeld erkannt und ausgefiltert. Wenn keine Ressourcen verfügbar sind, dann verläuft der Reservierungsprozesses effizienter als beim ersten Algorithmus.
Mit dem dritten Algorithmus wird die Unfairness verringert, die Co-Reservierungen für wartende Jobs in den Batchsystemen bewirken können. Vor dem Reservierungsschritt werden Statusanfragen bei Cluster-Kandidaten durchgeführt. Aus den geplanten Batch-Jobs und Reservierungen wird für jeden eine Auftragsbestandszeit ermittelt, die eine Untergrenze der erlaubten Startzeiten für Teilreservierungskandidaten darstellt. Teilreservierungsvarianten mit zu frühen Startzeiten, die dem Fairness-Prinzip der Job-Verarbeitung widersprechen, werden verworfen. Das Verfahren hat Nebeneffekte auf die Verteilung der Reservierungslast und die Erfolgsrate von Co-Reservierungsanfragen. Cluster mit niedrigerer durchschnittlicher Auslastung können dadurch stärker mit Teilreservierungen belastet werden. Insbesondere können Jobs von Clustern, deren Jobs im Schnitt eher höhere CPU-Anforderungen haben durch diesen Co-Reservierungs-Algorithmus stärker verzögert werden. Mit dem dritten Algorithmus lassen sich fast keine unmittelbar startenden Co-Reservierungen anfordern, auch wenn die Batchsysteme über ungenutzte freie Ressourcen verfügen. Aus diesem Grund wurde der dritte Algorithmus durch die Fähigkeit „Backfill-Reservierungen“ vornehmen zu können erweitert. Durch das Backfilling können die zur Anfragezeit freien Ressourcen genutzt werden, die durch wartende Jobs nicht verwendet werden. Die Fairness zu den wartenden Jobs bleibt dadurch erhalten und es können mehr Co-Reservierungsanfragen erfolgreich durchgeführt werden. Besonders kürzere Co-Reservierungen, die wenig Ressourcen benötigen aber zu frühen Startzeiten angefordert werden, schlagen seltener fehl.
Der vierte vorgestellte Algorithmus beachtet die Benutzervorgabe, die früheste mögliche Co-Reservierungsvariante für eine Anfrage zu reservieren. Dies wird durch eine einmalige Sortierung nach der Startzeit nach der Ermittelung der Co-Reservierungsvarianten bewerkstelligt. Der Algorithmus beachtet nur die Startzeiten, die ihm nach der Fairness-Prüfung der Teilreservierungsvarianten verbleiben. Wegen der in Algorithmus 2 vorgestellten Verfügbarkeitsprüfung kann er mit hoher Wahrscheinlichkeit die frühesten gebildeten Startzeiten re-

servieren, in einigen Fällen auch die Startzeiten, die ihm durch die Backfill-Optimierung ermöglicht wurde. Auf diese Weise können optimierte Co-Reservierungsvarianten durchgeführt werden, ohne Abstriche bei Effizienz oder Fairness in Kauf nehmen zu müssen.
Mit dem letzten Co-Reservierungs-Algorithmus können mehrere Auswahlkriterien für Co-Reservierungsvarianten gleichzeitig berücksichtigt werden. Er erlaubt eine flexible Sortierung der Co-Reservierungsvarianten nach einem konfigurierbaren Trade-Off für jedes Sortierattribut. Exemplarisch wurde in dieser Arbeit ein Kosten und Startzeit optimierender Algorithmus entworfen. Je nach Konfiguration der Gewichtungen für die Sortierattribute können frühere oder billigere Co-Reservierungsvarianten mit dem Algorithmus reserviert werden. Anhand der Gewichtungsfaktoren für jedes Sortierattribut verrechnet der Algorithmus für alle Co-Reservierungsvarianten gewichtete Teilwerte zu kalkulierten Gesamtwerten. Bei Clustern, die nach dieser Trade-Off-Berechnung günstig sind, werden bevorzugt Reservierungen durchgeführt. Der Algorithmus beachtet ebenfalls die Fairness zu wartenden Jobs. Wenn Cluster jedoch dauerhaft günstig sind und Anfragen mit großen Vorausbuchzeiten gestellt werden, können zukünftige Jobs stark verzögert werden. Die Algorithmen, besonders der zu letzt vorgestellte, sollten noch durch eine Lastbegrenzungskomponente erweitert werden, bevor man sie praktisch einsetzen kann.
Die Co-Reservierungsalgorithmen wurden praxisnah in einer Simulationsumgebung mit dem Maui-Scheduler getestet. Bei der Entwicklung einer Implementierung eines Co-Reservierungsdienstes kann auf Teile der Simulationsumgebung zurückgegriffen werden.

```
1  -- Signatur der Funktion
2  -- Typ formaler Parameter 1 -> ..-> Typ formaler Parameter n -> Typ Ausgabe
3  plus :: Int -> Int -> Int
4  -- Funktionsdefinition
5  plus a b = a+b
6  --Beispielausdruck
7  plus 1 2
8  -- Ausgabe davon
9  3
10 -- Funktionsdefinition mit Fallunterscheidung durch Guardzeichen "|"
11 fact n                          -- = fehlt -> Fallunterscheidung
12   | n == 0 = 1                  -- als erstes geprüft
13   | n == 1 = 1                  -- als zweites geprüft
14   | otherwise = n * fact (n - 1) -- immer wahr
15
16 -- Funktionsdefinition mit Hilfsdefinitionen
17 sub a b = differenz
18   where                -- Einleitung des Hilfsdefinitionsblocks
19     differenz = a - b -- Menge von Definitionen
```

Listing 8: Definition von Funktionen in Haskell

A Haskell

In diesem Appendix wird eine kurze Einführung in die funktionale Programmiersprache Haskell gegeben [28].

A.1 Funktionsdefinition

Haskellprogramme bestehen aus Funktionsdefinitionen, die in Form von Gleichungen angegeben werden. Im linken Term der Gleichung werden der Name und die formalen Parameter angegeben, im rechten die Abbildung. Für jede Funktion kann eine Signatur definiert werden. Die Typen der formalen Parameter werden in ihr angegeben und als letztes Element zusätzlich der Typ der Funktionsausgabe. Beispiele für Funktionsdefinitionen sind in Listing 8 gegeben.

A.2 Standardfunktionen und Datentypen

Haskells wichtigste Datenstrukturen sind Tupel und Listen. Listen sind in Haskell streng getypt und werden mit eckigen Klammern beschrieben. Tupel verhalten sich wie Listen mit fester Länge, sie werden mit runden Klammern beschrieben. Tupel-Elemente können unterschiedliche Typen haben. Arrays gibt es in Haskell nicht, diese können aber mit Listen dargestellt werden. Listing 9 zeigt die wichtigsten Operationen auf Listen in Haskell.

```
1  -- Gleichheit - vergleicht alle Werte der Liste
2  [1,0] == [1,0]
3  True
4  -- Ungleichheit - vergleicht alle Werte der Liste
5  [1,0] /= [1,0]
6  False
7  -- leere Liste
8  []
9  -- Anfügen von Elementen an Listen
10 1:[2,3]
11 [1,2,3]
12 -- Vereinigung von Listen
13 [1,2] ++ [3]
14 -- Array-ähnlicher Elementzugriff auf eine Liste
15 [1,2,3] !! 0
16 1
17 -- Filtern von Elementen aus Listen
18 filter (<3) [1,2,3]
19 [1,2]
20 -- Transformation von Listen
21 -- Anwenden der Funktion (*2) auf jedes Element der Liste
22 map (*2) [1,2,3]
23 [2,4,6]
24 -- Listenerzeugung
25 [1 .. 3]
26 [1, 2, 3]
27 -- Listenerzeugung mit Schritten
28 [1,3 .. 10]
29 [1,3,5,7,9]
30 -- Listen können in Haskell ähnlich wie Mengen konstruiert werden.
31 -- Listengenerator Syntax:
32 -- Allgemein [Listenelement | Generator 1, ..., Generator n, Filter 1, ..., Filter m]
33 -- Beschreibung: Für jede Kombination der Generatorlistenelemente, füge,
34 -- wenn alle Filterbedinungen gelten, diese der Liste hinzu.
35 -- Bei mehreren Generatoren werden die rechts stehenden mehrfach durchlaufen.
36 -- Dies ist hilfreich, um Suchräume effizient zu beschreiben.
37 -- Beispiel 1:
38 [(a,b) | a <- [1 .. 3], b <- [1 .. 3], a <= b]
39 [(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]
40 -- Beispiel 2:
41 ungeradeziffern = [ziffer | ziffer <- [0 .. 9], odd ziffer]
42 [1,3,5,7,9]
```

Listing 9: Beispiele für Haskell Operationen

Literatur

[1] A. Afzal. *Integration of Advanced Reservation Support and SGE (Sun Grid Engine) into ICENI*. PhD thesis, Imperial College e-Science Networked Infrastructure, 2003.

[2] C. Kesselman K. Czajkowski, I. Foster. Resource co-allocation in computational grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, CA, USA*, page 37, August 1999.

[3] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the 7th International Workshop on Quality of Service, London, UK*, pages 27–36, 1999.

[4] Q. Snell, M. J. Clement, D. B. Jackson, C. Gregory. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing, Cancun, Mexico*, pages 137–153. Springer-Verlag, 2000.

[5] F. Schintke A. Reinefeld. Concepts and technologies for a worldwide grid infrastructure. In *Proceedings of the 8th International Euro-Par Conference on Parallel Processing, Paderborn, Germany*, pages 62–72. Springer, August 2002.

[6] C. Kesselman I. Foster. Globus: A metacomputing infrastructure toolkit. In *International Journal of Supercomputer Applications and High Performance Computing*, volume 11, pages 115–128. Sage Science Press, 1997.

[7] F. Schintke J. Wendler. Grid-Enabled Computational Fluid Dynamics using FlowGrid. In *Proceedings of the Cracow Grid Workshop '03*, pages 59–66. Academic Computer Centre CYFRONET AGH, February 2004.

[8] R. L. Henderson. Job scheduling under the portable batch system. In *1st Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA*, volume 949 of *Lecture Notes in Computer Science*. Springer-Verlag London, UK, 1995.

[9] S. Zhou. Lsf: Load sharing in large-scale heterogenous distributed systems. In *Proceedings of the Workshop on Cluster Computing, Tallahassee, FL, USA*, 1992.

[10] Center for HPC Cluster Resource Management, Scheduling and Grid Scheduling Research and Development. The maui scheduling system, 2004. `http://supercluster.org/projects/maui`.

[11] C. Lee, Y. Schwartzman, J. Hardy, A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th Job Scheduling Strategies for Parallel Processing, New York*. Springer-Verlag, June 2004.

[12] Chiang Su-Hui, Andrea Arpaci-Dusseau, Mary K. Vernon. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. In *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag London, UK, July 2002.

[13] V. Taylor W. Smith, I. Foster. Predicting application run times using historical information. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, FL, USA*, volume 1459 of *Lecture Notes in Computer Science*, pages 122–142. Springer-Verlag, March 1998.

[14] D. S. Johnson M. R. Garey. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1990.

[15] J. Giddy R. Buyya, D. Abramson. Economy driven resource management architecture for computational power grids. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00), Las Vegas, Nevada, USA*, 2000.

[16] M. Murshed R. Buyya. A deadline and budget constrained cost-time optimize algorithm for scheduling parameter sweep applications on the grid. In *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, Nevada, USA*, June 2002.

[17] J. Wendler T. Röblitz, F. Schintke. Elastic Grid Reservations with User-Defined Optimization Policies. In *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM'04), Antibes Juan-les-Pins, France*, September 2004.

[18] L. Rudolph D. Feitelson. Metrics and benchmarking for parallel job scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, FL, USA*, volume 1459 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag London, UK, 1998.

[19] R. Meyer R. Bagrodia. Parsec: A parallel simulation environment for complex system. In *Computer Magazine*, volume 31, issue 10, pages 77–85. IEEE Computer Society Press Los Alamitos, CA, USA, October 1998.

[20] M. Murshed R. Buyya. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. In *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*. Wiley Press, May 2002.

[21] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, May 2001.

[22] E. C. Russell. Simscript ii.5 tutorial. In *Proceedings of the 17th conference on Winter simulation San Francisco CA, USA*, pages 57–58. ACM Press New York, NY, USA, 1985.

[23] R. McNab F. Howell. Simjava: A discrete event simulation library for java. In *Proceedings of the International Conference on Web-Based Modeling and Simulation*. Society for Computer Simulation International (SCS), January 1998.

[24] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, A. A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Washington, DC, USA, November 2000.

[25] M. Clement D. Jackson, Q. Snell. Core algorithms of the maui scheduler. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, Cambridge, MA*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag London, UK, 2001.

[26] D. Feitelson. Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures, Rome,*

Italy, volume 2459 of *Lecture Notes in Computer Science*, pages 114–141. Springer-Verlag London, UK, 2002.

[27] M. Maheswaran R. Min. Scheduling co-reservations with priorities in grid computing systems. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID.02) Berlin, Germany*. IEEE Computer Society, 2002.

[28] Peter Pepper. *Funktionale Programmierung (in Opal, ML und Haskell)*. TU Berlin, Skript für die Lehrveranstaltung Informatik-1, 1997.