Strategien zur effizienzsteigernden Verteilung von Rechenjobs auf unzuverlässige Knoten im JoSchKa-System

Matthias Bonn, Hartmut Schmeck

Institut AIFB, Universität Karlsruhe (TH), Englerstr. 11, 76131 Karlsruhe
bonn@aifb.uni-karlsruhe.de
schmeck@aifb.uni-karlsruhe.de

Kurzfassung. Dieser Beitrag beschreibt verschiedene Strategien zur Verteilung von rechenzeitintensiven Jobs auf heterogene, unzuverlässige Rechensysteme. Dabei werden verschiedene Verfahren und Strategien untersucht, wie sich durch Rechnerausfälle unnütz verwendete CPU-Zeit verringern lässt. Neben der zu erzielenden Effizienzsteigerung sind dabei auch die faire Behandlung aller beteiligter Nutzer sowie eine möglichst schnelle Abarbeitung der einzelnen Jobs zu beachten.

1 Aufgabenverteilung im Netz: Das JoSchKa-System

Im Zuge der fortschreitenden Verbreitung des Internets seit dem Ende der 1990er Jahre existieren einige Projekte, die die brachliegende Rechenleistung von modernen Personal Computern, die sich im Büro- oder Heimeinsatz überwiegend im Leerlauf befinden, für wissenschaftliche Zwecke nutzen. Neben dem bekannten SETI@home (das inzwischen unter Einsatz der BOINC-Technologie [BOIN05] weitergeführt wird), existieren noch einige weitere dieser „Public-Ressource Computing“-Projekte, zum Beispiel zetaGrid [ZeGr05] oder Folding@Home. Neben diesen meist auf eine bestimmte Aufgabe spezialisierten Projekten existieren die Cluster-Systeme, die flexible Verfahren zur Verteilung von vielfältigen Problemstellungen auf homogenen Rechnernetzen bereitstellen [PBS05, COND05]. Sie eignen sich allerdings nicht notwendigerweise dazu, brachliegende Rechenleistung auf normalen Arbeitsplatz- oder gar Internet-PCs zu nutzen. Der bei JoSchKa [BTS05] beschriebene Ansatz folgt der Idee, dass ein zentraler Server die einzelnen Jobs verwaltet und ein auf dem Rechenknoten installierter Agent selbsttätig beim Server anfragt, ob es etwas zu bearbeiten gibt. Der gesamte Rechenjob wird daraufhin vom Server heruntergeladen, ausgeführt und danach das Ergebnis wieder zum Server übertragen (Abbildung 1).

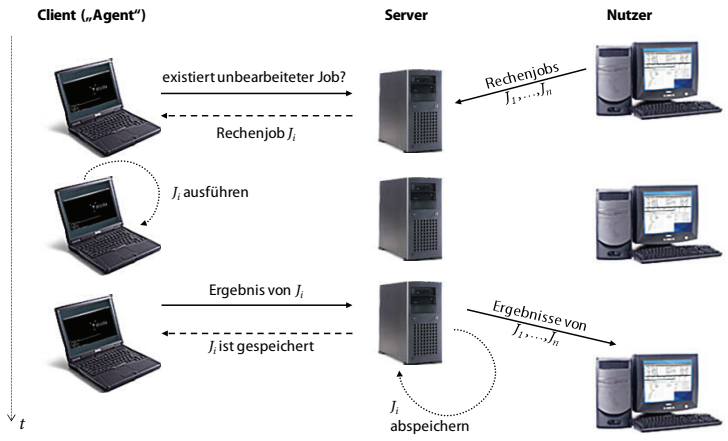

Abbildung 1: Arbeitsprinzip von JoSchKa

Typische Merkmale von JoSchKa sind die Unterstützung autonomer Rechenknoten auf heterogenen Systemen und beliebiger Programmiersprachen, sofern diese auf dem Zielsystem lauffä-

hig sind. Der Datenaustausch erfolgt nicht über ein gemeinsames Dateisystem sondern ausschließlich per HTTP, die Jobs aller beteiligten Entwickler werden dabei fair verteilt.

Das Verfahren hat gewisse Ähnlichkeit mit dem Tuple-Space-Ansatz [GEL85], da die Rechenclients selbsttätig entsprechend dem „pull"-Prinzip Rechenjobs vom Server anfordern. Der Praxiseinsatz hat gezeigt, dass sich die einzelnen Rechenknoten in ihrem Verhalten unterscheiden. Im Wesentlichen betrifft das die Zuverlässigkeit, mit der ein Rechner einen Job abarbeitet: Manche Rechner arbeiten stabil, andere fallen häufig aus. Im Folgenden wird zunächst kurz beschrieben, wie das JoSchKa-System auf diese Unzuverlässigkeit reagiert. Anschließend werden verschiedene Strategien vorgestellt, wie man Jobs mit unterschiedlichen Laufzeitanforderungen auf die heterogenen Knoten so verteilen kann, dass die Unzuverlässigkeit möglichst geringe negative Auswirkungen hat. Die Verteilstrategien werden abschließend per Simulation bewertet und zu einer Gesamtstrategie kombiniert, die auch in der Praxis eingesetzt wird.

2 Fehlertoleranz

Betreibt man einen dedizierten Rechencluster, kann man sich im Wesentlichen darauf verlassen, dass jeder Rechner einen Job auch vollständig bearbeitet, ohne auszufallen. Betreibt man die Knoten jedoch in einer unsicheren Umgebung wie normalen Arbeitsplätzen oder im Poolbetrieb, ist mit Ausfällen oder nicht vollständig durchgeführten Jobs zu rechnen: Die PCs können vom Benutzer einfach ausgeschaltet oder heruntergefahren werden, und die bisher auf diesem Knoten getätigten Berechnungen sind verloren. Aus diesem Grund wurden Mechanismen entwickelt, die diesem Problem entgegenwirken: Der Server verwaltet für jeden gestarteten Job einen Zähler, der periodisch um 1 erniedrigt wird. Der Agent, auf dem der Job läuft, sendet regelmäßig ein Keep-Alive-Signal, das den Server veranlasst, den Zähler wieder auf den maximalen Initialwert zu setzen. Fällt der Clientrechner aus, wird für diesen Job dieses Keep-Alive-Signal ausbleiben und der Zähler beim Server damit nach einer gewissen Zeit auf 0 abgelaufen sein. In diesem Falle wird der Job für einen Neustart bereitgestellt.

3 Effizienzverbessernde Verteilung von Jobs

Beim Entwurf von JoSchKa wurde also davon ausgegangen, dass sich die einzelnen Rechenknoten nicht nur in ihrer Hard- und Softwareausstattung unterscheiden, sondern auch im jeweiligen Zuverlässigkeitsverhalten: Es gibt Knoten, die einen Rechenjob unabhängig von dessen Laufzeit zuverlässig vollständig abarbeiten, ebenso existieren Knoten, die nicht vorhersagbar ausfallen. Weiterhin besteht die Möglichkeit, dass ein bisher sehr zuverlässiger Rechenknoten sein Arbeitsverhalten ändert und unzuverlässig wird, bzw. im umgekehrten Fall ein bisher unsicherer Rechner nun robust arbeitet.

3.1 Monitoring der Rechenknoten

Um die einzelnen Rechner beurteilen zu können, wurde eine Komponente entwickelt, die das Verhalten eines jeden am System bekannten Knotens protokolliert. Ziel ist es, über diese Messwerte Aussagen darüber treffen zu können, wie sich ein Knoten in Zukunft verhalten wird und dieses Wissen bei der Verteilung der einzelnen Jobs auf die Knoten berücksichtigen zu können. Pro Knoten werden dabei die folgenden Werte gespeichert:

- B : Ein synthetischer Benchmarkindex, der die CPU-Leistung eines Rechenknotens repräsentiert, wobei $B \in \{-4, -2, 0, 2, 4\}$ gilt.
- avF : Die durchschnittliche Arbeitszeit (gemessen in Echtzeit-Minuten), die der Knoten für einen nicht erfolgreich bearbeiteten Job bis zum Ausfall benötigt, wobei $avF \in \mathbb{N}$ gilt.

- avS: Die durchschnittliche Arbeitszeit (gemessen in Echtzeit-Minuten), die der Knoten für einen vollständig bearbeiteten Job benötigt, wobei $avS \in \mathbb{N}$ gilt.
- R: Der Zuverlässigkeitsindex des Knotens, wobei $R \in \mathbb{Z}$ und $R \in [-k;k]$ gilt.
- nP: Der normierte Leistungsindex des Knotens, der die Leistungsfähigkeit eines Knotens einer von $nPMAX + 1$ verschiedenen Leistungsklassen zuteilt, wobei $nP \in \mathbb{N}$ und $nP \in [0;nPMAX]$ gilt.

Formal berechnen sich die Werte für einen Knoten $NODE_i, i \in \mathbb{N}$, wie folgt:

- $$B_i = \begin{cases} 4, & rB_i < 5000 \\ 2, & 5000 \leq rB_i < 10000 \\ 0, & 10000 \leq rB_i < 15000 \\ -2, & 15000 \leq rB_i < 20000 \\ -4, & 20000 \leq rB_i \end{cases},$$

  wobei $rB_i \in \mathbb{N}$ die in Millisekunden gemessene Echtzeit darstellt, die der Knoten für einen fest definierten logisch-arithmetischen Benchmark benötigt. Dieser Testlauf wird von jedem Knoten einmalig beim Start durchgeführt.
- $$avF_i = \frac{1}{t_f} \sum_{j=1}^{t_f} f_j ,$$

  wobei die f_j die letzten t_f Bearbeitungszeiten für nicht erfolgreich bearbeitete Jobs darstellen.
- $$avS_i = \frac{1}{t_s} \sum_{j=1}^{t_s} s_j ,$$

  wobei die s_j die letzten t_s Bearbeitungszeiten für erfolgreich bearbeitete Jobs darstellen.
- $$R_i = \frac{1}{2} \sum_{j=1}^{2k} r_j ,$$

  wobei die r_j jeweils ein Erfolgsmaß für die letzten $2k$ bearbeiteten Jobs darstellen, mit $r_j = 1$ für fertig gestellte und $r_j = -1$ für abgebrochene Jobs. R_i wird vom Server mit dem Benchmarkindex des Knotens initialisiert, es gilt zunächst also $R_i = B_i$.
- $$nP_i = \left\lfloor \frac{R_i - \min(R_j)}{\max(R_j) - \min(R_j)} \cdot nPMAX + 0,5 \right\rfloor,$$

  hierbei wird also der Knoten mit dem kleinsten Zuverlässigkeitsindex der Leistungsklasse 0 zugeteilt, derjenige mit dem größten Zuverlässigkeitsindex wird der Klasse $nPMAX$ zugeteilt. Alle anderen werden entsprechend den verbleibenden $nPMAX - 1$ Zwischenstufen zugeteilt.

Für die Praxis wurden die Werte $t_f = t_s = 5$, $k = 5$ und $nPMAX = 20$ gewählt.

Diese Daten, die der Server für jeden einzelnen am System bekannten Rechner sammelt, lassen sich wie folgt zusammenfassen: Man weiß für jeden Knoten,

- wie lange er durchschnittlich arbeitet, bevor er ausfällt,
- wie lange er durchschnittlich mindestens arbeitet, wenn er nicht ausfällt,
- wie zuverlässig er bei den letzten 10 bearbeiteten Jobs war und
- wie zuverlässig er im Vergleich mit den anderen Knoten arbeitet.

Dieses Wissen wird bei der leistungsbasierten und bei der laufzeitbasierten Verteilung der Jobs auf die einzelnen Rechenknoten verwendet. Die Verteilung und die dahinterliegenden Verfahren werden im folgenden Abschnitt beschrieben.

3.2 Verteilung der Jobs

Wie in den letzten beiden Abschnitten deutlich wurde, hat man es nicht mit einem spezialisierten Rechencluster zu tun, sondern mit einem heterogenen Pool aus unterschiedlichsten Rechnern, die unterschiedliche Rechenleistung und vor allem unterschiedliche Zuverlässigkeit besitzen: Ein Rechner kann tagelang ununterbrochen laufen und dann plötzlich unvorhersehbar neu gestartet werden. Ein Job, der eine stundenlange Laufzeit hat, kann auf einem derart unzuverlässigen Rechner nun nicht mehr ausgeführt werden. Gleichzeitig wäre es Verschwendung, wenn ein zuverlässiger Knoten nur Jobs ausführt, die jeweils nach wenigen Minuten beendet sind. Wenn nun das Verteilsystem verschiedene Jobs zur Auswahl hat, liegt es nahe, sie so zu verteilen, dass die durch Ausfälle unnötig verwendete Rechenzeit möglichst klein gehalten wird, die einzelnen Nutzer jedoch so fair wie möglich behandelt werden, d.h. kein Nutzer soll das Gefühl haben, er würde benachteiligt. In diesem Abschnitt werden zunächst die grundlegenden Verteilstrategien von JoSchKa beschrieben, bevor sie dann bewertet und zu einem Gesamt-Verteilalgorithmus zusammengeschaltet werden.

Insgesamt existieren vier verschiedene Verteilverfahren:

(a) Ausbalancierte Verteilung: Hierbei werden die Jobs für alle Nutzer fair an die einzelnen Knoten verteilt, so dass für alle Nutzer die gleiche Anzahl an Rechenknoten arbeitet.

(b) Bevorzugung neuer Nutzer: Bei dieser Strategie werden die Jobs des Benutzers bevorzugt, von dem bisher die wenigsten Jobs berechnet wurden.

(c) Leistungsbasierte Verteilung: Zuverlässige Rechenknoten bearbeiten Jobs, die hohe Anforderungen haben, unzuverlässige Knoten bearbeiten Jobs mit geringen Anforderungen.

(d) Laufzeitbasierte Verteilung: Arbeitet ähnlich der leistungsbasierten Verteilung, jedoch basiert die Zuordnung von Jobs zu Rechnern nicht auf einem normierten Leistungsindex, sondern auf den realen Arbeitszeiten des Rechenknotens und den realen Laufzeitanforderungen des Jobs.

Die Strategien werden im Folgenden jeweils detailliert vorgestellt. Bei den beschriebenen Verfahren unterscheidet der Scheduler nicht zwischen einzelnen Benutzern, sondern zwischen vom Benutzer zu definierenden, einzelnen *JobType*s, was den Vorteil hat, dass man auf möglicherweise unterschiedliche Anforderungen, die der gleiche Nutzer an das System stellt, besser eingehen kann. Weiterhin wird eine vom Benutzer eventuell vorgegebene Priorisierung berücksichtigt: Unabhängig vom gewählten Verfahren werden die Jobs entsprechend ihrer Priorität ausgeliefert. Im Folgenden wird davon ausgegangen, dass im System n Jobs zur Verteilung bereitstehen, die sich auf m verschiedene Job-Typen verteilen, wobei $n, m \in \mathbb{N}$ und $m \leq n$ gilt. Weiterhin bezeichne

- J_i, $i \in [1;n]$ den Job i,
- JT_k, $k \in [1;m]$ den Job-Typ k und
- $jT : [1;n] \rightarrow [1;m]$ mit $jT(i) = k$ die Job-Typ-Nummer des Jobs i.

3.2.1 Ausbalancierte Verteilung

Sei $nrWORKING_k$ die Anzahl der Jobs des Typs JT_k, die zum Zeitpunkt der Anfrage des Rechenknotens bearbeitet werden (wobei $nrWORKING_k \in \mathbb{N}$). Dann wird der Job i ausgeliefert, der $nrWORKING_{jT(i)}$ minimiert. Ein Knoten erhält also grundsätzlich einen Job des Typs, von

dem gerade am wenigsten Jobs bearbeitet werden. Dies hat zur Folge, dass für jeden Type/Benutzer, die gleiche Anzahl an Rechenknoten aktiv ist.

3.2.2 Bevorzugung neuer Nutzer

Sei $relDONE_k$ der Anteil an Jobs des Typs JT_k, die zum Zeitpunkt der Anfrage des Rechenknotens erfolgreich berechnet wurden (wobei $relDONE_k \in \mathbb{R}$ und $relDONE_k \in [0;1]$). Dann wird der Job i ausgeliefert, der $relDONE_{jT(i)}$ minimiert. Dies hat zur Folge, dass Job-Typen/Benutzer deren relativer Anteil an abgeschlossenen Jobs am niedrigsten ist, massiv bevorzugt werden, bis sich dieser Anteil an den der anderen Benutzer angepasst hat.

3.2.3 Leistungsbasierte Verteilung

Bei dieser Verteilstrategie werden die vorhandenen Job-Typen zunächst nach ihren Laufzeitanforderungen klassifiziert. Diese geschieht analog zum normierten Leistungsindex eines Rechenknotens. Dabei sei zunächst avT_k die durchschnittliche Laufzeit eines Jobs vom Typ JT_k, gemessen in Minuten. Diese Laufzeit wird gemäß der folgenden Vorschrift auf einen synthetischen Laufzeitindex $avTI_k$ abgebildet:

$$avTI_k = \begin{cases} -8, & avT_k < 15 \\ -5, & 15 \leq avT_k < 60 \\ -3, & 60 \leq avT_k < 180 \\ 0, & 180 \leq avT_k < 480 \\ 3, & 480 \leq avT_k < 960 \\ 5, & 960 \leq avT_k < 2160 \\ 8, & 2160 \leq avT_k \end{cases}$$

Anhand dieses Laufzeitindex werden die einzelnen Typen in $TIMEMAX + 1$ Klassen eingeteilt. Für die Laufzeitklasse $nTIME_k$ eines Jobs des Typs JT_k gilt: $nTIME_k \in \mathbb{N}$, $nTIME_k \in [0;TIMEMAX]$, und sie berechnet sich wie folgt:

$$nTIME_k = \left\lfloor \frac{avTI_k - \min\left(avTI_j\right)}{\max\left(avTI_j\right) - \min\left(avTI_j\right)} \cdot TIMEMAX + 0,5 \right\rfloor$$

Wie beim Leistungsindex eines Rechenknotens wurde hier ebenfalls $TIMEMAX = 20$ gewählt. Wenn nun ein Knoten $NODE_j$ nach einem Job anfragt, wird derjenige Job J_i ausgeliefert, der $\left| nTIME_{jT(i)} - nP_j \right|$ minimiert. Das Verfahren soll im Folgenden durch ein Beispiel erläutert werden:

Angenommen, dem System wären 5 Rechenknoten $NODE_1,\ldots,NODE_5$ bekannt. Diese hätten die Zuverlässigkeitsindices:

$$R_1 = -4, \quad R_2 = -2, \quad R_3 = 0, \quad R_4 = 3, \quad R_5 = 5$$

Daraus würden sich die folgenden normierten Leistungsklassen ergeben:

$$nP_1 = 0, \quad nP_2 = 4, \quad nP_3 = 9, \quad nP_4 = 16, \quad nP_5 = 20$$

Weiterhin seien im System Jobs dreier verschiedener Job-Typen $JT_1,\ldots,JT_3$ vorhanden, die die folgenden Laufzeitanforderungen (in Minuten) hätten:

$$avT_1 = 5, \quad avT_2 = 40, \quad avT_3 = 120$$

Daraus würden sich dann die normierten Laufzeitklassen

$$nTIME_1 = 0, \quad nTIME_2 = 6, \quad nTIME_3 = 20$$

berechnen. Jobs vom Typ JT_1 würden also von $NODE_1$ berechnet werden, Jobs aus JT_2 würden an $NODE_2$ und an $NODE_3$ verteilt, $NODE_4$ und $NODE_5$ wären für JT_3 zuständig. Abbildung 2 zeigt, wie die einzelnen Knoten und Job-Typen des Beispiels den einzelnen Klassen (in der Tabelle in der mittleren Zeile dargestellt) zugeordnet werden und welche Knoten für welche Typen zuständig sind.

NODE	**1**				**2**					**3**							**4**				**5**
nP *nTIME*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
JT	**1**						**2**														**3**

Abbildung 2: Beispiel zur leistungsbasierten Verteilung

Das Verfahren sorgt also dafür, dass die zuverlässigen Knoten immer diejenigen Jobs bearbeiten, die die längste erwartete Laufzeit haben. Analog dazu bearbeiten die unzuverlässigen Knoten immer kurzlaufende Jobs. Sollte ein Knoten einer Klasse zugeteilt werden, die nicht eindeutig einer Job-Typen-Klasse zugeordnet werden kann (im Beispiel die Klassen 3 und 13), erfolgt die Zuteilung zufällig auf eine der beiden nächstliegenden Job-Typen-Klassen.

3.2.4 Laufzeitbasierte Verteilung

Dieses Verfahren berücksichtigt bei der Zuordnung eines Jobs zu einem Rechenknoten nicht nur dessen Zuverlässigkeit, sondern primär die durchschnittliche Zeit, die der Knoten mit dem Bearbeiten von Jobs beschäftigt war. Dabei wird unterschieden, ob der Rechner den Job erfolgreich abgeschlossen hat oder ob der Knoten ausgefallen ist, bevor der Job beendet war. Es werden also die in Abschnitt 3.1 beschriebenen Werte avF_i und avS_i eines Rechenknotens verwendet. Wenn nun ein Knoten $NODE_i$ eine Anfrage nach einem Job stellt, berechnet das Verfahren aus seinen Leistungsdaten zunächst einen Durchschnitts-Zielwert $avTARGET$ wie folgt:

$$avTARGET = \begin{cases} \alpha \cdot avF_i, & R_i \in \{-5\} \\ \beta \cdot avF_i, & R_i \in \{-4,-3\} \\ \gamma \cdot avF_i, & R_i \in \{-2,-1\} \\ \delta \cdot avS_i, & R_i \in \{0,1\} \\ \varepsilon \cdot avS_i, & R_i \in \{2,3\} \\ \zeta \cdot avS_i, & R_i \in \{4,5\} \end{cases} \quad \text{mit} \quad \begin{array}{l} \alpha = 2^{-0,5s} \\ \beta = 2^{-0,25s} \\ \gamma = 1,0 \\ \delta = 1,0 \\ \varepsilon = 1,0 + 0,5s \\ \zeta = 1,0 + s \end{array} \quad \text{und } s \geq 0$$

Der Durchschnitts-Zielwert wird also aus den durchschnittlichen Arbeitszeiten des Knotens berechnet. In Abhängigkeit davon, ob der Rechner eher zuverlässig oder eher unzuverlässig ist, wird die durchschnittliche Erfolgszeit oder die durchschnittliche Fehlerzeit verwendet. Mit dem Parameter s kann der Durchschnitts-Zielwert noch nach oben oder unten gespreizt werden, je nach dem Grad der (Un-)Zuverlässigkeit des Knotens.

Weiterhin verwendet das Verfahren die durchschnittlich zu erwartende Laufzeit avT_k der Job-Typen. Im Folgenden gelte $avT_i \leq avT_j$ (für $i < j$), sie seien also entsprechend ihrer Laufzeit sortiert. Für je zwei benachbarte Typen wird der Mittelwert der Laufzeiten bestimmt:

$$avTM_k = \frac{avT_k + avT_{k+1}}{2}$$

Schließlich wird der eigentliche Laufzeit-Zielwert $RLTV$ berechnet:

$$RLTV = avTM* + \mathrm{rnd}(-2,2), \text{ wobei}$$

$$\left| avTARGET - avTM* \right| = \min_k \left| avTARGET - avTM_k \right|$$

Es wird also derjenige Mittelwert bestimmt, der dem zuverlässigkeitsabhängig gespreizten Durchschnitts-Zielwert des anfragenden Knotens am nächsten liegt. Da das System hier nur mit Ganzzahlen rechnet, wird $RLTV$, um Rundungseffekte auszuschließen, noch um einen Wert von maximal 2 nach oben oder unten randomisiert. An den Knoten ausgeliefert wird letztlich der Job J_i, der $\left| RLTV - avT_{jT(i)} \right|$ minimiert. Da das Verfahren im Falle, dass nur 2 verschiedene Job Typen vorhanden sind, die Knoten nur zufällig zwischen diesen beiden verteilen würde, wird bei diesem Verfahren (zusätzlich zu den real vorhandenen) ein Pseudojobtyp JT_0 mit $avT_0 = 0$ verwendet. Das Verfahren soll nun an einem Beispiel verdeutlicht werden:

Angenommen, der Verteil-Algorithmus sei mit dem Parameter $s = 2$ konfiguriert und in der Datenbasis seien Jobs der Typen $JT_1,\ldots,JT_4$ zu Bearbeitung verfügbar und diese hätten die folgenden durchschnittlichen Bearbeitungszeiten (in Minuten):

$$avT_1 = 20, \quad avT_2 = 50, \quad avT_3 = 150, \quad avT_4 = 160$$

Weiterhin sei angenommen, es würde nun ein Knoten am System nach einem Job anfragen und für diesen Knoten wären bisher die Leistungsdaten

$$avF = 60, \quad avS = 90, \quad R = 3$$

erfasst worden. Dann würde sich ein $avTARGET = avS \cdot (1,0 + 0,5s) = 180$ ergeben. Der diesem Wert am nächsten liegende Mittelwert ist $avTM_3 = \frac{1}{2}(avT_3 + avT_4) = 155$. Dem Knoten würde also wegen $RLTV = 155 + \mathrm{rnd}(-2,2)$ ein Job der Typen JT_3 oder JT_4 (zufällig ausgewählt) zugeteilt. Das Beispiel wird in Abbildung 3 graphisch veranschaulicht.

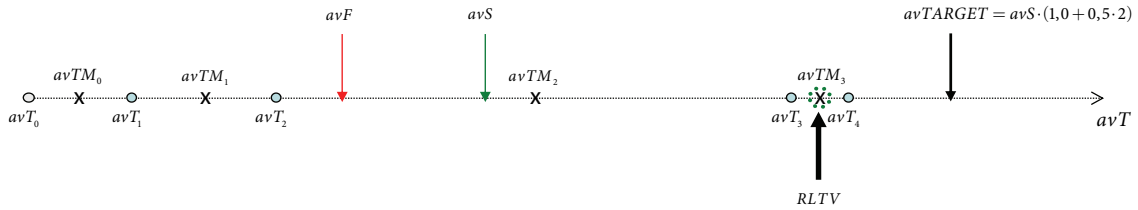

Abbildung 3: Beispiel zur laufzeitbasierten Verteilung

Hier wird auch deutlich, welche Funktion die Spreizung mit dem Parameter s innehat: Ein Knoten, der sehr zuverlässig arbeitet erhält dadurch die Chance, einen Job zu bearbeiten, der mehr Zeit beansprucht, als der Knoten bisher durchschnittlich gearbeitet hat.

4 Simulation und Bewertung der Strategien

Um die einzelnen Strategien zu testen, zu bewerten und zu verbessern wurde ein Simulator entwickelt, der es erlaubt zu simulieren, wie sich das System bei bestimmten Job-Typ und Client-Situationen verhalten würde. Für jeden zu simulierenden Rechenknoten lassen sich die folgenden Parameter konfigurieren: Den Benchmarkwert rB (in Millisekunden, siehe Abschnitt 3.1), die der Client zu Beginn misst und zum Server überträgt, zwei Ausfallwahrscheinlichkeiten (in %), die festlegen, mit welcher Wahrscheinlichkeit ein Knoten pro Simulationsminute ausfällt und die Anzahl der Clients, die auf diese Weise parametrisiert sind. Durch die Angabe von zwei verschiedenen Ausfallwahrscheinlichkeiten kann simuliert werden, wie sich das System verhält, wenn ein Knoten plötzlich seine Zuverlässigkeit ändert: Nach 1000 simulierten Minuten wird

von der ersten auf die zweite Ausfallwahrscheinlichkeit umgeschaltet. Jeder simulierte Job-Typ lässt sich über die Anzahl der Jobs dieses Typs und dessen Namen, die Laufzeit (in Simulationsminuten), die ein Job dieses Typs benötigt und die Anzahl der Minuten parametrisieren, die nach Hinzufügen der Jobs dieses Typs simuliert werden sollen.

Die einzelnen Strategien wurden mit zwei verschiedenen Simulationskonfigurationen getestet. Bei beiden Situationen wurden 3000 Minuten simuliert, wobei nach 1000 Minuten die Ausfallwahrscheinlichkeit der Knoten getauscht wurde: Zuverlässige wurden unzuverlässig und umgekehrt. Bei beiden Simulationen waren lang laufende Jobs zu je 120 Minuten definiert. Bei Simulation A kamen noch kurzlaufende Job zu je 5 Minuten und mittellang laufende Jobs mit 35 Minuten hinzu, bei Simulation B wurden Jobs mit 10 Minuten und weitere lang dauernde mit 130 Minuten Laufzeit simuliert.

4.1 Bewertung der Verteilstrategien

Um die Leistung der Verteilstrategien quantifizieren zu können, wurden zwei Bewertungskriterien festgelegt:

(a) Die *durchschnittliche Effizienz* $avEff$ (angegeben in %) einer Strategie berechnet sich aus dem Verhältnis von erfolgreich „gearbeiteten“ Simulationsminuten aller Knoten zur Gesamtzahl der von allen Knoten „gearbeiteten“ Zeit, gemessen über den gesamten Simulationszeitraum. Ein hoher Wert bedeutet hier, dass die einzelnen Knoten nicht sinnlos gearbeitet haben (was z.B. bei häufigen Abstürzen der Fall wäre).

(b) Die *gesamtdurchschnittliche Fertigungsrate* $avDONE$ (ebenfalls in % angegeben) berechnet sich aus dem Mittelwert der durchschnittlichen Fertigungsraten der einzelnen Job-Typen, gemessen über den gesamten Simulationszeitraum. Ein hoher Wert würde hier bedeuten, dass die einzelnen Jobs schnell erledigt werden.

Im Folgenden werden die 4 Verteilverfahren mit den beiden Simulationen getestet und bewertet. Beim laufzeitbasierten Verfahren wird zusätzlich noch auf die Rolle des Parameters s eingegangen. Die wesentlichen Fragestellungen bei der Beurteilung der einzelnen Verfahren lauten:

- Sind die Werte $avEff$ und $avDONE$ möglichst hoch?
- Sind die Job-Typen gut ausbalanciert?
- Wie verhält sich die Verteilstrategie, wenn die Rechenknoten nach der 1000. Simulationsminute ihre Zuverlässigkeit ändern?

4.1.1 Ausbalancierte Verteilung

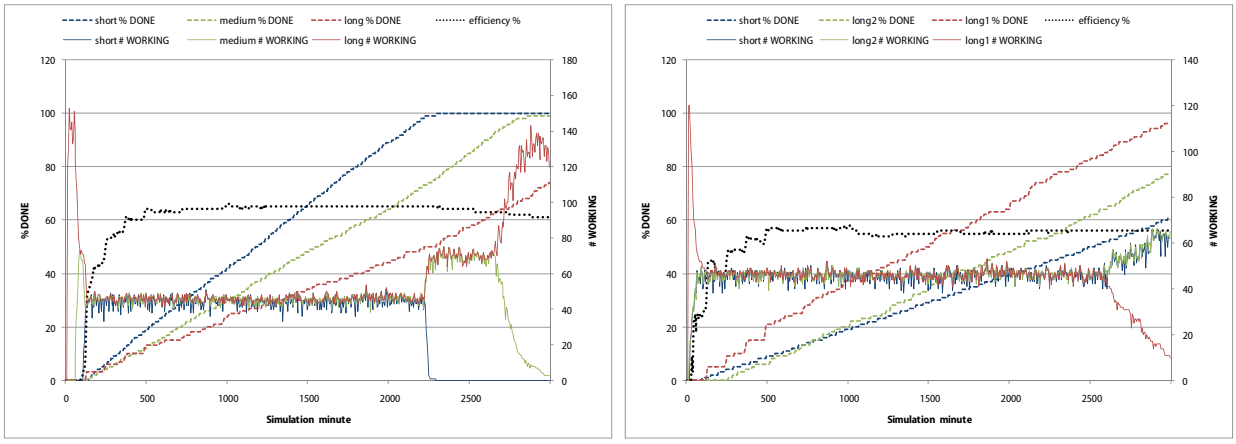

Abbildung 4: Simulation A / Simulation B bei ausbalancierter Verteilung

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	60%	48%	sehr gut ausbalanciert / keine Reaktion auf Zuverlässigkeitsänderung, sehr stabil
B	53%	38%	sehr gut ausbalanciert / keine Reaktion auf Zuverlässigkeitsänderung, sehr stabil

Charakteristisches Merkmal des ausbalancierten Verfahrens ist, dass für alle Job-Typen immer die gleiche Anzahl an Knoten tätig ist. Da die Zuverlässigkeit der Knoten nicht berücksichtigt wird, hat die Änderung der Ausfallwahrscheinlichkeit keinen Einfluss auf die Verteilung. Bei diesem Verfahren kann und wird es passieren, dass ein sehr zuverlässiger Rechenknoten kurz laufende Jobs bearbeitet, während unzuverlässige Knoten lange laufende Jobs berechnen müssen (die sie dann mit hoher Wahrscheinlichkeit nicht zu Ende bringen).

4.1.2 Bevorzugung neuer Nutzer

Diese Strategie bevorzugt immer denjenigen Job-Typ, der die niedrigste Fertigungsrate besitzt. Dies hat zur Folge, dass die Anzahl der für einen Typ arbeitenden Knoten solange maximal ist, bis die Fertigungsrate die Rate der anderen Benutzer erreicht oder überholt hat, danach fällt sie auf 0 zurück. Da die Zuverlässigkeit der Knoten nicht berücksichtig wird, hat die Änderung der Ausfallwahrscheinlichkeit keinen Einfluss auf die Verteilung. Gegenüber der ausbalancierten Verteilung sinken vor allem bei Simulation A (siehe Abbildung 5) die Effizienz und der Fertigungsratendurchschnitt stark ab.

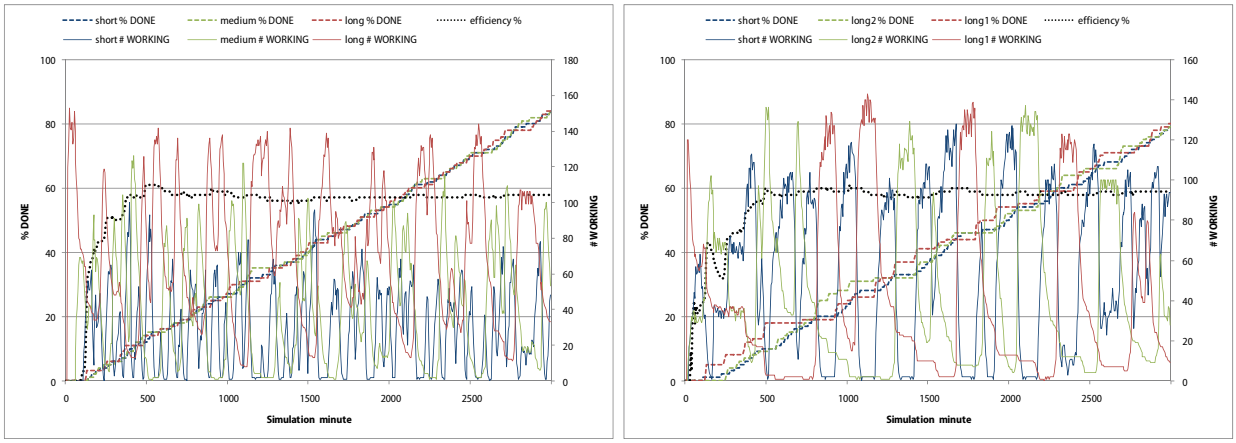

Abbildung 5: Simulation A / Simulation B bei Bevorzugung neuer Nutzer

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	54%	41%	Balance schwankt extrem / keine Reaktion auf Zuverlässigkeitsänderung, sehr stabil
B	55%	38%	Balance schwankt extrem / keine Reaktion auf Zuverlässigkeitsänderung, sehr stabil

4.1.3 Leistungsbasierte Verteilung

Auffällig ist bei diesem Verfahren zunächst, dass vor allem bei Simulation A die erzielte Effizienz und die Fertigungsraten nicht an die Werte der strikt ausbalancierten Verteilung heranreichen. Da dieses Verfahren die Jobs aufgrund der Zuverlässigkeit der Knoten verteilt, können nicht alle Typen gleich fair behandelt werden. Offensichtlich passen die Zuverlässigkeitswerte der meisten Knoten bei beiden Simulationen (Abbildung 6) zunächst am besten zu den Anforderungen der mittellang laufenden Jobs.

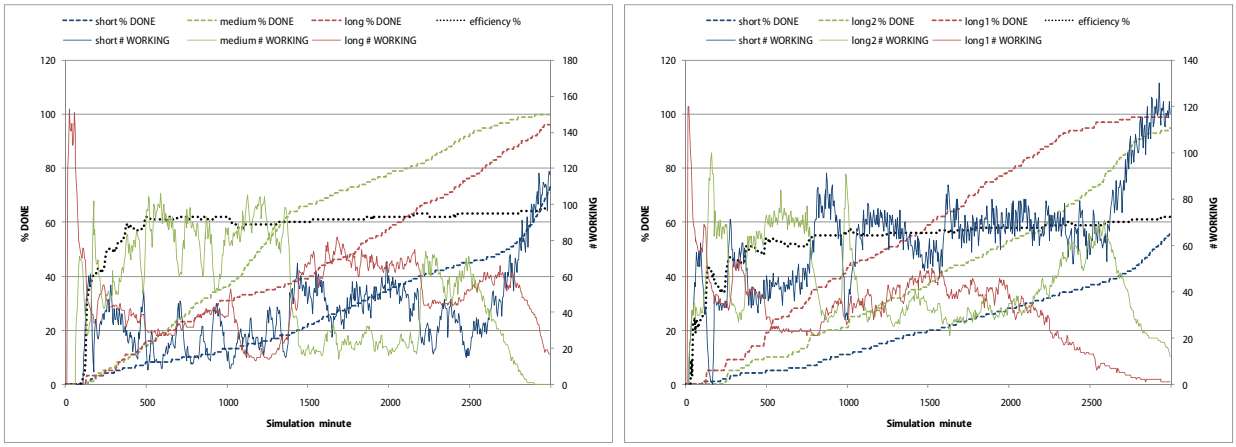

Abbildung 6: Simulation A / Simulation B bei leistungsbasierter Verteilung

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	58%	42%	gut ausbalanciert, mit Schwankungen / Berücksichtigt Zuverlässigkeitsänderung, stabiles Gleichgewicht wird wieder hergestellt
B	54%	39%	gut ausbalanciert, mit Schwankungen / Berücksichtigt Zuverlässigkeitsänderung, stabiles Gleichgewicht wird wieder hergestellt

Weiterhin fällt auf, dass die Anzahl der für einen Typ arbeitenden Knoten wechselt, was daran liegt, dass ein Knoten seinen Zuverlässigkeitsindex durch Bearbeitung kurz laufender Jobs steigert und danach länger laufende Jobs zugeteilt bekommt. Kann er diese nicht erfolgreich abarbeiten, sinkt der Index und als Folge davon erhält er wieder kürzer laufende Jobs. Dieses Hin- und Herwandern der Knoten zwischen den einzelnen Leistungsklassen kann man gut an den *#WORKING*-Kurven der beiden Diagramme ablesen. Da das Verfahren ausschließlich die Zuverlässigkeit der Rechner zum Verteilen berücksichtigt, reagiert es gut auf die Zuverlässigkeitsänderungen: Das vor der Änderung herrschende relativ gut ausbalancierte Gleichgewicht stellt sich nach einiger Zeit wieder ein.

4.1.4 Laufzeitbasierte Verteilung

Diese Verteilung teilt die Jobs den Knoten aufgrund deren Arbeitszeitverhalten zu: Ein Knoten, der eine bestimmte Zeit zuverlässig arbeitet, wird Jobs bearbeiten, die passende Laufzeitanforderungen haben (Abbildung 7).

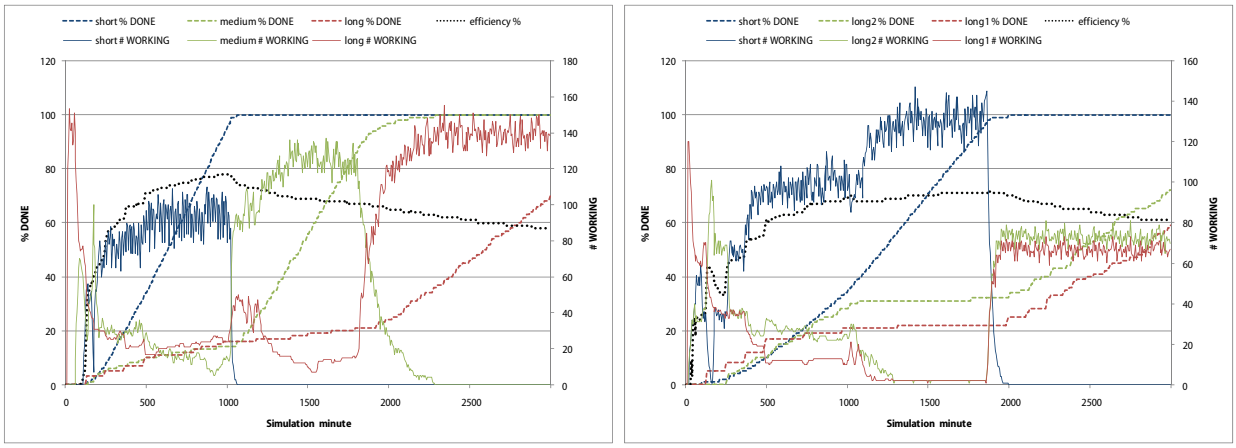

Abbildung 7: Simulation A / Simulation B bei laufzeitbasierter Verteilung mit $s = 0$

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	63%	52%	Balance entsprechend dem Laufzeitverhalten / instabil, vorhergehendes Gleichgewicht stellt sich nicht wieder ein
B	62%	39%	Balance entsprechend dem Laufzeitverhalten / sehr instabil, vorhergehendes Gleichgewicht stellt sich nicht wieder ein, ignoriert langlaufende Job-Typen

Der wächst aber eben nur, wenn lange Jobs erfolgreich bearbeitet werden. Damit wird das Verfahren instabil: Positive Zuverlässigkeitsänderungen werden nicht berücksichtigt. Den Effekt sieht man vor allem im rechten Teil der Abbildung 7: Nach Simulationsschritt 1000 werden die beiden lange laufenden Job-Typen nicht mehr bearbeitet. Das Verfahren muss also so geändert werden, dass zuverlässige Clients die Chance erhalten, Jobs zu berechnen, die mehr Zeit benötigen als der Knoten bisher im Schnitt erfolgreich gearbeitet hat. Diese Spreizung, die durch den im vorigen Abschnitt vorgestellten Parameter s gesteuert wird, soll es einem Knoten erleichtern, zwischen den verschiedenen Job-Typen hin- und herzuwandern, wenn sich seine Leistungsfähigkeit ändert. Das laufzeitbasierte Verfahren wurde auf beiden Simulationen mit mehreren Einstellungen des Spreizungsparameters getestet, hier die Variante mit $s = 2$:

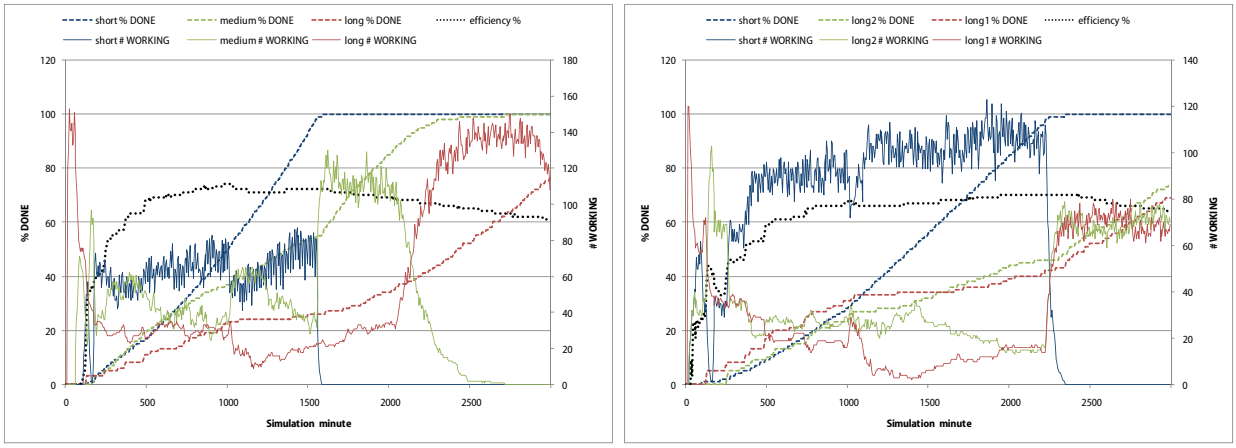

Abbildung 8: Simulation A / Simulation B bei laufzeitbasierter Verteilung und $s = 2$

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	64%	52%	Balance entsprechend dem Laufzeitverhalten / vorhergehendes Gleichgewicht stellt sich nach einiger Zeit wieder ein
B	62%	41%	Balance entsprechend dem Laufzeitverhalten / instabil, vorhergehendes Gleichgewicht stellt sich nicht wieder ein, ignoriert langlaufende Job-Typen, jedoch verbessert gegenüber $s = 0$

Die jetzt erzielten Fertigungsraten ähneln denen mit $s = 0$ und die Effizienzwerte sind nicht mehr ganz so hoch. Das Verfahren verhält sich nun aber bei beiden Simulationen stabiler. Leider wird bei der B-Simulation der erste lang laufende Typ nach dem Umschalten der Ausfallwahrscheinlichkeiten immer noch benachteiligt, obwohl seine Anforderungen denen des zweiten lang laufenden Typs sehr ähneln. Versuche mit größeren Spreizwerten wurden nicht mehr durchgeführt, anstatt dessen wurden Überlegungen angestellt, wie sich die vier vorgestellten Verfahren zu einem Gesamt-Algorithmus kombinieren lassen, der hohe Effizienz und Fertigungsraten erzielt, stabil gegenüber Zuverlässigkeitsänderung ist und die einzelnen Nutzer – entsprechend ihren Laufzeitanforderungen – trotzdem fair behandelt: Nutzer mit gleichen Anforderungen sollten die gleiche Anzahl an Rechenknoten bzw. die gleiche Rechenleistung erhalten.

4.2 Kombinierte Verteilung

Das kombinierte Verfahren soll die Vorteile der vier einzelnen Verfahren vereinen, die jeweiligen Nachteile sollen kompensiert werden. Ideal wäre ein Verfahren, das die hohen Effizienzwerte und Fertigungsraten des laufzeitbasierten Verfahrens erreicht, dabei auf Zuverlässigkeitsänderungen so robust und adaptiv wie das leistungsbasierte Verfahren reagiert und gleichzeitig die Verteilung der Knoten auf die einzelnen Job-Typen so ausbalanciert gestaltet, dass jeder Nutzer ausreichend fair behandelt wird. Dieses bezüglich mehrerer Kriterien günstige Verhalten sollte bei allen möglichen Kombinationen aus Rechenknoten (und deren Leistung bzw. Zuverlässigkeit) und Job-Anforderungen erreicht werden.

Das resultierende Verfahren wird von vier Parametern gesteuert:

- $fairLevel, \in \mathbb{R}, \in [0;1]$: Der Parameter steuert den Grad der fairen/ausbalancierten Verteilung der Knoten auf die Nutzer, der mindestens erfüllt sein muss.
- $doneRateLowBoost, \in \mathbb{R}, \in [0;1]$: Dieser Wert legt fest, bis zu welcher Grenze Job-Typen bevorzugt werden, die einen niedrigen Fertigungsgrad haben.
- $powerIndexProb, \in \mathbb{R}, \in [0;1]$: Die Wahrscheinlichkeit, mit der das leistungsbasierte Verteilverfahren anstelle des laufzeitbasierten Verfahrens eingesetzt wird.
- $runlengthScale, \in \mathbb{R}_0^+$: Der aus dem vorangegangenen Abschnitt bekannte Parameter s des laufzeitbasierten Verfahrens.

Fragt ein Rechenknoten $NODE_i$ beim Server nach einem Job, läuft der folgende Algorithmus ab:

(1) Zunächst bestimmt der Server alle Job-Kandidaten, die für eine Zuteilung zu diesem Knoten in Frage kommen. Gleichzeitig werden für jeden am System bekannten Job-Typ JT_k die Werte $nrWORKING_k$, $relDONE_k$, und $avTI_k$ bestimmt.

(2) Die Zuverlässigkeitswerte R_i aller bekannten Knoten werden sortiert, aus den sortierten Werten wird $majorityInterval = Q_{0,9} - Q_{0,1}$ (also die Breite des Intervalls, in dem die mittleren 80% der Zuverlässigkeitswerte liegen) bestimmt. Weiterhin wird der Wert $avTIdiff = \max_k(avTI_k) - \min_k(avTI_k)$ berechnet.

(3) Mit diesen Werten wird in manchen Fällen der (administrativ vorgegebene) Grad der Ausbalancierung erhöht:

- $(avTIdiff < 4 \vee majorityInterval < 4) \wedge fairLevel < 0,5 \Rightarrow fairLevel = 0,5$
- $(avTIdiff = 0 \vee majorityInterval < 2) \wedge fairLevel < 0,9 \Rightarrow fairLevel = 0,9$

Sind die Anforderungen der einzelnen Jobs zu ähnlich, oder verhalten sich die am System bekannten Rechenknoten überwiegend gleich zuverlässig, braucht man die aufwändigen Leistungs- und Laufzeitverfahren nicht anzuwenden. Es ist in diesem Fall besser, die Jobs fair auf die einzelnen Knoten zu verteilen.

(4) Falls $\min_k(nrWORKING_k) / \max_k(nrWORKING_k) < fairLevel$: Job nach ausbalanciertem Verfahren ausliefern und Algorithmus beenden.

(5) Falls $\min_k(relDONE_k) < doneRateLowBoost$: Job nach dem Verfahren ausliefern, das neue Benutzer bevorzugt und Algorithmus beenden.

(6) Mit der Wahrscheinlichkeit $powerIndexProb$ einen Job nach dem leistungsbasierten Verteilverfahren ausliefern, und Algorithmus beenden, ansonsten einen Job nach laufzeitbasiertem Verfahren (mit $s = runlengthScale$) ausliefern und Algorithmus beenden.

Durch das Setzen des Parameters $fairLevel = 1$ erzeugt man die rein ausbalancierte Verteilung. Setzt man $fairLevel = 0$ und $doneRateLowBoost = 1$ werden die Jobs ausschließlich mit Bevorzugung neuer Nutzer verteilt. Eine rein leistungsbasierte Verteilung erreicht man mit $fairLevel = 0$, $doneRateLowBoost = 0$ und $powerIndexProb = 1$. Eine rein laufzeitbasierte Verteilung wird mit dem Setzen aller Parameter auf 0 erzielt. Die beiden Diagramme in Abbildung 9 zeigen die beiden Simulationen für die folgende Parametereinstellung:

$$fairLevel = 0 \qquad powerIndexProb = 0{,}1$$
$$doneRateLowBoost = 0{,}03 \qquad runlengthScale = 2$$

Die 10%ige Einstreuung des leistungsbasierten Verfahrens hat den Effekt, dass das System die hohen Effizienz- und Fertigungsraten des laufzeitbasierten Verfahren halten kann, jedoch wesentlich stabiler und adaptiver auf Zuverlässigkeitsänderungen reagiert. Dass bei der Simulation B deutlich mehr Rechner für die Kurzjobs arbeiten, liegt daran, dass die Knoten in der Mehrheit eben nicht zuverlässig genug sind für die langlaufenden. Entscheidend ist jedoch, dass sich die Clients, die für die lange laufenden Jobs zuständig sind, gleichmäßig auf die beiden Langläufer verteilen und dass dieses Verhältnis auch nach dem Umschalten der Zuverlässigkeit erhalten bleibt. Der kurze Boost zu Beginn, der jeden Job-Typ massiv bevorzugt bis 3% Fertigungsrate erreicht sind, unterstützt die beiden leistungs-/laufzeitbasierten Verfahren, da diese nur richtig arbeiten können, wenn man für jeden Knoten und jeden Job-Typ dessen Anforderungen kennt. Dies erreicht man, indem man Jobs, von denen nur wenige fertig bearbeitet sind, zunächst bevorzugt behandelt.

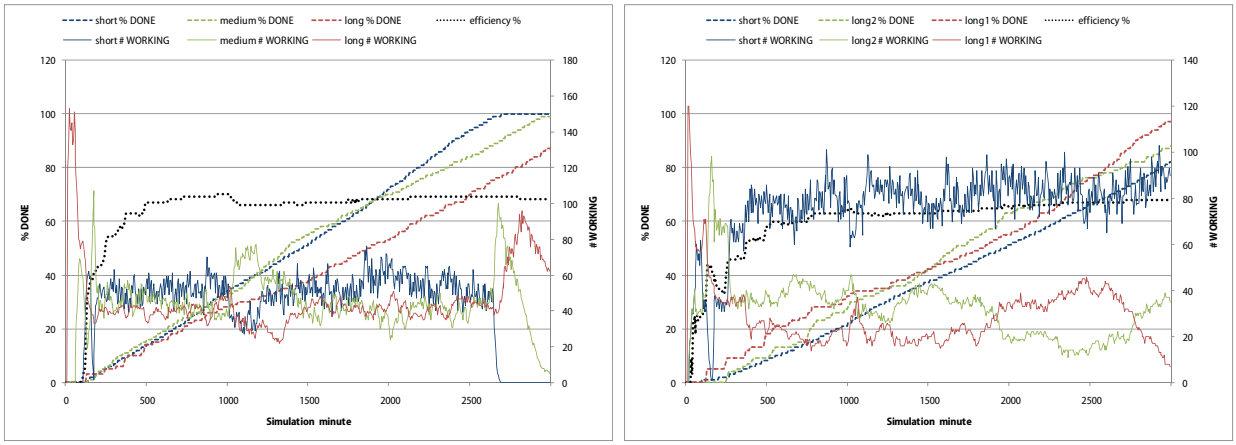

Abbildung 9: Simulation A / Simulation B bei kombinierter Verteilung

Simulation	*avEff*	*avDONE*	Balance / Verhalten bei Zuverlässigkeitsänderung
A	63%	48%	Balance entsprechend dem Laufzeitverhalten, gut ausbalanciert / stabil, vorhergehendes Gleichgewicht stellt sich schnell wieder ein
B	60%	42%	Balance entsprechend dem Laufzeitverhalten / stabil, vorhergehendes Gleichgewicht stellt sich schnell wieder ein

Das ausbalancierende Verfahren wird nur dann angewendet, wenn die beiden effizienzoptimierenden Verfahren aufgrund einer zu homogenen Knotenleistung oder zu ähnlichen Job-Anforderungen nicht sinnvoll einsetzbar wären.

5 Zusammenfassung

Die Simulation der verschiedenen Verteilverfahren hat gezeigt, dass auch unter unsicheren Umgebungen ein gewisses Optimierungspotential besteht und dass die Bearbeitungseffizienz trotz unvorhersehbarer Ausfälle durch Einsatz geeigneter Heuristiken gesteigert werden kann. Im Weiteren hat sich gezeigt, dass nicht alle Verfahren fähig sind, eine geänderte Ausfallwahrscheinlichkeit zu erkennen und entsprechend zu reagieren. Die verschiedenen Verfahren konnten so zu einem Gesamt-Algorithmus kombiniert werden, dass die jeweiligen Vorteile erhalten bleiben, die Nachteile jedoch kompensiert werden. Die im Abschnitt 4.2 vorgestellte kombinierte Strategie wird zur Zeit auf einem Pool aus rund 100 Standard-PCs der Fakultät für Wirtschaftwissenschaften an der Universität Karlsruhe eingesetzt. Über einen Zeitraum von 2 Jahren wurden dort ca. 180000 Jobs aus dem Bereich der naturanalogen Optimierungsverfahren bearbeitet. Diese hatten eine Laufzeit von wenigen Sekunden bis zu mehreren Tagen pro Job.

Literaturverzeichnis

[BOIN05] Berkeley Open Infrastructure for Network Computing, http://boinc.berkeley.edu/

[BTS05] Matthias Bonn, Frederic Toussaint und Hartmut Schmeck. JOSCHKA: Job-Scheduling in heterogenen Systemen. In Erik Maehle, PARS Mitteilungen 2005, pp. 99-106. 20. PARS Workshop, Gesellschaft für Informatik, June 2005

[COND05] Condor High Throughput Computing, http://www.cs.wisc.edu/condor/

[GEL85] David Gelernter: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1, January 1985, Pages 80-112.

[PBS05] Portable Batch System PBSpro, http://www.pbspro.com/

[ZeGr05] zetaGrid, http://www.zetagrid.net/