technische universität
dortmund

Simulation von Online Scheduling-Algorithmen für parallele Systeme

Bachelorarbeit

Hendrik Rassmann
September 2020

Gutachter:
Prof. Dr. Peter Bucholz
Dr. Falko Bause

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

INHALTSVERZEICHNIS

1 EINLEITUNG 3

2 METHODIK 5

2.1 Problemstellung 5

2.1.1 Auftrag (Job) 5

2.1.2 Maschinenumfeld (Machine Environment) . . . 6

2.1.3 Scheduler 7

2.2 Zielfunktionen 7

2.3 Scheduling Algorithmen 8

2.3.1 Referenziell transparente Scheduler 8

2.3.2 Referenziell intransparente Scheduler 8

2.4 Simulation 9

2.4.1 Discrete Event Simulation 9

2.4.2 Statistisches Auswerten 10

3 EVALUATION 11

3.1 Reproduktion 11

3.1.1 Simulationsaufbau 11

3.1.2 Vergleich von Läufen 12

3.2 Weiterführende Untersuchungen 20

3.2.1 Backfilling und Fit als Funktionen höherer Ordnung 25

3.2.2 Property Based Testing zum Erkenntnisgewinn 29

3.2.3 Vergleich von Simulation und PBT 35

4 VERWANDTE ARBEITEN 41

5 FAZIT 43

1 EINLEITUNG

Die effiziente Zuteilung von Rechenaufgaben auf unterschiedliche Ressourcen ist nicht erst seit dem Zeitalter der Cloud wichtig. Dies war schon ein kritischer Bereich von Operations Research , als "Computer" noch ein Beruf war. In der Ära der Mainframe Computer wurden simple Wartelisten, die von OperatorInnen betreut wurden, nach und nach durch ausgeklügelte Systeme ersetzt. Heute übernimmt diese Aufgabe auf fast jedem Computer ein Betriebssystem. Dieses stellt sicher, dass alle Prozesse mit Rechenzeit versorgt werden. Gleichzeitig wird dem Benutzer eine niedrige Antwortzeit seiner Eingaben geboten wird. Sobald mehrere Rechner in einem Cluster zusammen an Aufgaben arbeiten, verschieben sich die Prioritäten. Ein Rechencluster kann nicht einfach wie ein Computer mit vielen Prozessorkernen behandelt werden. Zum einen ist eine vorzeitige Unterbrechung einer Berechnung entweder sehr teuer oder technisch kompliziert, fordert die Planung und Programmierung von Checkpoints [LPC+15], oder ist gar unmöglich [Ada79]. Zum anderen muss die Antwortzeit des Systems bei neuen Eingaben nicht im Millisekundenbereich liegen. Außerdem kann sich die Rechenleistung der einzelnen Knoten eines Clusters um einen beliebig großen Faktor unterscheiden. Des Weiteren können Aufträge strikte Beschränkungen haben, welche und wie viele Knoten sie benötigen, um ausgeführt werden zu können. Aus Zeit und (Oportunitäts) Kosten sind Experimente an echten Rechenclustern höchstens zur Validierung von Forschungsergebnissen durchführbar sind. Aus diesen Gründen werden solche Systeme in der Regel simuliert.

In dieser Arbeit werden zunächst relevante Grundlagen geschildert 2. Danach wird ein solches Simulationsmodell nachgebaut und erweitert. Als Basis dafür dient *A comparative study of online scheduling algorithms for networks of workstations* von Olaf Arndt et al. [AFKT00]. In jenem Paper wird ein Modell vorgestellt, simuliert, ausgewertet und experimentell validiert. Das vorgestellte Modell soll hier nachgebildet werden 3.1. Zur Überprüfung der Nachbildung werden Simulationsläufe analysiert, und die Ergebnisse mit den Ergebnissen in [AFKT00] verglichen 3.1.2.

Darüber hinaus wird das Modell erweitert, indem Kombinationen von Scheduling Algorithmen durch die Sichtweise der *Funktionalen Programmierung* betrachtet werden 3.2.1. Zusätzlich sollen verschieden Algorithmen zur Zuteilung von Rechenknoten und Aufträgen[1]

1 Im Folgenden werden diese auch als Scheduling Algorithmen oder Scheduler(s) bezeichnet.

statistisch verglichen werden 3.2.1. Durch Modellchecking Verfahren werden diese intuitiv anschaubar gemacht 3.2.2. Dadurch ergibt sich eine neue Möglichkeit, Scheduler zu vergleichen, ohne die Performance in einer konkreten Situation zu messen. Anschließend wird überprüft, wie gut die so gewonnenen Intuitionen das tatsächliche Verhalten vorhersagen 3.2.3.

2 METHODIK

Da der angehängte Programmcode die üblichen englischen Bezeichnungen verwendet, werden in dieser Arbeit auf Deutsch vorgestellte Konzepte zusätzlich zu üblicher mathematischer Notation auch immer einmal auf Englisch benannt.

2.1 PROBLEMSTELLUNG

Ein Rechencluster, bestehend aus einer Vielzahl von Knoten-Rechnern mit potenziell verschiedenen Rechenleistungen soll eine Menge von Aufträgen an Berechnungen durchführen. Die unterschiedlichen Aufträge sind dabei nicht im Voraus bekannt (OnLine Scheduling), und könnten gestartet werden, sobald sie angemeldet sind. Die Reihenfolge der Bearbeitung ist dabei nicht von Bedeutung, Aufträge hängen nicht voneinander ab. Die Aufträge sollen dabei durch eine geeignete Methode auf die Knoten aufgeteilt werden. Diese Aufteilung wird Schedule genannt. In diesem Kapitel werden Eigenschaften von Aufträgen und Knoten, sowie Kriterien zur Bemessung der Eignung der verschiedenen Methoden vorgestellt.
Es wird die Notation aus dem Kapitel *Scheduling Algorithms In: Handbook of Algorithms and Theory of Computation* [KSCW97] verwendet.

2.1.1 *Auftrag (Job)*

Ein Auftrag (im Englischen "job") ist ein Prozess, der Arbeitszeit einer oder mehrerer Maschinen benötigt, um abgeschlossen zu werden. Im Folgenden werden verschieden Charakterisierungen eines Auftrags p vorgestellt.

FESTSTEHENDE EIGENSCHAFTEN

ID: Eine eindeutige Identifikationsnummer. Diese kodiert keine weiteren Informationen.

Bearbeitungszeit (processing time) p_j: Dauer, die der Auftrag auf einem Knoten mit genormter Arbeitsgeschwindigkeit benötigt, um abgeschlossen zu werden. Diese Arbeit behandelt nicht, wie diese zuverlässig ermittelt werden können.

Einreihung (queueing time) q_j: Der Zeitpunkt, zu dem ein Auftrag j bekannt wird.

Parallelität (degree of parallelism) π_j: Die Anzahl an zugewiesenen Maschinen, die ein Auftrag benötigt, um gestartet zu werden. Hierbei erfolgt eine Aufteilung der Bearbeitungszeit auf

die zugewiesenen Knoten. Ein Auftrag von der Parallelität 1 wird als sequenziell bezeichnet, ansonsten wird von einem parallelen Auftrag gesprochen.

DURCH SIMULATION BESTIMMTE EIGENSCHAFTEN

Bearbeitungsbeginn (start time) s_j: Der Zeitpunkt, zu dem ein Auftrag j begonnen wird.
Abschluss (completion time) c_j: Der Zeitpunkt, zu dem ein Auftrag j abgeschlossen wird.
Zugewiesene Knoten (scheduled on nodes) n_j: Die Menge an Knoten, auf denen ein Auftrag j ausgeführt wird.

WEITERE EIGENSCHAFTEN

Darüber hinaus können Aufträge mit zusätzlichen Einschränkungen versehen werden, wie mit einen **Frühesten Bearbeitungsbeginn** (release date) r_j, oder mit einem **Spätesten Abschluss** (due date) d_j. Diese werden im Rahmen einer Simulation von voneinander unabhängigen Berechnungen nicht weiter untersucht.

2.1.2 *Maschinenumfeld (Machine Environment)*

Das Machine Environment, in dem die Aufträge ausgeführt werden, besteht aus einer Anzahl an Knoten, einer Warteliste bereits bekannter, wartender Aufträge (Q) und einem Scheduler. Die Knoten werden charakterisiert durch:

FESTSTEHENDE EIGENSCHAFTEN

ID: Eine eindeutige Identifikationsnummer. Diese kodiert keine weiteren Informationen.
Geschwindigkeit (speed) sp_k: Der Geschwindigkeit, mit der ein laufender Auftrag bearbeitet wird. Ein Auftrag j mit $p_j = 10$ und $\pi_j = 1$ benötigt 5 Zeiteinheiten auf einem Knoten k mit $sp_k = 2$. Ein Auftrag j' mit $p_{j'} = 10$ und $\pi_{j'} = 2$ benötigt 5 Zeiteinheiten auf zwei Knoten mit $sp_{k_1,k_2} = 1$.

VARIABLE EIGENSCHAFTEN

Führt aus (running) R_k: Die Menge an Aufträgen, die auf diesem Knoten ausgeführt werden. In dieser Arbeit wird wie bei Arndt et al. [AFKToo] nur der Spezialfall betrachtet, in dem Knoten nur einen Auftrag gleichzeitig ausführen.

2.1.3 *Scheduler*

Eine Scheduling Funktion ist eine Funktion, die eine Menge an wartenden Aufträgen auf bis zu ein (kleinstes) Element dieser Menge abbildet. Die Aufträge müssen eine totale Quasiordnung (auch Präferenzordnung) formen, d.h. alle Elemente sind mit einander vergleichbar. Diese Abbildung ist dabei nicht notwendigerweise deterministisch. Wenn zum Beispiel der Auftrag mit der geringsten Bearbeitungszeit gewählt werden soll, kann es mehrere geeignete Kandidaten geben ("breaking ties arbitrarily")[KSCW97]. Zugunsten der Reproduzierbarkeit von Läufen wird bei Gleichstand der Auftrag mit der niedrigeren ID gewählt. Aus diesem Grund muss sichergestellt werden, dass diese zufällig verteilt werden, und nicht etwa alle sequenziellen Aufträge niedrige IDs zugewiesen bekommen.
Kern der Problemstellung ist das Finden von Schedulern, die ein effektives Betreiben des Systems erlauben und einfach berechenbar sind.

2.2 ZIELFUNKTIONEN

Bevor verschiedene Scheduler miteinander verglichen werden können, muss zuerst festgelegt werden, mit welchem Maß gemessen werden soll. Die Ziele eines Schedulers sind, Fairness zwischen den Aufträgen zu garantieren, einen schnellen Start von Aufträgen zur ermöglichen und gleichzeitig eine hohe Auslastung der Knoten zu erreichen. Auch sollte, falls Attribute der Aufträge von Nutzern angegeben werden, durch gezielte falsche Angaben kein eigener Vorteil erzielbar sein. Diese Ziele sind oft unvereinbar. Beispielsweise ist es für eine geringe durchschnittliche Wartezeit nützlich, kurzen Aufträgen Vorlass zu gewähren. Dies könnte Nutzer dazu verleiten, unrealistisch kurze Laufzeiten anzugeben.

MAXIMUM WAITING TIME Die Wartezeit eines Auftrags j wird definiert als die Zeitspanne zwischen Einreihung q_j und Abschluss c_j. Die größte Wartezeit verhält sich umgekehrt zur Fairness eines Schedulers.

MAKESPAN Die Bearbeitungsspanne ist die vergangene Zeit zwischen dem frühsten Bearbeitungsbeginn und dem spätesten Abschluss.

AVERAGE WAITING TIME Die durchschnittliche Wartezeit ist die summierte Wartezeit aller Aufträge.

2.3 SCHEDULING ALGORITHMEN

Bei allen im folgenden vorgestellten Algorithmen wird der beste Auftrag ausgewählt, wobei sich das Kriterium dafür, welcher als bester bewertet wird, unterscheidet. Erfüllen mehrere Aufträge das Kriterium gleich gut, wird davon einer arbiträr ausgewählt. Die Algorithmen lassen sich dabei in die zwei Kategorien Referenziell Transparent und Intransparent einteilen. Details dazu in 3.2.1.

2.3.1 *Referenziell transparente Scheduler*

Die Wahl des nächsten Auftrags eines referenziell transparenten Schedulers hängt einzig von den Aufträgen in der Warteschlange ab, nicht vom sonstigen (eventuell unbekannten) Zustand des Systems.

FIFO **First in First out** (im Folgenden FiFo) wählt zu jedem Zeitpunkt denjenigen Auftrag, der sich bereits am längsten in der Warteschlange befindet. Stehen nicht genug Knoten zur Verfügung, um diesen Auftrag zu starten, müssen alle anderen Aufträge warten. Dieses Verfahren ist zwar fair, führt allerdings zu viel ungenutzter Rechenzeit.

SPT **Shortest Processing Time first** wählt den Auftrag mit der die kürzesten angegebene Bearbeitungszeit. Dadurch wird versucht, die average waiting time gering zu halten. Darunter leidet die Fairness, und die maximum waiting time eines nach hinten verdrängten langen Auftrags steigt.

GPT **Greatest Processing Time first** wählt genau invers zu SPT den Auftrag mit der längsten angegebenen Bearbeitungszeit aus. Dadurch wird die Bearbeitungsspanne (makespan) zu Gunsten des der durchschnittlichen Systemzeit (average flow time) minimiert.

2.3.2 *Referenziell intransparente Scheduler*

Diese Auswahl dieser Scheduler hängt zusätzlich von äußeren Faktoren ab, wie etwa dem Zustand der Knoten, oder vom Zustand anderer parallel laufender Aufträge. In einem realen verteilten System kann dies zu Problemen führen, wenn der globale Zustand nicht oder nur schwer ermittelt werden kann.

FIRST FIT **First Fit** ist eine Abwandelung von FiFo. Hier wird der Auftrag, der bereits am längsten wartet und sofort gestartet werden kann, ausgewählt. Dies verringert ungenutzte Rechenzeit, allerdings werden Aufträge mit einer hohen Parallelität nach hinten verdrängt.

BACKFILLING **Backfilling** ist eine weitere Abwandlung von FiFo. Zuerst wird der Auftrag bestimmt, der bereits am längsten wartet. Sollte dieser nicht gestartet werden können, dürfen andere Aufträge gestartet werden, vorausgesetzt, sie werden früher abgeschlossen, als der zuerst gewählte Auftrag beginnen wird. So wird ungenutzte Rechenzeit vermieden, ohne dass lange große Aufträge im Vergleich zu FiFo benachteiligt werden.

RANDOM **Random** wählt immer einen zufälligen Auftrag aus. Kann dieser nicht gestartet werden, so wird kein Auftrag gestartet.

2.4 SIMULATION

2.4.1 *Discrete Event Simulation*

Soll ein System zu diskreten Zeitpunkten simuliert werden, stehen die drei in *Introduction to discrete-event simulation and the simpy language* beschriebenen Paradigmen zur Auswahl [Mato8].

AKTIVITÄTSORIENTIERT Das System wird immer von einem Zustand S zu einem bestimmten Zeitpunkt t_n zu einem Zustand S' in t_{n+1} transformiert. Es wird also vom Startzustand S_0 aus einmal für jeden Zeitpunkt ein Zustand produziert.
Dieses Paradigma liefert eine einfache Implementierung, jedoch auch eine höhere Laufzeit als die beiden Alternativen.

EREIGNISORIENTIERT Hier wird zu von einem Zustand S_n ausgehend der nächste Zustand gefunden, der sich von S_n abgesehen von der aktuellen Zeit t unterscheidet. Es werden also, wenn bekannt ist, dass die nächste Aktion zum Zeitpunkt t_{n+k} stattfindet, alle diskreten Zeiteinheiten zwischen n und $n+k$ übersprungen.
Dieses Paradigma kann Rechenzeit einsparen, wenn der zusätzliche Overhead und die Frequenz der Aktionen klein genug ist.

PROZESSORIENTIERT Dieses Paradigma funktioniert ähnlich wie Ereignisorientiert, allerdings werden alle Aktivitäten durch Prozesse abgebildet. Dabei ähnelt jede Aktivität einem Prozess wie in einem Betriebssystem. Ein Prozess unterbricht seine eigene Ausführung, entweder für eine bestimmte Zeit, oder bis eine Bedingung erfüllt ist.
Dies modularisiert das Simulationsprogramm. Allerdings wird auch Kontrolle abgegeben. Soll zum Beispiel festgelegt werden, welche von zwei Aktionen zuerst ausgeführt werden soll, wenn diese gleichzeitig auftreten, ist dies Aktivitäts- oder Ereignisorientiert simpel. Prozess Orientiert erfordert diese Situation allerdings Synchronisation, zum Beispiel über einen Beobachter, der weiß, welche Prozesse auf welche Bedingungen warten.

2.4.2 *Statistisches Auswerten*

Um für die Auswertung stabile Werte zu erzielen, müssen die Ergebnisse mehrerer Simulationsläufe verwendet werden. In der Regel werden dazu die verschiedenen Maßzahlen wie Bearbeitungsspanne, Systemzeit etc., arithmetisch von 50 bis 100 Läufen gemittelt. Bei diesen Zahlen handelt es sich um einen Erfahrungswert. Manche Experimente erfordern mehr oder weniger Läufe (vgl. *Reasoning under Uncertainty* [Jel]). Dabei wird jeder Auftragsmix einmal mit jedem betrachteten Scheduling Algorithmus abgearbeitet. Mitunter entstehen S-Kurven, oder andere Kurven, die stellenweise konstant sind. Für diese werden manchmal mehr Läufe, mit Parametern der ”interessanten” Stellen simuliert.

3 EVALUATION

3.1 REPRODUKTION

Der hier verwendete Simulationsaufbau erlaubt auf Wunsch eine deterministische sequentielle Ausführung. Dies erlaubt weiterführende Untersuchungen, die in Kapitel 3.2.2 ausgeführt werden.

3.1.1 *Simulationsaufbau*

Die Wahl des Paradigmas 2.4.1 fällt in dieser Arbeit auf das Ereignisorientiert Schema 2.4.1. Dies mag erstaunen. Zunächst scheinen die Vorteile der Prozessorientierten Ansicht zu überwiegen. Im Falle eines verteilten Systems, wirkt die Modellierung durch Prozesse wie eine offensichtlich gute Wahl. In Vorteile die sich ergeben, wenn man sich gegen diese Schema entscheidet, sind zweierlei.
Erstens, ist es dadurch leichter, nahe an einer Automaten-Spezifikation zu bleiben. Es bleibt die Sichtweise, das System als Zustand und einer Menge an möglichen Aktionen/Zustandsübergängen zu betrachten. Durch die Definition eines Standardverhaltens, nämlich zuerst alle fertigen Aufträge zu beenden, dann Aufträge zu starten, bis keine mehr startbar sind, und dann Menge an Zeiteinheiten vergehen zu lassen, erzielt man ein deterministisches Verhalten (solange die Scheduling Algorithmen deterministisch sind). Dieses reproduzierbare Verhalten ist gegenüber der Prozessorientierten Sicht ein Vorteil, da Fehler leichter gefunden und behoben werden können.
Zweitens leitet sich aus deterministischem Verhalten ein großer Vorteil für das Property Based Testing zum Erkenntnisgewinn in Kapitel 3.2.2 ab. Es werden "flaky Tests" verhindert, also Tests, die mit dem selben Systemzustand bei mehrfacher Ausführung sowohl fehlschlagen als auch gelingen. In diesen Fällen müssten sonst mehrere Tests durchgeführt werden.
Was das Shrinking in Kapitel 3.2.2 angeht, könnte hier ein Aktivitätsorientierter Ansatz Sinn ergeben, da immer kleinere Läufe erzeugt werden, sodass sich der Zusatzaufwand des Ereignisorientierte Paradigmas immer weniger lohnt. Jedoch ist dieser Overhead so gering, dass er im Vergleich zu den hohen Laufzeiteinbußen in der Ausführung von natürlichen Auftragslisten nicht ausschlaggebend ist.
Darüber hinaus lässt sich durch eine Alternative zum Standardverhaltens leicht untersuchen, wie sich das Verhalten des System ändert, wenn ein perfektes Betreiben des Clusters nicht möglich ist. Zum Beispiel kann überprüft werden, wie stark sich eine gröbere Zeitauf-

lösung (etwa einmal alle 60 Sekunden) auf das Simulationsergebnis auswirkt.

3.1.2 *Vergleich von Läufen*

Geschwindigkeit der Knoten

Um abschätzen zu können, ob die vorgestellte Simulation erfolgreich reproduziert werden konnte, werden zuerst die Auswertungen von rein sequentiellen Auftragszusammenstellungen verglichen. Obwohl die Parameter zur Erstellung der Aufträge angegeben wurden, sowie die Tatsache, dass alle Knoten mit der selben Geschwindigkeit operieren, fehlt die Angabe über diese Geschwindigkeit. Experimentell lässt sich ermitteln, dass dieser Wert etwa 100 beträgt.

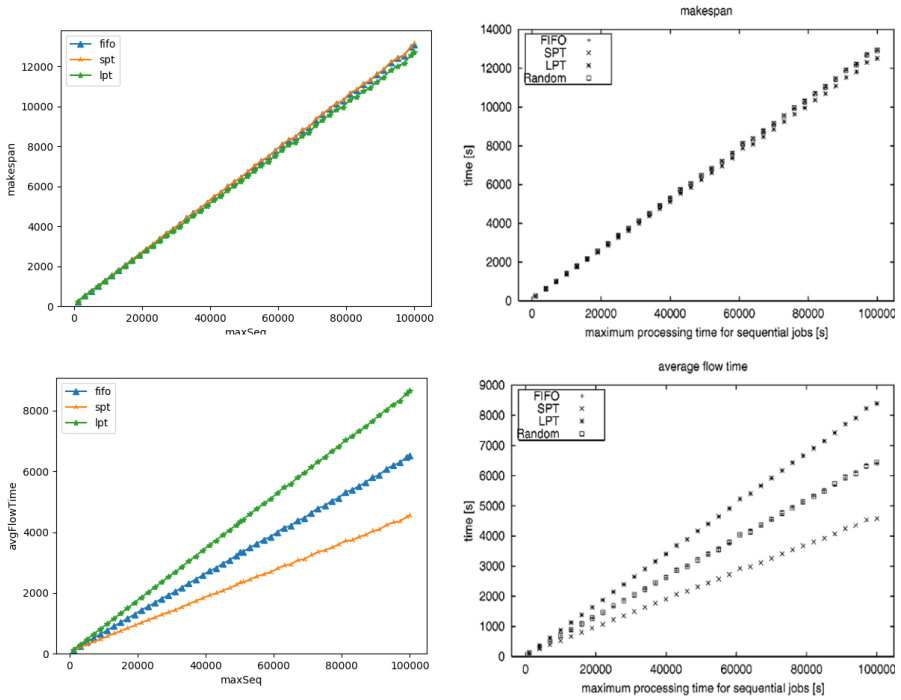
LPT, SPT

Bei Arndt et al. [AFKToo, p. 4] wird zwischen der Bearbeitungszeit ("processing time") und der Laufzeit ("runtime") eines Auftrags unterschieden. Dabei bezeichnet Laufzeit den "Quotienten aus der Bearbeitungszeit p_j und der kumulativen Geschwindigkeit der zugewiesenen Knoten". Das Ergebnis wird aufgerundet.
Bei der Simulation von SPT und LPT Läufen entstand folgendes Problem: Die Ergebnisse von Arndt et al. [AFKToo] ließen sich korrekt reproduzieren, solange alle Aufträge sequentiell waren. Sobald auch parallele Aufträge eingereiht wurden, entstanden geringe Abweichungen.
Es scheint, als würde der SPT Algorithmus in [AFKToo] nicht den Auftrag j mit der geringsten Bearbeitungszeit p_j auswählen, sondern denjenigen, der die geringste Zeit in Anspruch nimmt. Zum Beispiel würde j_1 mit $p_{j_1} = 10, \pi_{j_1} = 5$ gegenüber j_2 mit $p_{j_2} = 4, \pi_{j_1} = 1$ vorgezogen werden, da j_1 für $10/5 = 2$ Zeiteinheiten und j_2 für $4/1 = 4$ Zeiteinheiten ausgeführt wird.
Dies erfordert eine einfache Anpassung bzw. Interpretation des SPT Algorithmus. Es wird nach genannter Bearbeitungszeit geteilt durch den jeweiligen Grad an Parallelität sortiert. Diese Anpassung ergibt allerdings nur Sinn, solange alle Knoten mit der selben Geschwindigkeit arbeiten. Sobald Knoten unterschiedliche Geschwindigkeitseigenschaften besitzen, muss man die Bearbeitungszeit j_p durch die Summe der Geschwindigkeiten der schnellsten freien Knoten teilen, um ein korrektes Ergebnis zu erhalten. Jedoch kann nun nicht bestimmt werden, wie lange ein Auftrag laufen wird, wenn nicht genügend Knoten zur Verfügung stehen. Zwar könnte man ähnlich dem Backfilling-Verfahren für jeden Auftrag vorhersehen, wann genügend Knoten bereitstehen werden, und dann die in der Zukunft freien Knoten zur Bestimmung der Laufzeit nutzen. Dies ist aber eine unverhältnismäßige Verkomplizierung eines eigentlich eleganten Algorithmus.

Dieses Problem wird auch in [AFKT00] beschrieben, und es wird ohne nähere Erläuterung angegeben, den "statischen Bedarf" der Aufträge zur Selektion zu verwenden. Aus diesen Gründen werden in dieser Arbeit LPT und SPT nach dem Quotienten aus Bearbeitungszeit und Parallelität ausgewählt. Dies erscheint wie eine sinnvolle Interpretation von "statischem Bedarf".

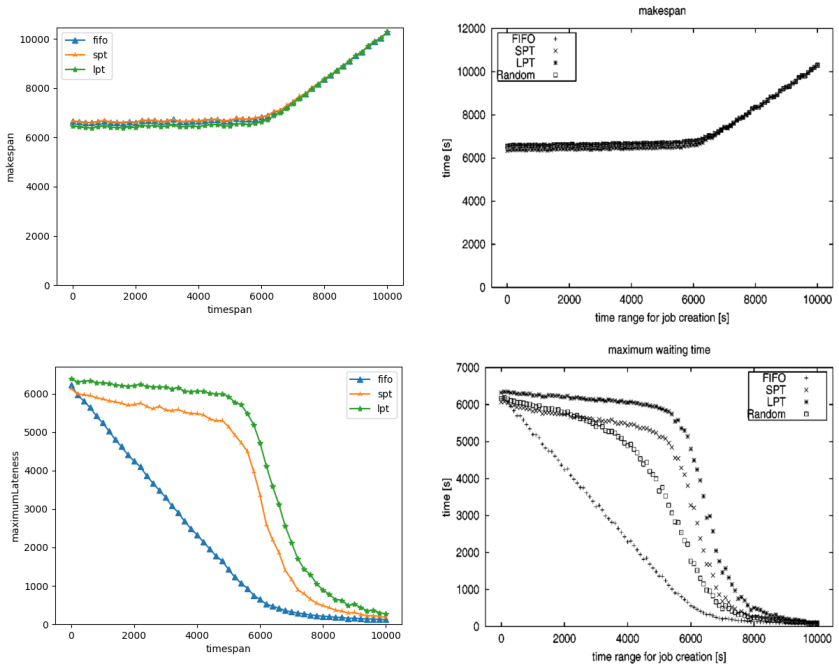
Grafischer Abgleich Experimente 1,2 und 3

Bevor das Simulationsmodell erweitert werden kann, sollte zuerst abgeglichen werden, ob das nachgebaute Modell näherungsweise ähnliche Läufe erzeugt. Dazu werden die in [AFKT00] beschriebenen Experimente nachgestellt, und die grafischen Auswertungen nebeneinander gestellt. Da die Untersuchung des Random Schedulingalgorithmus im Verlaufe von [AFKT00] von Arndt aufgegeben wurde und mehr Läufe benötigt werden, um stabile Werte zu erzeugen, wird dieser hier nicht betrachtet.
Abbildungen eins bis drei 3.1.1,3.1.2,3.1.3 lassen sich ohne Probleme, abgesehen von den nicht angegeben Geschwindigkeiten der Knoten, reproduzieren. Dabei steht links in Farbe die Reproduktion, und rechts in Schwarz und Weiß eine von Arndt et al. [AFKT00] übernommene Abbildung.



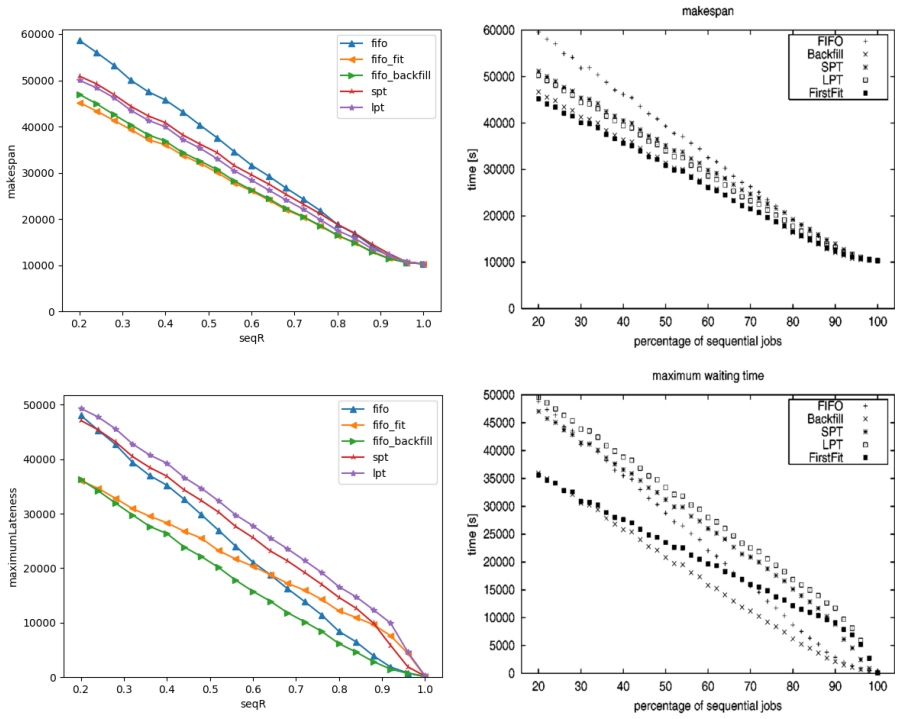
$P_s = 100\%$	$P_{lp} = —$
$t_s/1000 = 1 - 100$	$t_p/1000 = —$
$T_{C,max} = 0$	nodes = 10
$n = 250$	speed = equal

Abbildung 3.1.1: Figure 1, Vergleich von sequentiellen Aufträgen nach Bearbeitungsspanne und längster Wartezeit



$P_s = 100\%$	$P_{lp} = —$
$t_s/1000 = 1 - 50$	$t_p/1000 = —$
$T_{C,max} = 0 - 10000s$	nodes = 10
$n = 250$	speed = equal

Abbildung 3.1.2: Figure 2, Vergleich von sequentiellen Aufträgen, nach Bearbeitungsspanne und längster Wartezeit, Erstellungszeit wird variiert



$P_s = 20\% - 100\%$	$P_{lp} = 30\%$
$t_s/1000 = 1 - 50$	$t_p/1000 = 10 - 400$
$T_{C,max} = 10000$s	nodes $= 10$
$n = 250$	speed $=$ equal

Abbildung 3.1.3: Figure 3, Vergleich von Sequentiellen Aufträgen nach längster Bearbeitungsspanne und längster Wartezeit, Anteil an sequentiellen Aufträgen wird zwischen 20% und 100% variiert

Unterschiedlich schnelle Knoten ab Experiment 4

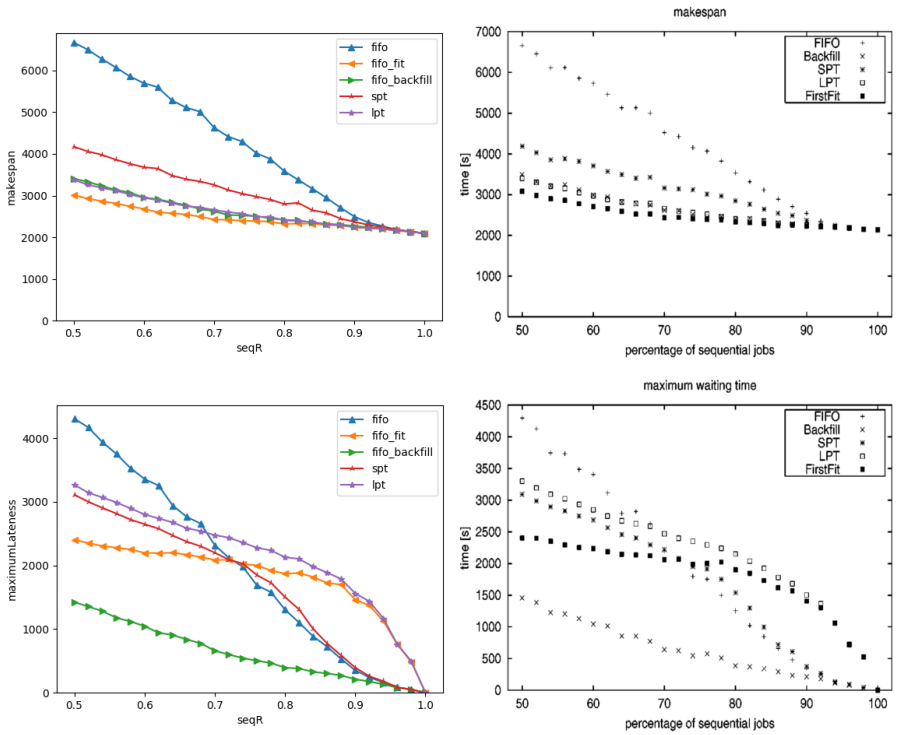
Ab dem vierten Experiment offenbart sich folgendes Problem: Die Geschwindigkeiten der Knoten variieren. Sie entsprechen den Messungen von "22 Knoten des lokalen Netzwerks". Jedoch werden diese gemessen Geschwindigkeiten weder in der besprochenen, noch in früheren Veröffentlichungen dargelegt [AFKT98, AFKT99]. Dies stellt eine hohe Hürde für die Reproduktion dar.

Um die in Kapitel 3.2.1 vorgeschlagenen Scheduler auch in den Experimenten 4 bis 9 bewerten zu können, wird eine ungefähre Verteilung der Geschwindigkeiten geschätzt. Hierfür wird Abbildung 5 3.1.4 herangezogen. In diesem Experiment wird die Bearbeitungsspanne (makespan) angegeben. Dadurch lässt sich eine untere Schranke für die gesamte Rechenleistung des Clusters finden. Wenn durch FirstFit jeder Knoten zu jedem Zeitpunkt ausgelastet wäre, müsste die durchschnittliche Rechenleistung aller Knoten im beschriebenen Versuchsaufbau 384 betragen. Dies wird am Wert für einen Auftragsmix aus 50% sequentiellen Aufträgen abgelesen. 100 Knoten unbekannter Geschwindigkeit benötigen etwa 3000 Sekunden um 500 Aufträge mit einer durchschnittlichen Bearbeitungszeit von 230500 Sekunden fertigzustellen. Um die durch die gierigen Algorithmen und die online eingehenden, also nicht von Anfang an bekannten Aufträge, entstehenden Effizienzeinbußen auszugleichen, wird die Geschwindigkeit der Knoten zusätzlich um 7,5% erhöht. Der Anteil an ungenutzter Rechenzeit variiert je nach Experiment zwischen 2% und über 50%. Bei den 7,5% handelt es sich also nicht um einen mathematisch begründeten Wert, sonder um einen empirisch ermittelten Wert, der plausible Ergebnisse liefert.

$$\frac{250 * (51 * 10^3 + 410 * 10^3) * s}{100x} = 3000s$$

$$\frac{250 * 461}{300} = x$$

Die Frage nach der Art der verwendeten Verteilung lässt sich nicht klären. Deswegen wird hier die einfachste Verteilung gewählt: 2 Klassen von Knoten, mit unbekannten Geschwindigkeiten und Anzahlen. Der zweidimensionale Lösungsraum kann durch eine weitere Annahme auf einen eindimensionalen Raum reduziert: Die Menge der langsamen Knoten soll die selbe Gesamtleistung besitzen wie die Menge der schnellen Knoten.
Dies führt zu einer größeren Anzahl langsamer und einer geringeren Anzahl schnellerer Knoten. Diese Konstellationen sind sicherlich interessanter als ihre Umkehrungen. Gäbe es viele schnelle und wenige langsame Knoten, wären ähnliche Ergebnisse dadurch zu erreichen, die langsamen Knoten wegzulassen, wodurch sich die Situation nicht viel von den zuvor untersuchten unterscheiden würde.



$P_s = 50\% - 100\%$	$P_{lp} = 30\%$
$t_s/1000 = 2 - 100$	$t_p/1000 = 20 - 800$
$T_{C,max} = 2000$s	nodes = 100
$n = 500$	speed = varying

Abbildung 3.1.4: Unterschiedlich schnelle Knoten:
75 Knoten mit Geschwindigkeit 275, 25 Knoten mit Geschwindigkeit 825

Damit bleibt nur eine einzige Variable, der Anteil der langsamen Knoten im Cluster. Der Wert 0,75 erzeugt visuell vergleichbare Ergebnisse.

Experimente 4 bis 9

Ab Experiment 4 kann keine Vergleichbarkeit der Modelle mehr sichergestellt werden. Trotzdem werden die Versuche nachgestellt, um die Algorithmen in 3.2.1 gegen die fünf bekannten antreten zu lassen. Die Auswertung findet in 3.2 statt.

3.2 WEITERFÜHRENDE UNTERSUCHUNGEN

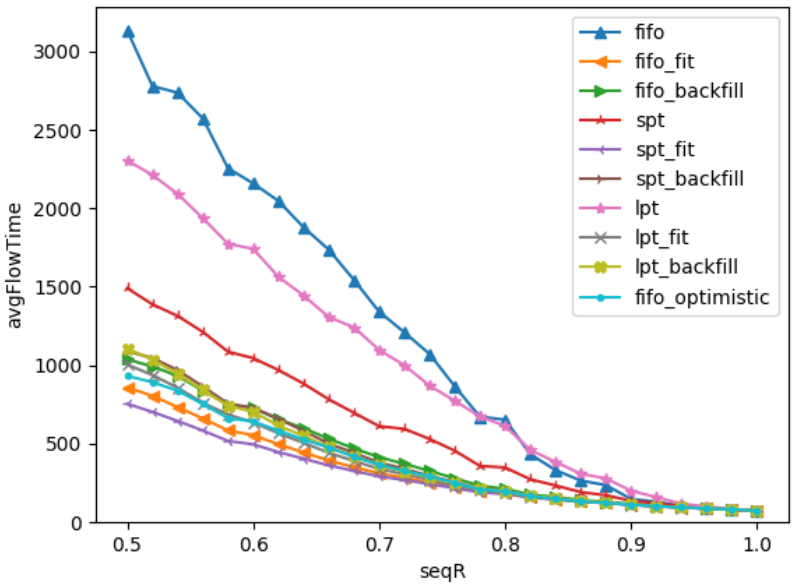


Abbildung 3.2.1: Variieren des Anteils von sequenziellen Aufträgen

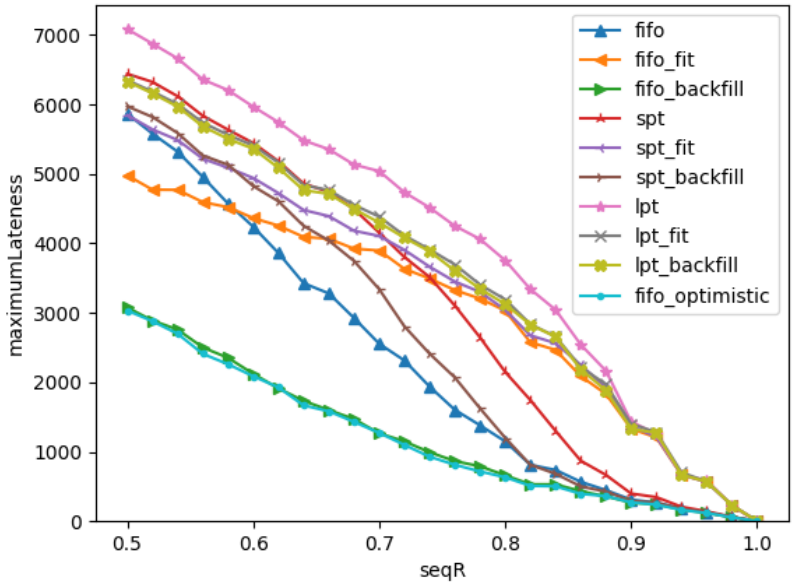


Abbildung 3.2.2: Variieren des Anteils von sequenziellen Aufträgen

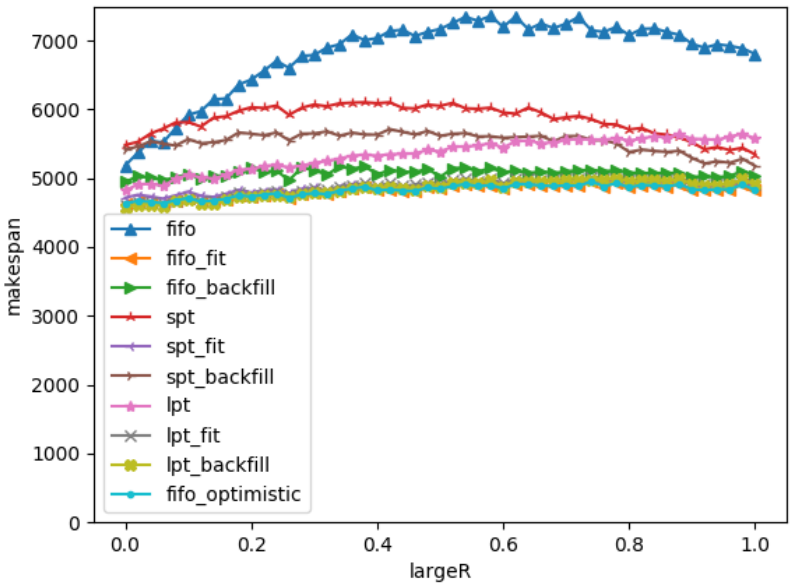


Abbildung 3.2.3: Variieren des Anteils von großen parallelen Aufträgen

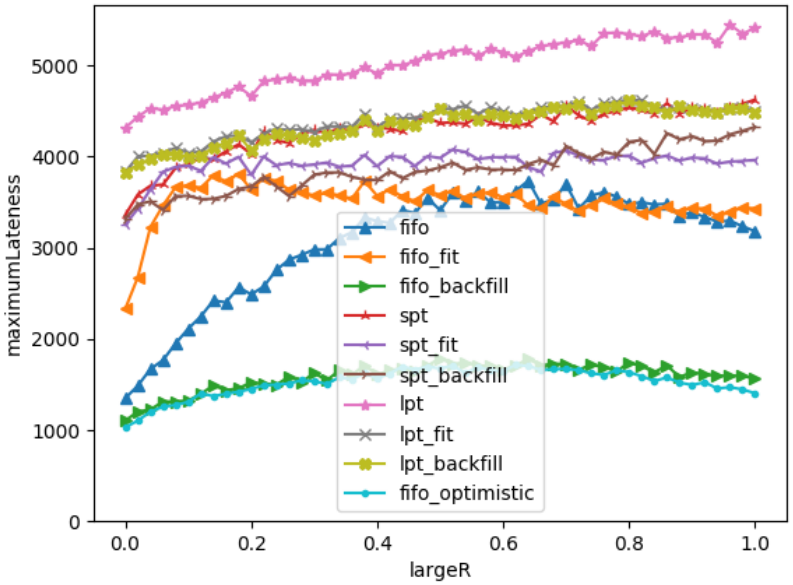


Abbildung 3.2.4: Variieren des Anteils von großen parallelen Aufträgen

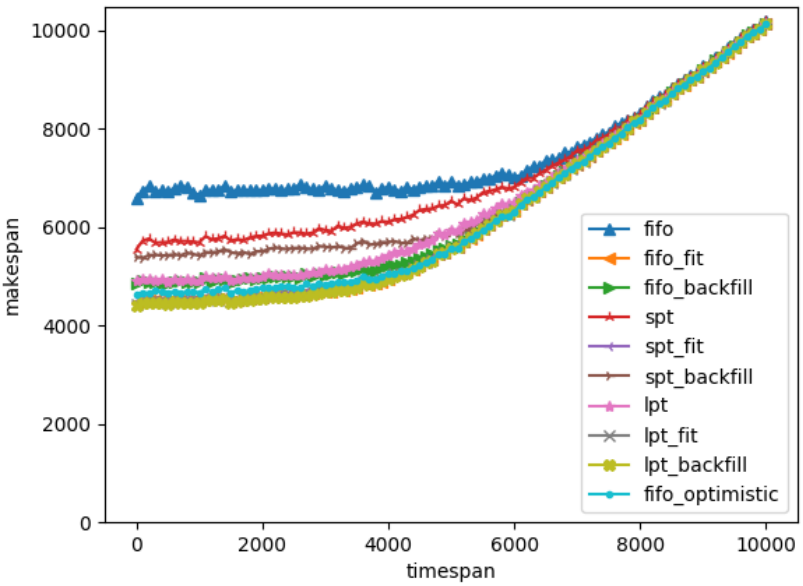


Abbildung 3.2.5: Variieren der Zeitspanne des Anmeldens von Aufträgen

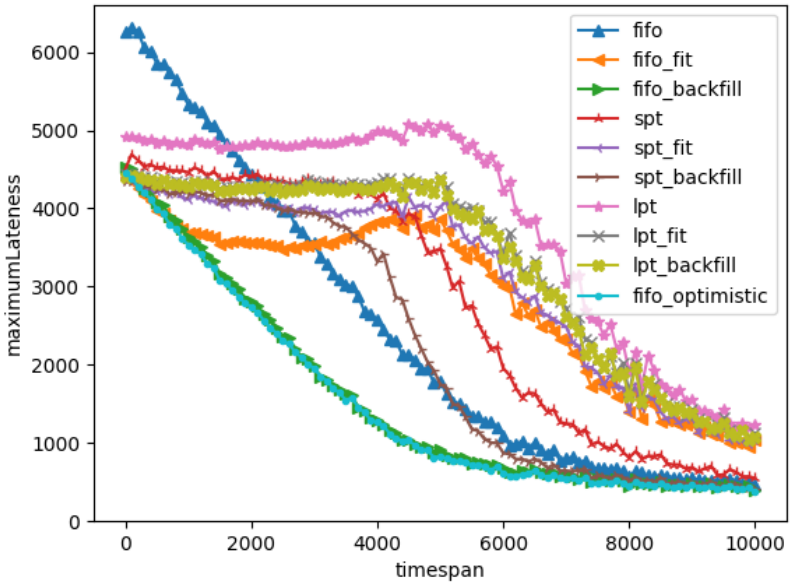


Abbildung 3.2.6: Variieren der Zeitspanne des Anmeldens von Aufträgen

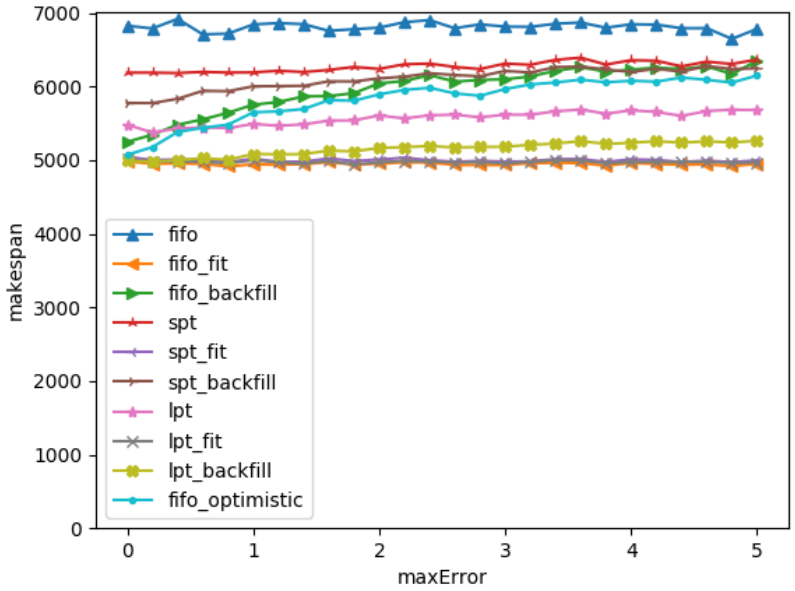


Abbildung 3.2.7: Variieren des erlaubten relativen Fehlers in der Angabe der Laufzeit

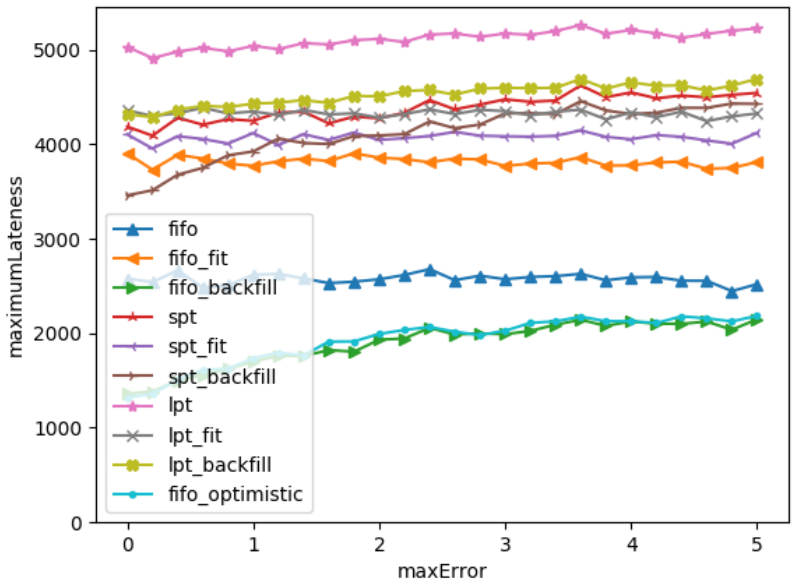


Abbildung 3.2.8: Variieren des erlaubten relativen Fehlers in der Angabe der Laufzeit

3.2.1 *Backfilling und Fit als Funktionen höherer Ordnung*

Die in Kapitel 2 vorgestellten Funktionen FirstFit und Backfilling sind nur Abwandelungen der FiFo Funktion. Beide wählen den nächsten Auftrag abhängig vom Einreihungszeitpunkt q_j, jedoch unter Einschränkungen. "FiFo-Fit" wählt den ersten unter den startbaren Aufträgen aus, "FiFo-BackFilling" wählt den ersten Auftrag, oder einen, der den Bearbeitungsbeginn des ersten Auftrags nicht verzögert.
"-Fit" und "-Backfilling" fügen also einen relevanten Kontext hinzu. Es liegt nahe, diese Erweiterungen als Funktionen höherer Ordnung zu betrachten. Ihre Domäne besteht aus einer Scheduling Funktion, den wartenden Aufträgen und einem Kontext. Der Kontext ist die Anzahl an verfügbaren Knoten oder die Abschlusszeitpunkte und die Parallelität der laufenden Aufträge. Die Scheduling Funktion, die Wartelist und der Kontext wird partiell auf einen Auftrag abgebildet. Da FirstFit-Backfilling von Arndt et al. als Sieger auserkoren wurde [AFKTo0, p. 10], LPT und SPT aber nicht im Backfilling Kontext untersucht wurden, wird im Folgenden untersucht, ob LPT oder SPT mit "-FirstFit" oder "-Backfilling" ähnlich gute Ergebnisse erzielen können.
Später 3.2.1 wird eine Abwandlung des Backfillings vorgestellt, das optimistische Backfilling. Dieses wird in dieser Arbeit auch mit untersucht, allerdings hauptsächlich im FiFo-Kontext.

SPT In Abbildung 3.2.1 wird die durchschnittliche Zeit eines Auftrags im System gemessen. Dabei schneidet SPT-Fit besser ab als alle anderen Algorithmen. Den zweiten Platz belegt FiFo-Fit. Für die durchschnittliche Systemzeit ist es wichtig, so schnell wie möglich Aufträge fertig zu stellen. Deshalb eigenen sich "-Fit" Funktionen, besonders in Kombination mit SPT. Erstaunlicherweise ist "-Fit" jedoch im Bezug zur Systemzeit nicht vollständig monoton. So gilt für die Messungen zwar $\text{SPT} < \text{LPT} < \text{FiFo}$, jedoch $\text{Fit}(\text{FiFo}) < \text{Fit}(\text{LPT})$.
In Abbildung 3.2.6 erzielt SPT-Fit die selben Werte wie FiFo-Fit und LPT-Fit. In diesem Experiment wird die Bearbeitungsspanne in Abhängigkeit von einer relativen Fehlerrate in der Angabe von Bearbeitungszeiten ermittelt.

LPT In Abbildung 3.2.3 erzielen mehrere Algorithmen ähnlich gute Ergebnisse. Darunter FiFo-Fit, SPT-Fit, LPT-Fit, LPT-Backfill und FiFo-Optimistic-Backfill. In diesem Experiment wird die Bearbeitungsspanne in Abhängigkeit des Anteils von großen parallelen Aufträgen gemessen. LPT-Fit ist hier deshalb eine gute Wahl, da LPT im Vergleich zu SPT und FiFo für eine geringere Spanne sorgt (vgl. auch 3.1.3). "-Fit" und "-Backfill" sorgt dafür, dass lange Phasen zum Anfang des Laufs, in denen einige Knoten nicht genutzt werden, während lange Aufträge abgearbeitet werden, gefüllt werden können.

Diese Beobachtung wiederholt sich in Abbildung 3.2.5. Auch hier wird die Bearbeitungsspanne als Maß angelegt.

OPTIMISTISCHES BACKFILLING Das vorgestellte "-Backfilling" Verfahren wirkt auf den ersten Blick zurückhaltend. Das angegebene Ziel, die zum "-Fit" verglichene Wartezeit gering zuhalten, wird erfüllt, indem große Aufträge nicht benachteiligt werden. Kleine werden nur vorgezogen, wenn sich dadurch die Wartezeit des besten, aber nicht startbaren Kandidaten, nicht verzögert. Dies wird erreicht, indem ein Auftrag P' nur starten darf, wenn er abgeschlossen wird, bevor der beste Kandidat P startet.
Warum aber darf ein Auftrag P', der wenige Knoten benötigt, nicht starten, vorausgesetzt, er nimmt nur so wenige Knoten in Anspruch, dass P wie geplant starten kann. Dies würden bedeuten, P benötigt noch n Knoten zum starten, sobald er starten wird, sind aber $n + k, k > 0$ Knoten frei, und $\pi_{P'} <= k$, oder $c_{P'} <= s_P$, falls P' sofort gestartet werden kann. Von Arndt et al. wird bezüglich dieser Idee auf [FW98] verwiesen.

Beim Optimistischen Backfilling stellt sich vor allem die Frage, wie groß der Unterschied zum "normalen" Backfilling ist, und ob es sich die Bearbeitungsspanne verringert werden kann. Hierfür wurde nur FiFo in beiden Kontexten herangezogen.
In Abbildung 3.2.1 bietet das Optimistische Verfahren einen leichten, jedoch erkennbaren Vorteil. Hier wird die durchschnittliche Systemzeit von Aufträgen in Abhängigkeit des Anteils sequentieller Aufträge ermittelt. Es erscheint nachvollziehbar, dass, vorausgesetzt es gibt etwa gleich viele sequenzielle und parallele Aufträge, eine Leistungssteigerung durch das Optimistische Verfahren erzielt werden kann. Das liegt daran, dass durch weniger Restriktionen mehr Lücken geschlossen werden können.
So auch in Abbildung 3.2.5. Hier wird die gesamte Bearbeitungszeit in Abhängigkeit eines erlaubten Fehlers untersucht. Die selbe Argumentation erscheint plausibel. Wenn die Bearbeitungszeit eines Auftrags nicht korrekt angegeben wurde, werden weniger Aufträge gestartet um Lücken zu füllen. Das Optimistische Backfilling startet Aufträge, die wenige Knoten benötigen und nutzt so die Freiräume besser aus.
In allen anderen Versuchen ist kein nennenswerter Unterschied feststellbar. Es scheint, als wären die in Abschnitt 3.2.1 genannten Konstellationen selten genug, als dass sie die nicht genutzten Chancen des "normalen" Backfilling überwiegen. Zumindest in den hier untersuchten Experimenten lässt sich die optimistische Variante als Ersatz empfehlen. In den meisten Fällen ist kein deutlicher Unterschied erkennbar, aber wenn, dann immer zu in Richtung geringerer Lauf-

zeiten. Die detaillierte Untersuchung der Auswirkung auf LPT und SPT übersteigen den Rahmen dieser Arbeit.

BACKFILLING UND FIT VON 2 FUNKTIONEN Darüber hinaus ist es natürlich auch möglich, Backfilling und Fit in Abhängigkeit von zwei Funktionen zu definieren. Das würde bedeutet, zuerst nach der ersten Funktion auszuwählen. Kann diese nicht gestartet werden, würde nach der zweiten Funktion aufgefüllt oder ein startbarer Auftrag gestartet. So könnten Schwächen einer Scheduling Funktion durch die Wahl einer anderen aufgewogen werden. Etwa könnten die längste Wartezeit des LPT-Backfill Algorithmus durch das Auffüllen nach FiFo verringert werden, die geringe Bearbeitungsspanne allerdings beibehalten werden. Alle Kombination zu vergleichen würde den Rahmen dieser Arbeit sprengen. Hier nun eine kurze erneute Betrachtung von Experiment 3.2.5.
Es wird das bekannte FiFo-Fit, FiFo-Backfilling und LPT-Backfilling sowie das neue LTP-FiFo-Optimistisch in Abbildung 3.2.9 betrachtet. Dieses versucht einen Auftrag nach LPT auszuwählen. Kann dieser nicht gestartet werden, wird ein anderer Auftrag nach FiFo ausgewählt, der das bekannte optimistische Auffüll-Kriterium erfüllt.
Es fällt auf, dass LPT mit Backfilling tatsächlich eine geringere Bearbeitungsspanne erzielt, als FiFo-Backfilling. Darüber hinaus, konkurriert LPT-Backfilling mit FiFo-Backfilling bezüglich der durchschnittlichen Systemzeit. Der LPT Algorithmus, der nach FiFo auffüllt, stellt sogar eine kleine aber klare Verbesserung dar. Es lässt sich also sagen, dass sich LPT, insbesondere in Kombination mit FiFo, sehr gut eignet, um die Bearbeitungsspanne gering zu halten. Auch scheint das Auffüllen nach FiFo geringfügig fairer zu sein, als das normale Auffüllen mit LPT. Allerdings reicht dies nicht aus, um die Stärke des FiFo-Backfillings, nämlich die gute Ausnutzung der Ressourcen und die geringe längste Wartezeit, zu erreichen.

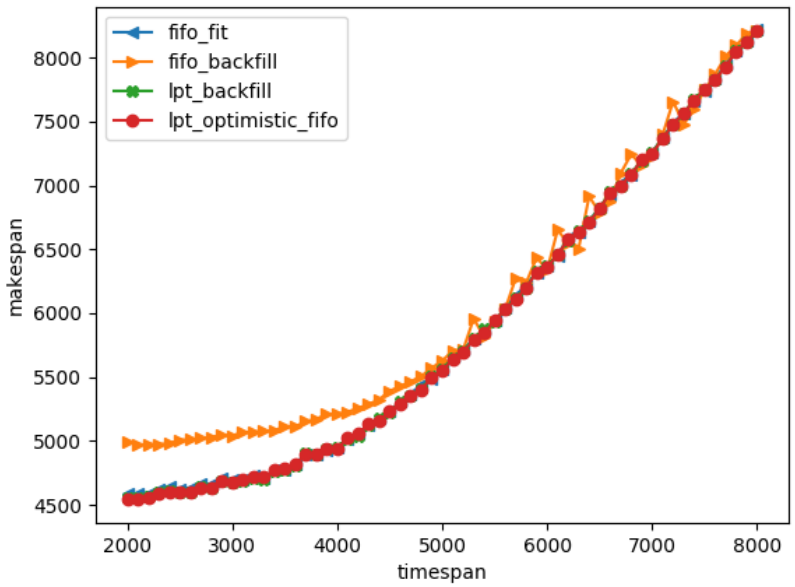


Abbildung 3.2.9: Bearbeitungsspanne abhängig vom Anmeldezeitraums

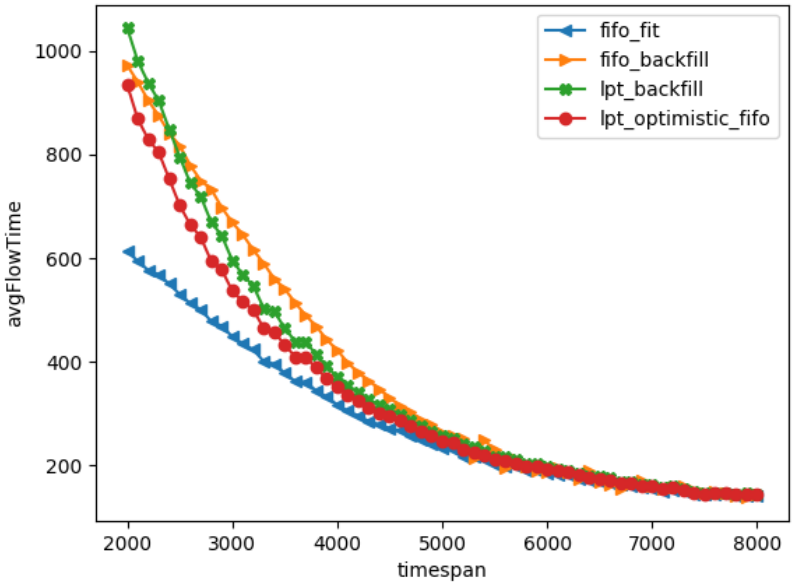


Abbildung 3.2.10: Durchschnittliche Systemzeit abhängig vom Anmeldezeitraums

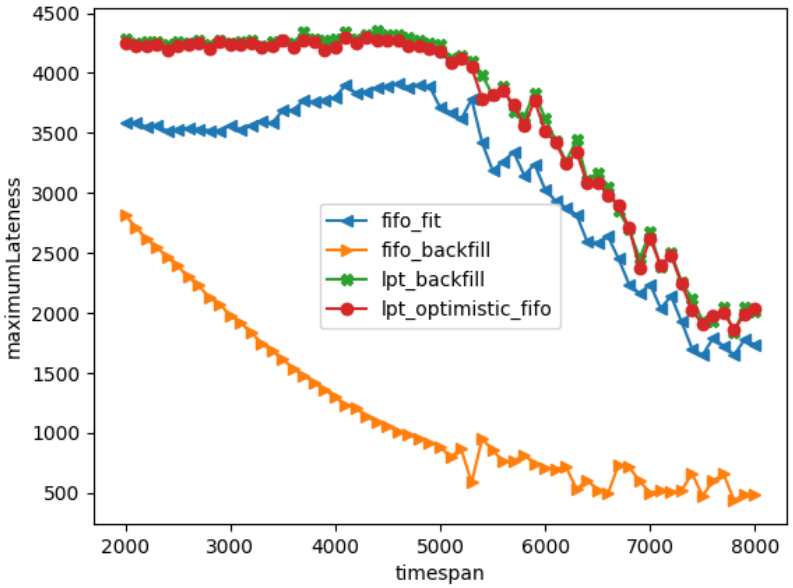


Abbildung 3.2.11: Längste Wartezeit abhängig vom Anmeldezeitraums

3.2.2 *Property Based Testing zum Erkenntnisgewinn*

TESTEN STATT SIMULLIEREN Während das Simulieren des Rechenclusters gute Auskunft darüber gibt, wie sich das System bei einer langen Reihe an Aufträgen verhält, kristallisiert sich dabei lediglich das durchschnittliche Verhalten heraus. Obwohl diese Analyse sinnvoll ist, um verschiedene Scheduling Funktionen asymptotisch miteinander zu vergleichen, wird durch das Mitteln von hunderten von Läufen nicht ersichtlich, in welchen Fällen eine normalerweise unterlegene Scheduling Funktion einer anderen überlegen ist.
Um eine Intuition für die Unterschiede zwischen Scheduling Funktionen entwickeln zu können ist es hilfreicher, die durch kleinstmögliche Listen an Aufgaben erzeugten Läufe zu vergleichen. Um diese minimalen Beispiele zu generieren, kann ein Property Based Testing Framework verwendet werden. Die zukunftsweisende Arbeit dieses Themas ist *Quickcheck: a lightweight tool for random testing of haskell programs* [HCoo]. Hier erfolgt nur eine kleine Zusammenfassung der Arbeit von John Hughes und Koen Claessen. Es werden nur die für diese Arbeit wichtigsten Aspekte betrachtet.
Ein Test besteht aus einer zu testenden Funktion, einer Eigenschaft, die die Ausgabe der Funktion haben soll, und einem Generator, der Eingaben produziert. Sobald eine Eingabe gefunden wird, deren Ausgabe die geforderte Eigenschaft nicht erfüllt, wird die Eingabe automatisch geschrumpft. Dies führt zu einem leichter interpretierbaren Beispiel, da der ausgegebene Fall "kein Rauschen" enthält. Eine Zahl wird geschrumpft, indem ihr Betrag reduziert wird, ein Tupel von

Zahlen, indem eines der Elemente geschrumpft wird, und eine Liste, indem Elemente der Liste weggelassen oder geschrumpft werden.

Um ein minimales Beispiel zu finden, in dem Scheduling Funktion S_1 Ziel Funktion T besser minimiert als Scheduler S_2, können wir die Eigenschaft $T(S_1(x)) >= T(S_2(1))$, mit einem Auftragsgenerator X: $(Anzahlkonten, [(\pi_j <= Anzahlknoten, p_j, q_j)])$ überprüfen. Ein vom Testframework gefundenes minimales Gegenbeispiel, zeigt uns einen speziellen Fall, in dem S_1 ein besseres Ergebnis erzielt als S_2 ([Mac]). Für die meisten Scheduling Funktionen ist das Minimalbeispiel eines mit 3 Aufträgen, und einer kleinen Anzahl an Knoten.
Hier nun einige Beispiele:

OPTIMISTISCHES BACKFILLING Das optimistische Backfilling wurde bereits in Kapitel 3.2.1 vorgestellt. Dabei werden Lücken aggressiv geschlossen, wenn der bestmögliche Auftrag nicht startbar ist. Es kann ein anderer Auftrag gestartet werden, wenn dieser abgeschlossen werden kann, bevor der beste Kandidat gestartet werden wird, oder wenn er weniger Knoten benötigt, als nach dem starten des besten Kandidaten noch ungenutzt sein werden.
Zunächst mag es erscheinen, als wäre dies eine klare Verbesserung des Backfilling Algorithmus, da weniger Restriktionen bestehen, um Lücken zu schließen. Es ist intuitiv, dass es einen Haken gibt. Es ist aber nicht einfach, aus dem Stand ein Beispiel zu konstruieren, in dem sich das optimistische Backfilling negativ auf die Wartezeit auswirkt. Allerdings kann dank Property Based Testing eines in wenigen Sekunden generiert werden.

```
Is fifo\_optimistic allways better than
 fifo\_backfill by maximumLateness?
No! counterexample:
queueintT, processingT, realProcessingT, degreeOfParallelism
id: 0, qT: 1, pT: 3, rPT: 3, doP: 1
id: 1, qT: 0, pT: 3, rPT: 3, doP: 2
id: 2, qT: 0, pT: 1, rPT: 1, doP: 2
id: 3, qT: 0, pT: 1, rPT: 1, doP: 3

maximumLateness of fifo\_optimistic: 4
[0]:112|-3
[1]:112|-3
[2]:-00|03

maximumLateness of fifo\_backfill: 3
[0]:112|3--|-
[1]:11-|300|0
[2]:--2|3--|-
```

Abbildung 3.2.12: Vergleich vom Normalem und Optimistischem Backfilling

Hier ist der optimistische Algorithmus zu voreilig. Auftrag 0 wird gestartet, obwohl er (als einziger) nicht seit Beginn angemeldet ist. Dadurch wird Auftrag 3 nach hinten geschoben und die maximale Wartezeit für 3 steigt. Trotzdem ist die Auslastung des optimistischen Algorithmus deutlich besser. Doch existiert auch ein Fall, in dem sowohl die maximale Verspätung, als auch die gesamt Bearbeitungszeit schlechter sind. Dieser ist kaum komplizierter als der letzte.

Auch hier kann ein durch Property Based Testing ein Fall konstruiert werden. Allerdings besteht hier das kleinste gefundene Beispiel bereits aus 5 Aufträgen. Es ist also zu erwarten, dass solche Konstellationen selten genug sind. Eine genauere Untersuchung dazu kann im Abschnitt 3.2.1 gefunden werden.

```
Is fifo_optimistic allways better than <function System.fifo_backfill at 0
No! counterexample:

queueintT, processingT, realProcessingT, degreeOfParallelism
id: 0, qT: 1, pT: 1, rPT: 1, doP: 4
id: 1, qT: 1, pT: 3, rPT: 3, doP: 1
id: 2, qT: 1, pT: 3, rPT: 3, doP: 1
id: 3, qT: 0, pT: 4, rPT: 4, doP: 3
id: 4, qT: 0, pT: 1, rPT: 1, doP: 2

maximumLateness offifo_optimistic: 4
makespan of fifo_optimistic: 8
[0]:334|-0-|--
[1]:334|-0-|--
[2]:33-|-02|22
[3]:-11|10-|--

maximumLateness offifo_backfill: 3
makespan of fifo_backfill: 7
[0]:334|0--|-
[1]:33-|011|1
[2]:33-|022|2
[3]:--4|0--|-
```

Abbildung 3.2.13: Vergleich vom Normalem und Optimistischem Backfilling in Spanne und Verspätung

Wie in der Grafik 3.2.12 nach einem Schritt erkennbar, benachteiligt ein solcher optimistisch gestarteter langer (also hohe Bearbeitungszeit), kleiner (also wenige Knoten notwendig) Auftrag nicht den besten Kandidaten (Auftrag 4), allerdings den zweitbesten Kandidaten (Auftrag 0). Ob dies ein seltener Fall ist, oder ob sich Optimismus im Mittel auszahlt, kann wiederum durch eine Statistische Auswertung untersucht werden.

Außerdem ist es interessant zu beobachten, welche Informationen implizit aus diesem Gegenbeispiel gezogen werden können. Zum Beispiel, dass im ersten Lauf 3.2.12 nur ein einziger online Auftrag (d.h. nicht schon zum Zeitpunkt 0 bekannt) nötig ist, aber um das optimistische Verfahren in beiden Metriken zu schlagen, werden 3 online Aufträge sowie ein zusätzlicher Auftrag benötigt. Es fällt auch auf, dass in diesem Beispiel FiFo-Backfill nie vom Backfilling Gebrauch macht. Es verhält sich immer wie das blanke FiFo Verfahren. Es kann auch einen Auftragsmix geben, in dem das normale Backfilling sowohl FiFo als auch das optimistische Backfilling trumpft, und zwar sowohl was die Bearbeitungsspanne als auch die längste Wartezeit angeht. Es erfordert eine sehr hohe Anstrengung, ein solches von Hand zu suchen.

Des Weiteren fällt auf, dass das Shrinking Verfahren herausgefunden hat, dass zur Berechnung der Laufzeiten von parallelen Aufträgen aufgerundet wird (s. 3.1.2). Da dies der Lesbarkeit der Tests zuwider läuft, kann die Bearbeitungszeit jedes Auftrags einfach auf das nächste Vielfache der Parallelität gesetzt werden. Zusätzlich wird oft die Gleichstände-auflösenden IDs zurückgegriffen. Es kann explizit gefordert werden, dass alle Einreihungszeitpunkte der Aufträge verschieden sind. Allerdings verlangsamt dies die Suche nach einem Gegenbeispiel, und das kleinste gefundene Gegenbeispiel hat eine Bearbeitungsspanne von 13 Zeiteinheiten oder mehr.

```
id: 0, qT: 0, pT: 1, rPT: 1, doP: 2
id: 1, qT: 0, pT: 1, rPT: 1, doP: 3
id: 2, qT: 0, pT: 2, rPT: 2, doP: 1
id: 3, qT: 0, pT: 1, rPT: 1, doP: 3
id: 4, qT: 0, pT: 1, rPT: 1, doP: 2

maximumLateness of fifo_optimistic : 3
makespan of fifo_optimistic : 4
[0]:013|4
[1]:013|4
[2]:223|-
[3]:-1-|-

maximumLateness of fifo_backfill : 2
makespan of fifo_backfill : 3
[0]:013
[1]:013
[2]:413
[3]:422

maximumLateness of fifo : 3
makespan of fifo : 4
[0]:013|4
[1]:013|4
[2]:-13|-
[3]:-22|-
```

Abbildung 3.2.14: Backfill besser als Optimistic und Fifo, bez. Bearbeitunsspanne und längster Wartezeit

3.2.3 *Vergleich von Simulation und PBT*

Die Simulation vergleicht Scheduling Algorithmen anhand ihrer Leistung beim Bearbeiten von - konstruierten - Auftragszusammenstellungen. Das PBT findet zu Scheduling Algorithmen einen dazu passenden Auftrags- und Knotenmix, in dem sich die Leistungen unterscheiden. Rückschlüsse können basierend auf der Komplexität dieser Szenarien gezogen werden. Die Vermutung, dass komplizierte oder gestellt Konstellationen im Normalbetrieb selten sind, und dass ein Algorithmus, der nur in solchen Situationen zu schlagen ist, sich auch im Normalbetrieb bewährt, ist noch zu überprüfen.

Vorab ist zu wählen, nach welcher Zielfunktion bemessen werden soll. Da auch [AFKT00] sich vor allem auf die Bearbeitungsspanne konzentrieren, wird das hier auch getan. Außerdem ist zu klären, was die kleinste Situation ist, in dem zwei verschieden Scheduler zwei verschiedene Ergebnisse liefern. Offensichtlich müssen dafür mindestens zwei Knoten beteiligt sein, und mehr als 2 Aufträge vorhanden sein. Zunächst wird diese Überlegung überprüft. Dafür wird ein neuer Scheduler ”FId” geschaffen, der immer den Auftrag mit der niedrigsten Id auswählt. Dann wird nach einem Test gesucht, in dem sich ein FId System anders verhält, wenn die IDs in der Auftragsliste vor Start der Simulation permutiert werden. Diese Vermutung lässt sich schnell durch einen Versuch bestätigen (s. 3.2.15).

Nun können die bisher vorgestellten Scheduler miteinander verglichen werden. Die "Gestelltheit"des kleinsten Beispiels setzt sich dabei aus der Anzahl an Aufträgen, Knoten, und online Aufträgen zusammen. Hier sei noch angemerkt, dass dies, ebenso wie das Simulieren, keine exakten Ergebnisse liefert. Es ist immer möglich, dass es einen noch einfacheren Fall gibt, den das Shrinking Verfahren nicht gefunden hat. Es werden für jede Paarung 100.000 Beispiele ausprobiert. Ebenso gibt es mehrere kleinste Beispiele. Zum Beispiel kann FiFo LPT mit 3-2-1 oder 4-2-0 schlagen. Das Shrinking Verfahren betrachtet dabei ersteres Ergebnis als "kleiner". Hierfür kann ”gezieltes PBT” ([MD]) verwendet werden. Hierbei wird ein Wert angegeben, den das Test Framework minimiert, zum Beispiel die Differenz zwischen den zwei Bearbeitungsspannen. Dadurch verringert sich zwar die Zeit, die gebraucht wird, um ein Gegenbeispiel zu finden, allerdings variiert dadurch die Größe der Beispiele stärker. Aus diesem Grund wird darauf hier verzichtet.

Hierbei ist zu beachten, dass FiFo nie seine Backfill-Variante besiegen kann. Dies liegt daran, dass das Auswahlkriterium von FiFo die Einreihungszeit ist. Per Definition kann also kein online eingereihter Auftrag erscheinen, der dieses Kriterium besser erfüllt, als die bereits

```
makespan : 2
id: 0, qT: 0, pT: 1, rPT: 1, doP: 2
id: 1, qT: 0, pT: 1, rPT: 1, doP: 1
id: 2, qT: 0, pT: 1, rPT: 1, doP: 1
[0]:01
[1]:02

makespan : 3
id: 0, qT: 0, pT: 1, rPT: 1, doP: 1
id: 1, qT: 0, pT: 1, rPT: 1, doP: 2
id: 2, qT: 0, pT: 1, rPT: 1, doP: 1
[0]:012
[1]:-1-
```

Abbildung 3.2.15: Minimales Beispiel, in dem sich die Wahl der Scheduling Funktion auswirkt

J+k+o	fifo	spt	lpt	Fifo-fit	Fifo-back	Fifo-optim	Spt-fit	Spt-back	Lpt-fit	Lpt-back
fifo	+	3+2+0	3+2+1	6+2+2	X	8+3+4	3+4+0	3+2+0	3+2+1	3+9+1
spt	3+2+1	+	3+2+1	4+2+1	4+2+1	4+2+1	6+8+0	6+10+5	3+3+1	4+5+3
lpt	3+2+0	3+2+0	+	4+4+2	3+2+0	3+2+2	3+2+0	3+2+0	4+2+2	7+5+2
Fifo-fit	3+2+2	3+2+2	3+2+0	+	5+4+2	3+2+1	3+2+0	3+2+0	3+2+1	4+3+0
Fifo-back	5+2+0	3+2+2	3+2+0	5+6+4	+	7+6+2	3+3+0	3+2+2	3+2+1	3+2+1
Fifo-optim	3+3+1	3+2+0	3+2+0	4+2+1	3+4+0	+	3+4+0	3+2+2	3+2+1	3+2+1
Spt-fit	3+2+2	3+2+2	3+2+0	3+2+1	3+2+2	3+2+0	+	3+3+0	3+2+1	5+5+4
Spt-back	3+2+0	5+2+2	3+2+1	3+2+1	3+2+1	3+2+1	5+4+2	+	3+2+1	3+2+1
Lpt-fit	3+2+2	3+2+1	3+2+0	3+2+0	3+2+1	3+2+0	3+2+0	3+2+0	+	4+4+3
Lpt-back	3+2+1	3+2+2	3+3+0	3+2+0	3+4+1	4+2+1	3+2+0	3+2+0	5+4+2	+

Tabelle 3.2.1: Größe des kleinsten gefundenen Falls, in dem der Spaltenkopf eine kleinere Spanne hat als der Zeilenkopf

	V/G	fifo	spt	lpt	Fifo-fit	Fifo-back	Fifo-optim	Spt-fit	Spt-back	Lpt-fit	Lpt-back	
0.756	fifo	0	5	6	10	nn	15	7	5	6	13	8.375
0.667	spt	6	0	6	7	7	7	14	21	7	12	9.667
0.766	lpt	5	5	0	10	5	7	5	5	8	14	7.111
1.203	Fifo-fit	7	7	5	0	11	6	5	5	6	7	6.556
0.867	Fifo-back	7	7	5	15	0	15	6	7	6	6	8.222
1.281	Fifo-optim	7	5	5	7	7	0	7	7	6	6	6.333
1.032	Spt-fit	7	7	5	6	7	5	0	6	6	14	7
1.082	Spt-back	5	9	6	6	6	6	11	0	6	6	6.778
1.127	Lpt-fit	7	6	5	5	6	5	5	5	0	11	6.111
1.483	Lpt-back	6	7	6	5	8	7	5	5	11	0	6.667
		6.333	6.444	5.444	7.889	7.125	8.111	7.222	7.333	6.889	9.889	

Tabelle 3.2.2: Durchschnittliche Größe jeder Zeile und Spalte; links das Verhältnis aus durchschn. Gewinner und Verlierer Größe

bekannten Aufträge. Für SPT und LPT gilt dies nicht.
Aus diesem Grund wird hier der Durchschnitt der nicht-null Werte verrechnet. Dies benachteiligt FiFo-Backfill und bevorteilt FiFo. Alternativ wäre eine Betrachtung ohne den FiFo Algorithmus möglich.
Als zweites fällt die Wahl eines Versuchsaufbau an, mit dem der Property Based Testing score (pbt-score) verglichen werden soll. Das PBT Werkzeug darf folgende Freiheiten ausnutzen: Anzahl an Knoten, Dauer, Einreihungszeitpunkt und Parallelität der Aufträge. Fest steht die Geschwindigkeit der Knoten, in diesem Fall der Lesbarkeit wegen 1. Ebenso wurden alle Bearbeitungszeiten korrekt angegeben. Aus diesem Grund wird Experiment 3 gewählt 3.1.3. In diesem haben alle Knoten die selbe Geschwindigkeit, der Anteil an sequentiellen Aufträgen wird variiert und 30% der parallelen Aufträge nehmen mindestens die Hälfte der Knoten in Anspruch. Das Experiment wird wiederholt, diesmal mit zusätzlichen Scheduling Algorithmen. Es wird ein quantitativer Wert zur Berechnung der Korrelation benötigt. Hierfür wird das Integral der verschiedenen Grafen berechnet, anstatt die Bearbeitungsspanne an einem willkürlich gewählten Punkt zu messen. Die Ergebnisse befinden sich in Tabelle 3.2.3. Der *Rangkorrelationskoeefizients nach Spearman* von *-1* legt eine starke negative Korrelation nahe.

algorithm	area under curve	pbt score	a.u.c. Rank	pbt Rank
fifo	25862.6322330097	0.756218905472637	9	2
spt	23798.8361165049	0.666666666666667	10	1
lpt	23054.2840776699	0.765625	8	3
Fifo-fit	21039.4811650485	1.20338983050847	3	8
Fifo-back	21378.4883018868	0.866554054054054	7	4
Fifo-optim	21008.5106796117	1.28070175438597	2	9
Spt-fit	21211.5052427184	1.03174603174603	6	5
Spt-back	23685.2388349515	1.08196721311475	5	6
Lpt-fit	21073.0885436893	1.12727272727273	4	7
Lpt-back	20213.0893203883	1.48333333333333	1	10

Tabelle 3.2.3: Vergleich von Experiment 3 und pbt-score; Rangkorrelations nach Spearman von -1

Vergleicht man die Tabelle 3.2.3 mit 3.2.3 fällt auf, dass die ermittelten Werte bezüglich der kleinsten Szenarien nur begrenzte Vorhersagekraft besitzen, sobald die Knoten unterschiedliche Geschwindigkeiten aufweisen. Die referenziell Transparenten Scheduler schneiden schlecht ab, die Backfilling- und Fit Allogrithmen besser, mit Ausnahme von SPT-Backfilling. Generell kann dieser Art der Analyse ohne weitere Untersuchungen nicht zu viel Vertrauen geschenkt werden. Der Versuch könnte zwar wiederholt werden, diesmal mit den Geschwindigkeiten der Knoten als freie Parametern. Dies könnte Ergebnisse liefern, die stärker mit den Experiment 4 bis 9 korrelieren. Es bleiben dennoch weitere Kritikpunkte. So ist nicht klar, ob die simple Formel der Größe eines Experiments, nämlich $\mathtt{Aufträge} + \mathtt{Knoten} + \mathtt{OnLineAufträge}$ ein sinnvolles Maß ist. Möglicherweise wäre die euklidische Entfernung zum $3-2-0$ Minimum eine bessere Metrik. Auch bestehen mehrere Möglichkeiten mit FiFo und FiFo-Backfilling umzugehen. Aufgrund des niedrigen Durchsatzes von FiFo könnte dieser ganz aus dem Experiment entfernt werden. Die hier getroffenen Wahlen erzielten zwar direkt ein ungeahntes hohes Maß an Korrelation, trotzdem könnten andere Entscheidungen im Versuchsaufbau und in der Auswertung auch gut begründet werden.
Wahrscheinlich wurde nicht zu jedem Paar die tatsächlich kleinste Konstellation an Aufträgen gefunden. Hierfür würde sich ein traditionelles Modellchecking Werkzeug wie *TLA+*, von Hillel Wayne in *Practical TLA+ Planning Driven Development* [Way18] vorgestellt, besser eigenen. Dieses kann eine vollständige Breitensuche durchführen. Dies würde natürlich mehr Rechenleistung benötigen, im Gegenzug dafür aber auch exakte Ergebnisse liefern. Ein weiterer Vorteil eine Formalen Spezifikation des Clusters wäre natürlich die einfachere Reproduktion durch dritte (vgl. Kaptiel 3.1.2).
Es ist fraglich, wieweit die Methode des kleinsten Falls, in dem ein Algorithmus gegen einen anderen die Oberhand gewinnt, für die Analy-

se der verschiedenen Leistungen taugt. Sicherlich ist sie ein sehr nützliches Hilfsmittel, das neue Einsichten gewähren kann. Sich schnell darüber klar werden zu können, in welchen Situationen ein (ausgeklügelter) Algorithmus wie das Backfilling sich selbst überlistet, ist sicherlich eine gute Möglichkeit die eigene Intuition weiterzuentwickeln. Die intuitive Vermutung, dass "komplizierte" Konstellationen in Experimenten seltener auftreten, scheint wahr zu sein.

Die wichtigste Erkenntnis dieser Untersuchung ist, wie einfach es ist, Experimente zu entwerfen, in denen ein Menge an Scheduling Algorithmen eine beliebige Rangordnung erzielen. Daraus folgt, dass Experimente, die die Leistung verschiedener Algorithmen messen, gut motiviert seien müssen. Idealerweise sollte sich ein Versuchsaufbau also von einer Beobachtung, realer Cluster und Aufträge, ableiten.

4

VERWANDTE ARBEITEN

A COMPARATIVE STUDY OF ONLINE SCHEDULING ALGORITHMS FOR NETWORKS von Olaf Arndt, Bernd Freisleben, Thilo Kielmann und Frank Thilo [AFKT00] stellt die Grundlage da, auf der diese Arbeit aufbaut. Darin wird ein Simulationsmodell vorgestellt, verschiedene Scheduling Algorithmen in diversen Experimenten untersucht und die Ergebnisse an einem echten Rechencluster überprüft. Backfilling Abwandlungen werden nicht betrachtet. Es wird am meisten Wert auf Bearbeitungsspanne und längster Wartezeit gelegt. Zur Überprüfung wurde eine Auftragsliste der Länge 100, aus fünf verschiedenen Programmen zusammengesetzt.

UTILIZATION AND PREDICTABILITY IN SCHEDULING THE IBM SP2 WITH BACKFILLING von Feitelson, Dror G and Weil, Ahuva Mu'alem [FW98] vergleicht Scheduling Methoden auf einem IBM SP2 System. Optimistisches (wrtl. aggressives) FiFo-Backfilling wird abgelehnt. Die Möglichkeit, für jeden Auftrag in der Warteliste vorherzusagen, wann dieser gestartet wird, wird in der Arbeit als wichtig erachtet. Es wird, auf wird kein ausschlaggebender Unterschied bezüglich der Auslastung zwischen beiden Algorithmen festgestellt. Außerdem wird eine neue Metrik vorgestellt. Die Zielfunktion wird die *durchschnittliche Drosselung* (eng. *slowdown*). Dabei wird die *Drosselung* eines Auftrags j mit $slowdown(j) = \frac{s_j - q_j}{max(p_j, 10)}$ bestimmt. Dies stellt ein Maß für Fairness da. Die Wartezeit eines Auftrags soll proportional zu seiner Bearbeitungszeit sein.

ATTACKING THE BOTTLENECKS OF BACKFILLING SCHEDULERS von Keleher, Peter J and Zotkin, Dmitry and Perkovic, Dejan [KZP00] untersucht Abwandelungen des Backfilling Algorithmus. Auch hier wird die Drosselung zum Ziel genommen. Der im Namen der Arbeit betitelte Flaschenhals bezeichnet nicht geschlossene Lücken, die beim konservativen Backfilling auftreten können. Die Untersuchungen wurden von der kontraintuitiven Beobachtung angestoßen, dass in echten Rechenclustern verfälschte Angaben von Bearbeitungszeiten die Gesamtleistung des Systems verbessern können. Dieses Phänomen wurde weder in der Arbeit Arndt et al. [AFKT00] noch in dieser Arbeit festgestellt 3.2.7. Keleher et al. schreiben dazu, dass die Bearbeitungszeit von Aufträgen in realen Situationen selten gleichverteilt ist und dass, der Anteil an langen und großen Aufträgen normalerweise relativ klein ist.
Es werden Abwandlungen vorgeschlagen, die das Backfilling weniger restriktiv handeln lassen. Es wird LPT-Backfilling untersucht und die

gute Ausnutzung der Ressourcen gelobt. Auch wird zufällinges Auffüllen betrachtet. Zuletzt wird das optimistische Backfilling evaluiert. Auch hier wird eine messbare Leistungssteigerung vermerkt.

BACKFILLING USING SYSTEM-GENERATED PREDICTIONS RATHER THAN USER RUNTIME ESTIMATES von Tsafrir, Dan und Etsion, Yoav und Feitelson, Dror G untersucht den Zusammenhang zwischen falschen Laufzeitvorhersagen und der Leistung von Backfilling. Es wird dargelegt, dass überschätze Laufzeiten dazu führen, dass auftretende Lücken länger erscheinen, als sie sind. Dies erlaubt einem konservativen Backfilling Scheduler, aggressiver kurze Aufträge vorzuziehen. Daraus leitet sich eine erhöhte Leistung bezüglich der Drosselung ab. Es wird vorgeschlagen, Backfilling nach SPT einzusetzen. [TEF07]

5

FAZIT

SCHEDULING FUNKTIONEN HÖHERER ORDNUNG Die Sichtweise der Funktionalen Programmierung auf Scheduler ist offensichtlich nützlich. Arndt et al. [AFKToo] haben zurecht den Backfilling Algorithmus als bevorzugten Scheduler gewählt. Jedoch lassen sich zu den meisten vorgestellten Experimenten Kombinationen der Scheduler finden, die diese schlagen. Die Menge der möglichen Kombination ist hierbei 80, wenn zu LPT und SPT auch noch die Äquivalenten hinzu genommen werden, die die Bearbeitungszeit anstelle der Laufzeit beachten. Bei dieser Menge an Möglichkeiten ist eine qualitative Analyse wie in [AFKToo] oder in dieser Arbeit nicht mehr möglich. Da durch geschickte Kombinationen geringe Verbesserungen in einer Metrik im Gegenzug zu schwerer Verschlechterung in anderen Metriken erreichbar sind, müsste für so eine Untersuchung eine Gewichtung der Metriken gefunden werden. Diese könnte dann in verschiedenen Experimenten für alle Kombinationen ermittelt werden.

VERGLEICH VON SIMULATION UND TESTING Es ist fraglich, wieweit die Methode des kleinsten Falles, in dem ein Algorithmus gegen gegen einen anderen die Oberhand gewinnt, für die Analyse der verschiedenen Leistungen taugt. Sicherlich ist sie ein sehr nützliches Hilfsmittel, das neue Einsichten gewähren kann. Sich schnell darüber klar werden zu können, in welchen Situation ein (ausgeklügelter) Algorithmus wie das Backfilling sich selbst überlistet, ist sicherlich eine gute Möglichkeit, die eigene Intuition weiterzuentwickeln.
Zumal ist diese Methode beinahe ohne zusätzlichen Aufwand nutzbar. Bereits wenige Zeilen Code erlauben Einblicke in das Verhalten des Systems. Die eigentlichen Kosten dafür sind das Wählen einer ereignisorientieren Simulation. Eine prozessorientiere Sichtweise führt dazu, dass das simulierte System näher am tatsächlichen System ist. Effekte, die natürlich durch parallel agierenden Agenten entstehen, wie etwa kritische Wettlaufsituationen, treten ohne zusätzlichen Aufwand auf; ob gewünscht oder unerwünscht. Diese Effekte können durch eine zufällige Ausführungsreihenfolge mit simuliert werden, dies war allerdings nicht nötig, um vergleichbare Ergebnisse zu erzielen.

REPRODUZIERBARKEIT Die größte Hürde dieser Arbeit stellte die fehlende Reproduzierbarkeit da. Die zugrundeliegende Theorie ist interessant, ohne übermäßig kompliziert oder unverständlich zu sein. Das Aufbauen einer äquivalenten Simulation ist ebenfalls aufwendig,

aber nicht besonders anspruchsvoll. Zumal, je nach Ansatz, moderne Simulationswerkzeuge verwendet werden können, die einen großen Teil der Programmierarbeit stemmen. Den Aufbau der Experimente nachzuvollziehen, ist allerdings bestenfalls teilweise möglich, unnötig intransparent.
Des Weiteren waren Arndt et al. ebenso verschiedene Versionen des Backfilling Algorithmus bekannt. Sie untersuchten die Unterschiede nicht, und verwiesen auf [FW98]. Obwohl der darin gefundene Schluss, dass der Unterschied zwischen "konservativem" und "optimistischem" Backfilling gering wahr ist, wäre es ein Leichtes gewesen, dieses Ergebnis zu überprüfen. Der dazu befragte Artikel enthält nur Abbildungen, eine davon mit einer x-Achsen Auflösung von gerade einmal 10%. Diese vagen Ergebnisse nebenbei mit zu verifizieren, hätte einen sehr geringen Aufwand dargestellt. Und nebenbei hätte sogar festgestellt werden, dass in den meisten Szenarien der gegenteilige Effekt zu beobachten ist, und das optimistische Backfilling eine bessere Leistung erzielt.

WEITERE UNTERSUCHUNGEN Sowohl die Reproduktion als auch die Untersuchungen durch Property Based Testing würde von einer formalen Spezifikation des Systems profitieren. Eine Spezifikation, beispielsweise in *TLA+* würde einen exakten Vergleich von Algorithmen auf Basis des kleinsten Gegenbeispiels ermöglichen. Gleichzeitig wäre es dadurch einfach, ein System mit dem selben Verhalten in beliebigen Programmiersprachen und Simulationsumgebungen nach zubauen.
Außerdem können die Experimente 1 bis 9 noch einmal, mit einer praxistauglichen Metrik untersucht werden. Die Untersuchungen der verschiedenen Algorithmen verlaufen bisher recht qualitativ. Die benutzten Parameter der Auftragszusammenstellungen sind unmotiviert, und die Gewichtung der Bedeutung der einzelnen Metriken zueinander ist unklar. Drosselung als Zielfunktion kann in die Menge der untersuchten Metriken aufgenommen werden.
Auch können weitere Kombinationen der Scheduler untersucht werden. Zum Beispiel wäre es interessant, FiFo mit Backfilling nach LPT zu betreiben, um bestehende Lücken bestmöglich auszufüllen.
Die beschriebenen Methoden des Property Based Testing lieferten zwar interessante Ergebnisse, ob diese aber auch über die Intuitionsbildung hinaus hilfreich sind, ist noch unklar. Algorithmen nicht an konkreten Eingaben, sondern an minimalen Beispielen zu untersuchen scheint eine Vorhesagekraft zu besitzen. Diese stützt sich allerdings darauf, dass "komplizierte" Konstellationen in Experimenten seltener auftreten als "einfache" Konstellationen. Dies ist in den untersuchten Experimenten zwar der Fall, jedoch sind Aufträge in der Realität sicherlich selten über mehrere Eigenschaften gleichverteilt. Deshalb dürften in echten Auftragszusammenstellungen vermutlich

ähnliche Konstellationen wiederholt, andere hingegen gar nicht auftreten. Generell bleibt der Wunsch nach einem besseren Modell für Auftragslisten bestehen. Allerdings könnten Methoden der kleinsten Gegenbeispiele auch nützlich sein, wenn ein gutes Modell über den erwartbaren Auftragsmix besteht. Besteht so ein Modell mit entsprechenden Parametern, wäre es hilfreich herauszufinden, wie klein eine Veränderungen dieser Parameter seien kann, um den gewählten Scheduling Algorithmus durch einen anderen ersetzbar zu machen. Hierfür sollten sich allerdings klassische Optimierungsmethoden besser eignen.

LITERATURVERZEICHNIS

[Ada79] Adams, Douglas: *The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy, #1)*. Del Rey, 1979

[AFKT98] Arndt, Olaf ; Freisleben, Bernd ; Kielmann, Thilo ; Thilo, Frank: Scheduling parallel applications in networks of mixed uniprocessor/multiprocessor workstations. In: *Proc. ISCA 11th International Conference on Parallel and Distributed Computing Systems (PDCS-98)* Citeseer, 1998, S. 190–197

[AFKT99] Arndt, Olaf ; Freisleben, Bernd ; Kielmann, Thilo ; Thilo, Frank: Batch Queueing in the Winner Resource Management System. In: *PDPTA* Citeseer, 1999, S. 2523–2529

[AFKT00] Arndt, Olaf ; Freisleben, Bernd ; Kielmann, Thilo ; Thilo, Frank: A comparative study of online scheduling algorithms for networks of workstations. In: *Cluster Computing, The Journal of Networks, Software Tools and Applications*, 2000, S. 95–112

[FW98] Feitelson, Dror G. ; Weil, Ahuva M.: Utilization and predictability in scheduling the IBM SP2 with backfilling. In: *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing* IEEE, 1998, S. 542–546

[HC00] Hughes, John ; Claessen, Koen: Quickcheck: a lightweight tool for random testing of haskell programs. In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP*, 2000, S. 268–279

[Jel] Jelvis, Tikhon ; youtube.com (Hrsg.): *Reasoning under Uncertainty*. https://youtu.be/LHlZNYFraAs?t=1690

[KSCW97] Karger, David ; Stein, Cli ; College, Dartmouth ; Wein, Joel: Scheduling Algorithms. In: *Handbook of Algorithms and Theory of Computation*, 1997

[KZP00] Keleher, Peter J. ; Zotkin, Dmitry ; Perkovic, Dejan: Attacking the bottlenecks of backfilling schedulers. In: *Cluster Computing* 3 (2000), Nr. 4, S. 245–254

[LPC+15] Li, Jack ; Pu, Calton ; Chen, Yuan ; Talwar, Vanish ; Milojicic, Dejan: Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters, 2015, S. 222–234

[Mac] MACLVER, David R. ; HYPOTHESIS.WORKS/ARTICLE (Hrsg.): *Testing Optimizers*. https://hypothesis.works/articles/testing-optimizers-with-hypothesis/

[Mato8] MATLOFF, Norm: Introduction to discrete-event simulation and the simpy language. In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2 (2008), Nr. 2009, S. 1–33

[MD] MACIVER, David ; DONALDSON, Alastair: Test-Case Reduction via Test-Case Generation: Insights From the Hypothesis Reducer. In: *in https://2020. ecoop. org/details/ecoop-2020-papers/13/Test-Case-Reduction-via-Test-Case-Generation-Insights-From-the-Hypothesis-Reducer. URL: https://drmaciver. github. io/papers/reduction-via-generationpreview. pdf*

[TEF07] TSAFRIR, Dan ; ETSION, Yoav ; FEITELSON, Dror G.: Backfilling using system-generated predictions rather than user runtime estimates. In: *IEEE Transactions on Parallel and Distributed Systems* 18 (2007), Nr. 6, S. 789–803

[Way18] WAYNE, Hillel: *Practical TLA+ - Planning Driven Development*. New York : Apress, 2018. – 167–197 S. – ISBN 978–1–484–23829–5