Planungsverfahren in heterogenen Umgebungen mit Hilfe eines Genetischen Algorithmus

Silke Schuch, Rodolfo Bamberg, Thomas Bemmerl
Lehrstuhl für Betriebssysteme
RWTH Aachen
Kopernikusstrasse 16, 52056 Aachen, Germany
{schuch, bamberg, bemmerl}@lfbs.rwth-aachen.de

Abstract

Es existieren viele verschiedene Hardwarearchitekturen zur Ausführung paralleler Anwendungen. Eine MPI-Implementierung, die den Fokus auf die Unterstützung vieler verschiedener Plattformen und Interconnects legt, ist das am Lehrstuhl für Betriebssysteme der RWTH entwickelte MP-MPICH. Um die Ausführung einer MPI-Applikation auf einem heterogenen System zu starten, wurde am Lehrstuhl für Betriebssysteme das Programm MP-Cluma entwickelt. MP-Cluma unterstützt den Prozessstart einer MPI-Applikation auf allen von MP-MPICH unterstützten Plattformen.

In diesem Artikel werden die Anforderungen an ein Startprogramm für eine MPI-Applikation beschrieben, welche in einer solchen heterogenen Umgebung entstehen. Des Weiteren werden Ansätze vorgestellt, wie Scheduling-Systeme solche Umgebungen unterstützen. Im Anschluss erfolgt die Vorstellung des in MP-Cluma verwendeten Scheduling-Ansatzes, welcher auf einem Genetischen Algorithmus aufbaut.

1. Einleitung

Als Computecluster wird eine Menge von gekoppelten Rechnern bezeichnet, die gemeinsam an einer Aufgabe arbeiten. Diese Rechner können auch aus handelsüblichen Personalcomputern bestehen. Der Vorreiter für Cluster aus handelsüblichen PCs ist der von D. Becker und T. Sterling 1995 vorgestellte Beowulf-Cluster [11].

In der Literatur wird oft unterschieden zwischen *Cluster of Workstations* (COW) und *Network of Workstations* (NOW). Als COW wird eine Gruppe von meist einheitlichen Computern bezeichnet, die räumlich nah zueinander aufgestellt sind und meist über ein schnelles Interconnect verfügen. Ihre Hauptaufgabe es ist, als Clusterknoten eingesetzt zu werden.

Als (NOW) werden Arbeitsstationen im lokalen Netzwerk bezeichnet, die zusätzlich zur Berechnung paralleler Applikationen benutzt werden. Wenn zusätzlich zu dedizierten Clustern (COW) auch reguläre Arbeitsstationen (NOW) als Rechenknoten in einer parallelen Umgebung eingesetzt werden, dann wird eine solche Umgebung in diesem Artikel als *Office Grid* bezeichnet.

1.1 Office Grid

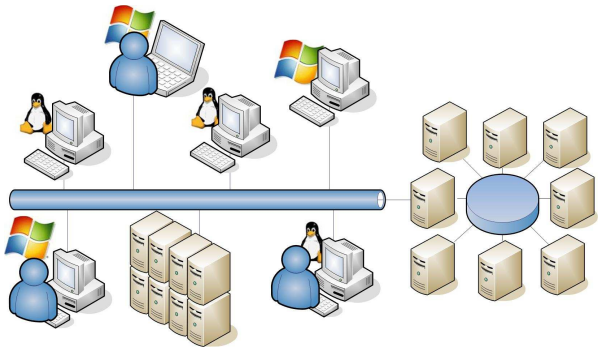

Abbildung 1. Beispiel eines Office Grid

Die Besonderheit des *Office Grid* ist eine erhöhte Heterogenität. Es werden nicht nur verschiedene Plattformen verwendet, sondern auch verschiedene Interconnects.

Das Projekt MP-MPICH [1] bietet die Möglichkeit, MPI-Applikationen auf einem *Office Grid* auszuführen. MP-Cluma bietet dem Benutzer eine komfortable Möglichkeit, alle zu einer MPI-Applikation gehörenden Prozesse zu starten. Wie in diesem Artikel gezeigt wird, ist der Start einer MPI-Applikation nicht trivial. Es müssen die Besonderheiten der zu der Applikation gehörenden Prozesse ebenso beachtet werden, wie verwendete Plattformen und die Struktur des *Office Grid*. Um dem Benutzer größtmöglichen

Komfort zu bieten, wird der Prozessstart optional von einem Scheduler übernommen, der mit einem genetischen Algorithmus arbeitet.

Dieser Artikel ist wie folgt strukturiert: In Kapitel 2 gehen wir kurz auf die Besonderheiten von Prozessen ein, die zu einer MPI-Applikation gehören. Danach wird in Kapitel 3 kurz die Struktur des Clustermanagement-Tools MP-Cluma [10] erklärt. Die Struktur des für MP-Cluma entwickelten Schedulers, zusammen mit Testresultaten und Möglichkeiten der Optimierung, wird in Kapitel 4 präsentiert. Eine Zusammenfassung und ein Ausblick auf künftige weitere Entwicklungen wird in Kapitel 5 gegeben.

2. MPI und MP-MPICH

Das Massage Passing Interface MPI [3] stellt einen Standard zur Implementierung von Kommunikation über Nachrichtenaustausch dar. Eine MPI-Applikation besteht dabei aus mehreren miteinander kommunizierenden Prozessen. Die Identifizierung eines Prozesses geschieht dabei über den so genannten *Rang*. Der Prozess mit dem *Rang* Null wird oft auch als *Masterprozess* bezeichnet, da diesem durch den Anwendungsprogrammierer oft spezielle Aufgaben zugewiesen werden.

MP-MPICH bietet verschiedene MPI-Bibliotheken für Rechner an, welche mit Unix/Linux oder Windows Betriebssystemen laufen. Die Unterstützung von bestimmten Plattformen und Netzwerken geschieht dabei über so genannte *Devices*. Das *ch_smi* Device [13] unterstützt Cluster mit SCI-Interconnect auf Basis von X86 und Sparc sowie die Betriebssysteme Linux, Solaris und Windows. Durch *ch_wsock2* [9] werden X86 basierte Rechner mit Windows-Betriebssystem und Kommunikation mittels TCP/IP über die Windows Sockets 2 Schnittstelle unterstützt. Zusätzlich unterstützt *ch_wsock2* die Kommunikation über gemeinsamen Speicher, wenn sich mehrere Prozesse auf dem selben Rechner befinden. Das neueste Device des Lehrstuhls für Betriebssysteme ist *ch_usock*. Hierbei handelt es sich um ein reines TCP/IP Device, welches auch Rechner mit Linux als Betriebssystem unterstützt. Mittels *ch_usock* ist es auch möglich, Prozesse, die zu einer MPI-Applikation gehören, sowohl auf Linux als auch auf Windows Rechnern im Gemischtbetrieb laufen zu lassen.

Beim Start der MPI-Applikation muss das passende Device für die entsprechende Plattform gewählt werden. Zusätzlich sind die Kommandozeilenparameter der einzelnen MPI-Prozesse abhängig von dem gewählten Device.

3. MP-Cluma

Bei MP-Cluma handelt es sich um eine verteilte Applikation, wie in Abbildung 2 gezeigt. Die Kommunikation zwischen den einzelnen Teilen wird mittels CORBA [6] durchgeführt.

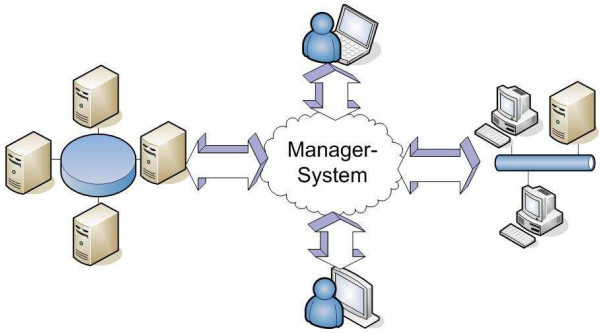

Abbildung 2. Schematische Struktur von MP-Cluma

Auf jedem Rechenknoten läuft ein Programm, welches den Start von MPI-Prozessen auf diesem Knoten ermöglicht. Dieses Programm bietet zusätzlich Aufgaben zur Verwaltung an und übermittelt Informationen über den Rechenknoten.

Das Managersystem verwaltet Informationen über die verfügbaren Rechenknoten und die Benutzer des Systems. Es besteht aus einem in Java geschriebenen Managerprogramm sowie zwei CORBA-Diensten. Dem Namensdienst [5], in welchen sich der Manager und die Rechenknoten eintragen und dem Ereignisdienst [4], über den Ereignisse zwischen den Komponenten ausgelöst werden können.

Das Managerprogramm, oder kurz der Manager, hält Informationen über alle zur Verfügung stehenden Rechenknoten bereit. Die Rechenknoten werden dabei in einer hierarchischen Struktur verwaltet. Informationen dazu sind in einer Konfigurationsdatei an jedem Rechenknoten gespeichert. Durch dieses System kann die physikalische Struktur der Rechenknoten innerhalb des *Office Grid* durch MP-Cluma repräsentiert werden.

Der Manager verwaltet ebenfalls die Rechte für alle Benutzer des Systems. So kann der Zugriff von Benutzer auf bestimmte Rechenknoten beschränkt werden. Zusätzlich werden für jeden Benutzer passwortgeschützte Informationen über die Betriebssystemprofile gespeichert. Ein Benutzer meldet sich mit den MP-Cluma Accountdaten im System an, der Manager wählt entsprechend den gewählten Rechenknoten das betriebssystemspezifische Benutzerprofil aus und verwendet dieses für den Prozessstart.

Der Benutzer kann die MPI-Applikation über ein spezielles grafisches Frontend starten oder über ein spezielles `mpiexec`. Innerhalb des grafischen Frontends sieht der Benutzer alle verfügbaren Rechenknoten in einer Baumstruktur, welche der physikalischen Struktur der Rechner entspricht. So können beispielsweise alle Rechner eines Clusters zusammengefasst werden. Der Pfad /DE/LFBS/P3

würde alle Rechner des Pentium 3-Clusters des Lehrstuhls enthalten, der Pfad /DE/LFBS/P4 alle Rechner des P4-Clusters.

3.1. Applikationsstart

Um eine MPI-Applikation zu starten, wählt der Benutzer eine Gruppe von Rechnern aus. Für die Auswahl stehen zusätzlich zu den hierarchischen Informationen Informationen über das Betriebssystem und die Hardware, wie Taktrate und Anzahl der Prozessoren, zur Verfügung. Es wird ebenfalls angezeigt, ob auf einer Arbeitsstation momentan Benutzer eingeloggt sind. Zusätzlich kann sich der Benutzer von MP-Cluma anzeigen lassen, welche Prozesse momentan auf den möglichen Zielknoten laufen, um so eine Abschätzung über die momentane Auslastung zu erhalten. Die Maximalanzahl der MPI-Prozesse wird durch die Angabe einer Zahl von *Slots* festgelegt. Diese Zahl wird dem Benutzter angezeigt sowie die momentane Anzahl freier *Slots* pro Rechenknoten.

Der Benutzer wählt für jedes vertretene Betriebssystem eine passende ausführbare Datei mit eventuellen Startparametern und ein passendes *Device* aus. Es ist auch möglich, diese Werte für jeden Knoten individuell zu wählen. Dann spezifiziert der Benutzer noch eine minimale und maximale Anzahl von Prozessen, mit denen die Applikation laufen soll. Falls der Benutzer entsprechende Rechte besitzt, kann er die Applikation sofort starten, ohne dass ein Scheduling stattfindet. Im Normalfall wird diese Beschreibung an den Scheduler von MP-Cluma übermittelt, der Teil des Managers ist. Nach welchen Kriterien die Auswahl des Startzeitpunktes und der Rechner für den Applikationsstart stattfindet, wird im folgenden Kapitel 4 erklärt.

4. Scheduling

Generell plant ein Scheduler den Ablauf eines Jobs in Abhängigkeit von anderen Jobs und verfügbaren Ressourcen. Im Fall von MP-Cluma besteht ein Job aus dem quasi gleichzeitigen Start aller Prozesse, die zu einer MPI-Applikation gehören. Wir gehen davon aus, dass diese MPI-Applikationen in sich geschlossen und nicht voneinander abhängig sind. Daher ist die Reihenfolge der Jobs beliebig. Verschiedene Jobs konkurrieren jedoch um die so genannten *Slots*, das heißt, um die zur Verfügung stehenden „freien Plätze“ auf den Rechenknoten. Dabei muss berücksichtigt werden, dass die Gruppe der Zielknoten durch den Benutzer vorgegeben ist sowie die minimale und maximale Anzahl von Prozessen.

Der Scheduler muss also für jeden Job innerhalb aller Rechenknoten eine bestimmte Anzahl und Menge von Knoten spezifizieren, auf denen der Prozessstart stattfinden soll. Für eine Menge von Jobs wird dann eine Ablaufreihenfolge bestimmt, in der die MPI-Applikationen auf den Knoten laufen. Die Bestimmung einer solchen Ablaufreihenfolge bezeichnet man als *Scheduling*, die Ablaufreihenfolge selbst als *Schedule*.

Der Begriff *gültiger Schedule* bezeichnet eine Ablaufreihenfolge, bei der jeder Job während seiner Laufzeit alle benötigten Ressourcen zur Verfügung stehen hat und über diese verfügen kann. Für die Ermittlung einer Ablaufreihenfolge ist jedoch das Wissen um die Laufzeit eines Jobs wichtig. In einem *Office Grid* ist durch die Heterogenität der Umgebung die Abschätzung der Laufzeit schwierig. Zusätzlich erschwert die flexible Anzahl der Prozesse eine Bestimmung. Von daher wird als Basis für die Laufzeit einer Applikation eine Laufzeitabschätzung des Benutzers verwendet.

Der Benutzer kann über seine Angaben bezüglich der Spanne in der Prozessanzahl und der Heterogenität der Gruppe der gewählten Rechenknoten eine Abschätzung über die Heterogenität des Jobs treffen. So ist ein Job, der auf 3 oder 4 Knoten des P4-Clusters läuft, wesentlich homogener als ein Job, der mit 2 bis 20 Prozessen auf allen verfügbaren Arbeitsstationen des Instituts gestartet werden soll. Davon ausgehend bietet MP-Cluma zwei Möglichkeiten, wie mit Laufzeitabschätzung des Benutzers verfahren werden soll.

Eventuell hat der Benutzer die MPI-Applikation schon ein- oder mehrfach in einer ähnlichen Konfiguration gestartet und kann daher eine gute Laufzeitabschätzung abgeben. In diesem Fall wird die Abschätzung des Benutzers unmodifiziert übernommen. Allerdings wird in diesem Fall nachgehalten, wie genau die Abschätzung war. Falls ein Benutzer wiederholt die Laufzeit seiner Applikation als zu gering angibt, wird auf die Abschätzung zukünftiger Applikationen eine „Strafzeit“ aufaddiert. So soll verhindert werden, dass sich Benutzer durch die Angabe zu kurzer Laufzeiten einen Vorteil beim Scheduling verschaffen.

Wenn der Job heterogen ist, dann kann der Benutzer seine Laufzeitabschätzung als ein Prozess auf Basis des langsamsten Rechenknotens angeben. In diesem Fall wird die Abschätzung durch den Manager modifiziert, je nach Anzahl der tatsächlich dem Job zur Verfügung gestellten Rechenknoten.

Ein *optimaler Schedule* ist ein auf bestimmte Kriterien hin optimierter Schedule. Kriterien können beispielsweise eine gleichmäßige Auslastung der Ressourcen oder eine minimale Gesamtlaufzeit aller Jobs sein. Das Problem, einen *optimalen Schedule* zu bestimmen, ist NP-Vollständig [2]. Ein Ansatz, das Problem zu lösen, ist die Verwendung eines Genetischen Algorithmus (GA). Zusätzlich zu einigen Ansätzen GA in homogenen Umgebungen einzusetzen, gibt es auch Lösungen in heterogenen Umgebungen [7, 12, 8, 14].

In den vorgenannten Arbeiten werden meist von einander abhängige Task geplant, wobei sich die Heterogenität nur auf unterschiedliche Ausführungsgeschwindigkeiten bezieht. Beispielsweise repräsentiert der in [7] vorgestellte Algorithmus einen Job anhand seiner benötigten MFLOPs. Alle verfügbaren Ressourcen werden ebenfalls durch ihre Geschwindigkeit in MFLOPs pro Sekunde charakterisiert. Diese Charakterisierung reicht für MP-Cluma nicht aus, da hier stärkere Restriktionen in Bezug auf mögliche Zielrechner bzw. Zielplattformen existieren.

4.1. Der Genetische Algorithmus in MP-Cluma

Ein GA wird benutzt, um aus einer Menge von möglichen Schedule einen optimalen Schedule zu entwickeln. Die Methode ist an die evolutionären Techniken der Natur angelehnt. Aus einem Pool von Chromosomen werden durch Veränderungen neue Chromosomen erzeugt. Aus diesen wird durch Anwenden bestimmter Qualitätskriterien der Pool für den nächsten Iterationsschritt bestimmt. Innerhalb des genetischen Algorithmus stellt ein Chromosom einen möglichen Schedule dar.

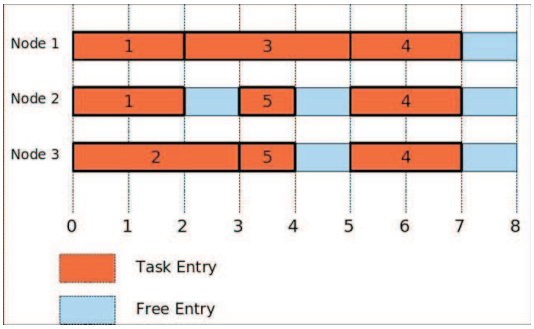

Abbildung 3. Beispiel für einen gültigen Schedule

Zur Erzeugung des Startpools werden daher eine Reihe unterschiedlicher gültiger Schedule benötigt. Im Fall von MP-Cluma ist nicht nur die Reihenfolge der Jobs sondern auch die Anzahl der tatsächlichen Prozesse innerhalb der gegebenen Spanne variabel. Gleichzeitig ist der Anspruch an einen gültigen Schedule besonders hoch, da für die Prozesse eines Jobs nicht der komplette Rechnerpool verwendet werden darf, sondern nur Knoten innerhalb der von Benutzer vorgegebenen Gruppe. Ein zufällig generiertes Schedule wäre unter unseren Rahmenbedingungen mit hoher Wahrscheinlichkeit ungültig.

Daher werden wir keine Chromosomen mit willkürlichem Schedule erzeugen und diese nachher auf Gültigkeit testen, sondern wir verwenden ausschließlich Chromosomen, die ein gültiges Schedule repräsentieren. In Abbildung 3 ist ein Chromosom gezeigt, welches einen gültigen Schedule enthält. Ein gültiges Schedule wird erzeugt, indem erst eine zufällige Reihenfolge der Jobs gewählt wird und dann der Reihe nach bei jedem Job eine zufällige Anzahl von Prozessen zwischen dem gegebenen Minimum und Maximum gewählt wird. Dann wird getestet, ob die gewählten Knoten freie Zeiten haben. Beim 1. Job sind noch keine Knoten belegt, er kann sich jeweils vorne in die Warteschlangen von Knoten eins und zwei einreihen. Job 2 soll nur auf einem Knoten ausgeführt werden, der früheste Startzeitpunkt liegt zum Zeitpunkt Null auf Knoten 3. Job 3 wird nach Job 1 auf Knoten 1 ausgeführt. Da Job 4 alle drei Knoten belegen will, ist der Startzeitpunkt für die Prozesse von Job 4 auf allen drei Knoten der Zeitpunkt 5. Auf dem Knoten 2 existieren jetzt ab dem Zeitpunkt 2 drei freie Zeiteinheiten und auf dem Knoten 3 existieren ab dem Zeitpunkt drei zwei freie Zeiteinheiten. Der Job 5 kann jetzt in diesen freien Bereichen auf den Knoten zwei und drei platziert werden.

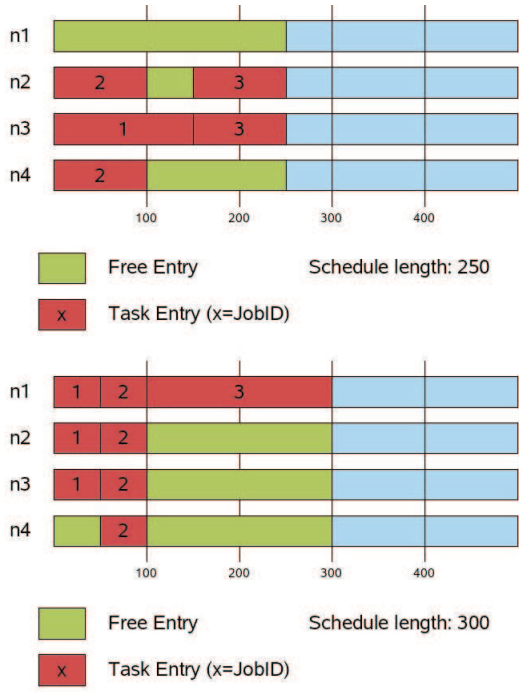

Abbildung 4. Zwei zufällige Chromosomen

Die Menge solcherart generierter Chromosome mit gültigem Schedule bildet den Ausgangspool des Genetischen Algorithmus. Operationen, die auf diesen Chromosomen durchgeführt werden, sind die Mutation und das Tauschen, welche im Folgenden erklärt werden.

4.1.1 Mutation

Bei der Mutation werden zwei Eigenschaften des Chromosoms geändert. Zum einen wird zufällig ein Job innerhalb

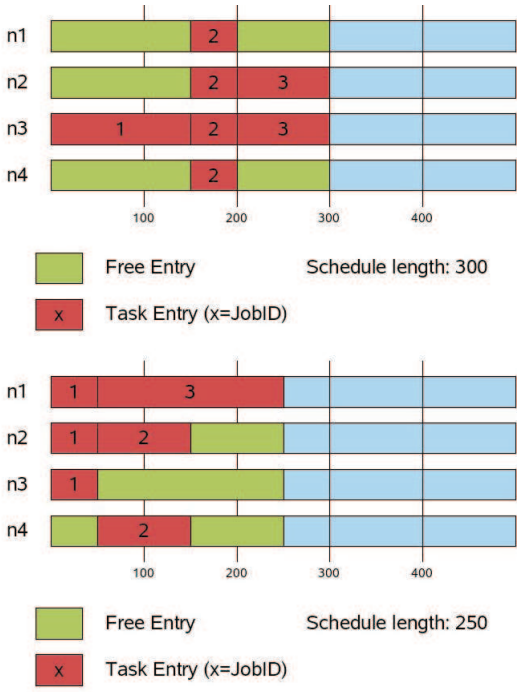

Abbildung 5. Chromosomen nach dem Tauschvorgang

des Chromosoms ausgewählt und die Anzahl der Zielprozesse verändert, zum anderen wird die Reihenfolge der Jobs modifiziert. Jetzt wird aus dem ursprünglichen Schedule und dem modifizierten Job ein neues gültiges Schedule erstellt.

4.1.2 Tauschen

Beim Tausch werden zufällig zwei Chromosomen gewählt und aus diesem wird per Zufall ein Job ausgewählt. Die Zielknoten der Jobs werden jetzt zwischen den beiden Chromosomen vertauscht. Die Reihenfolge der Jobs bleibt dabei unverändert, allerdings können sich die Startzeiten der nachfolgenden Jobs verschieben.

Ein Beispiel für den Tauschvorgang ist in den Abbildungen 4 und 5 gezeigt. Abbildung 4 stellt die ursprünglichen Chromosomen dar, die zufällig für den Tauschvorgang ausgewählt wurden. Job 2 wird jetzt ausgewählt, um zwischen den Chromosomen getauscht zu werden. Im ersten Chromosom belegt Job 2 die Knoten n2 und n3, im zweiten Chromosom belegt der Job alle Knoten. Nach dem Tauschvorgang belegt Job 2 alle Knoten im ersten Chromosom und nur noch die Knoten n2 und n3 im zweiten Chromosom. Das Ergebnis ist in Abbildung 5 dargestellt.

4.1.3 Test und Justierung

Im Folgenden wird das Scheduling von vier verschiedenen Jobs exemplarisch dargestellt, um die Arbeitsweise des Genetischen Algorithmus zu verdeutlichen:

- Job 1: Ausführungszeit: 50, Max. Prozesse: 2, Min. Prozesse: 2, mögliche Zielknoten: n1, n2, n3 und n4.
- Job 2: Ausführungszeit: 200, Max. Prozesse: 3, Min. Prozesse: 1, mögliche Zielknoten : n2, n3, und n4.
- Job 3: Ausführungszeit: 20, Max. Prozesse: 2, Min. Prozesse: 1, mögliche Zielknoten: n1 ,n3 und n4.
- Job 4: Ausführungszeit: 400, Max. Prozesse: 4, Min. Prozesse: 1, mögliche Zielknoten: n1, n2, n3 und n4.

Da der Ausgangspool bereits aus Chromosomen besteht, die ein gültiges Schedule enthalten, dient der Genetische Algorithmus nur zur Optimierung. Da wir also keine Chromosomen im Pool haben, die ein ungültiges Schedule darstellen, können wir den Pool klein halten. Um den Einfluss der verschiedenen Parameter auf die Güte des Ergebnisses zu ermitteln, werden im Folgenden vier unterschiedlich eingestellte Algorithmen vorgestellt. Es wird sich zeigen, dass die Qualität des Endergebnisses steigt, wenn die Häufigkeit der Modifikation von Chromosomen pro Iterationsschritt zunimmt. In allen Fällen wird jeweils ein Pool mit 3 Chromosomen betrachtet.

Im ersten Ansatz werden fünf Iterationen mit jeweils einer Mutation und einem Tauschvorgang durchgeführt. Abbildung 6 zeigt vier unabhängige Ergebnisse, die jeweils einem Durchlauf des Algorithmus entsprechen. Der beste Schedule hat eine Gesamtlaufzeit von 201 ZE, der schlechteste eine Gesamtlaufzeit von 291 ZE. Die Schwankung in der Gesamtlaufzeit der Ergebnis-Schedules verdeutlicht, dass die Anzahl von Iterationen, Mutationen und Tauschvorgängen nicht ausreichend ist, um zuverlässig gute Ergebnisse zu generieren.

Als nächstes werden zwei Fälle betrachtet, die auf unterschiedliche Weise die Genauigkeit und damit auch den Berechnungsaufwand im Vergleich zum vorherigen Beispiel erhöhen. Im ersten Fall wird nur die Anzahl der Iterationen von 5 auf 50 erhöht. Im zweiten Fall wird die Anzahl von Mutationen und Tauschvorgängen pro Iterationsschritt erhöht, es finden jedoch nur 5 Iterationen statt.

In Abbildung 7 erkennt man durch die erhöhte Anzahl an Iterationen eine Verbesserung in den Laufzeiten. Der beste Schedule hat eine Gesamtlaufzeit von 191 ZE, der schlechteste eine Gesamtlaufzeit von 225 ZE, damit schwanken die Laufzeiten in begrenztem Maß.

Die Ergebnisse der vier Durchläufe mit wenig Iterationen aber größeren Veränderungen an den Chromosomen sind in Abbildung 8 dargestellt. Hier entstehen als Endergebnisse jeweils zwei Schedule mit einer Laufzeit

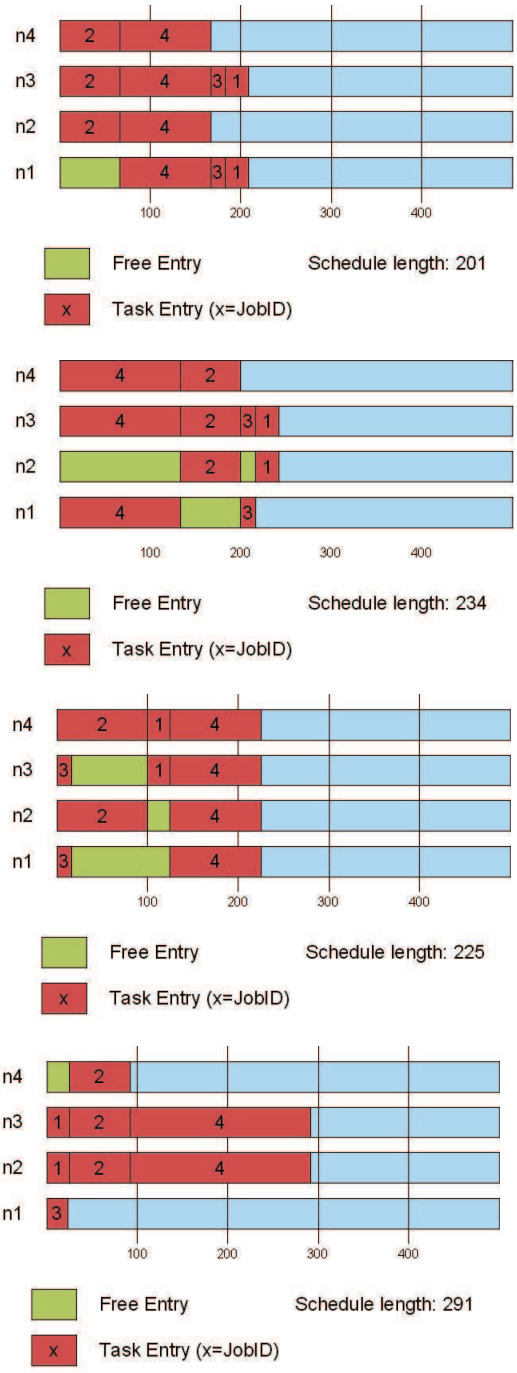

Abbildung 6. Vier Endergebnisse nach 5 Iterationen mit jeweils einer Mutation und einem Tauschvorgang

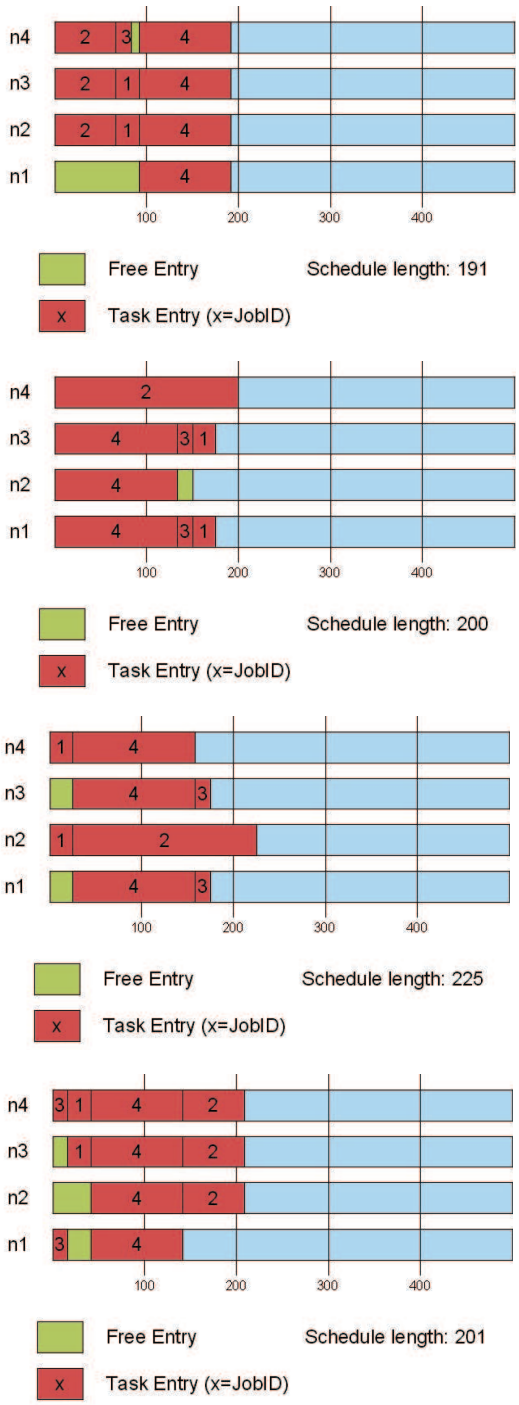

Abbildung 7. Vier Endergebnisse nach 50 Iterationen mit jeweils einer Mutation und einem Tauschvorgang

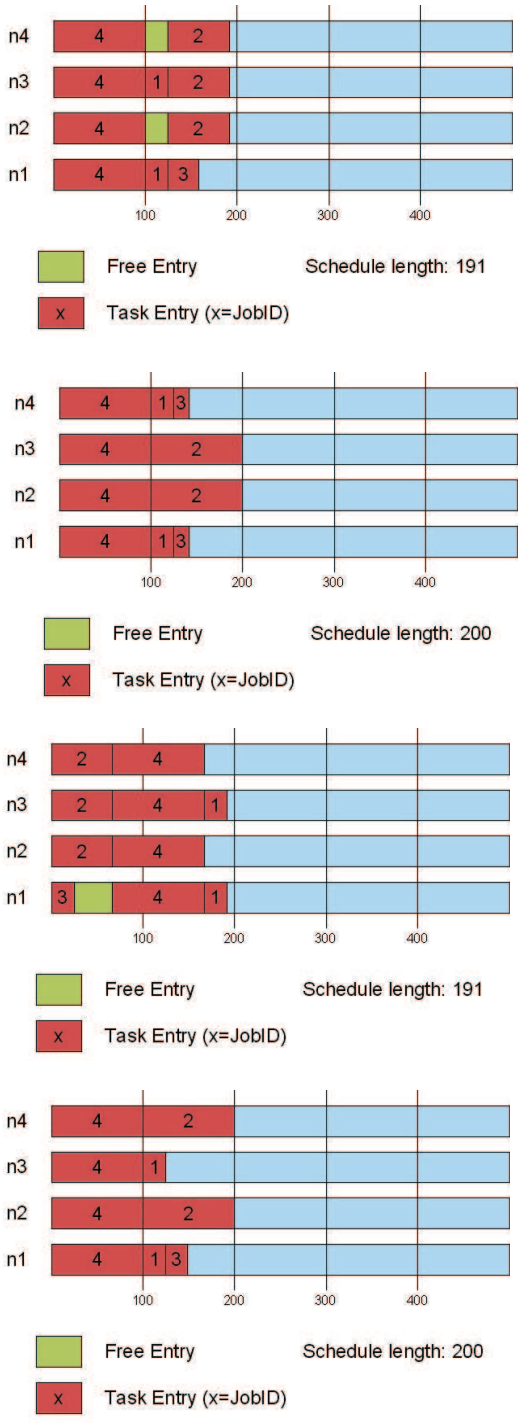

Abbildung 8. Vier Endergebnisse nach 5 Iterationen mit mehreren Mutation und Tauschvorgängen

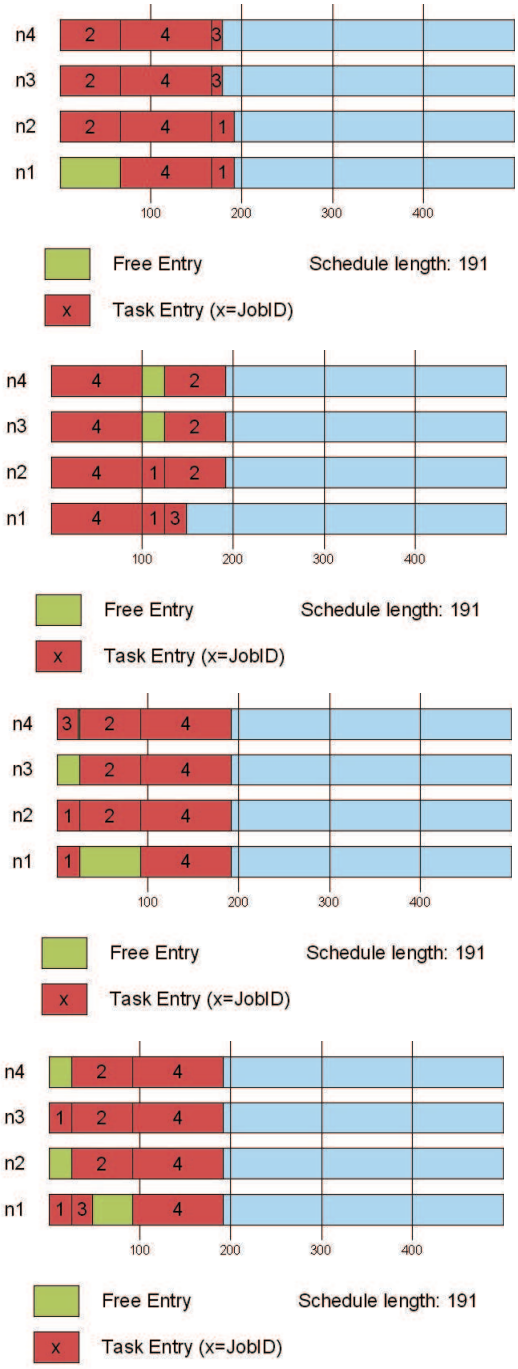

Abbildung 9. Vier Endergebnisse nach 50 Iterationen mit mehreren Mutation und Tauschvorgängen

von 191 ZE und 200 ZE. Die besseren Ergebnisse dieser Durchläufe und vor allem die geringe Schwankung der Laufzeiten zeigen, dass eine höhere Anzahl von Mutationen und Tauschvorgängen effizienter ist, als mehr Iterationsschritte zu verwenden. Der Grund dafür ist, dass bei mehreren Durchläufen die Chromosomen in kleineren Schritten progressiv verbessert werden. Im Gegensatz dazu werden bei mehreren Mutationen und Vertauschungen größere Variationen in jeder Iteration betrachtet.

Um zu überprüfen, ob eine längere Laufzeit das Ergebnis weiter verbessert, wird der Algorithmus mit mehreren Veränderungen in jedem Iterationsschritt nun auch mit 50 Iterationsschritten durchgeführt. In Abbildung 9 sind die Ergebnisse für diesen Fall dargestellt. Alle Endergebnisse enthalten ein Schedule mit einer Laufzeit von 191 ZE.

Man kann erkennen, dass der Algorithmus nach einer bestimmten Anzahl von Iterationen eine optimale Laufzeit für das Schedule erreicht. In Abbildung 8 ist dies bei zwei Durchläufen bereits nach 5 Iterationen der Fall. Eine Erhöhung auf 50 Iterationen hatte hier keinen wesentlichen Vorteil gebracht.

Die Laufzeit aller oben angeführten Beispiele bewegt sich im Sekundenbereich. Sowohl die Vergrößerung der Komplexität eines Iterationsschrittes als auch die Erhöhung der Iterationsschritte um den Faktor zehn haben jeweils zu einer Verzehnfachung der Laufzeit geführt.

Um Ressourcen effizient zu nutzen, ist es wichtig, die Parameter für den Algorithmus optimal bestimmen zu können. Da sich die Anzahl der Jobs mit ihren speziellen Rahmenbedingungen und die verfügbaren Rechenknoten zur Laufzeit ändern können, kann es nötig sein, die Parameter dynamisch der veränderten Situation anzupassen. Ein weiteres Augenmerk muss darauf gelegt werden, wie sich die Laufzeiten bei einer größeren Komplexität des Systems entwickeln.

5. Zusammenfassung und Ausblick

In diesem Artikel haben wir die erfolgreiche Integration einen Schedulers in MP-Cluma vorgestellt. Obwohl die Rahmenbedingungen für ein gültiges Schedule eingeschränkt sind, liefert der Genetische Algorithmus auch mit kleinem Chromosomenpool und wenigen Iterationen sehr gute Ergebnisse. Die nächste Aufgabe ist eine Bestimmung möglichst optimaler Parameter von Chromosomenpool, Iterationsanzahl und Stärke der Veränderung in Abhängigkeit von der Anzahl und Struktur der Jobs sowie der Anzahl der zur Verfügung stehenden Rechner.

Als erster Schritt ist daher vorgesehen, diese Parameter zur Laufzeit veränderbar zu machen. Eine Einstellmöglichkeit kann in das bereits vorhandene Administrationsprogramm von MP-Cluma eingebaut werden. Optimal wäre eine automatische Anpassung der Parameter an die äußeren Gegebenheiten durch den Scheduler selbst.

Des Weiteren wäre es interessant, weitere Schedulingmechanismen in MP-Cluma einzubauen und die Ergebnisse unter verschiedenen Rahmenbedingungen mit denen des Genetischen Algorithmus zu vergleichen.

Literatur

[1] Chair for Operating Systems, RWTH-Aachen, University. *MP-MPICH – User Documentation & Technical Notes*.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.

[3] MPI Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputing Applications*, 1994.

[4] OMG Technical Document formal/01-03-01. *Event Service Specification*, 1.1 edition, 2001.

[5] OMG Technical Document formal/04-10-03. *Naming Service Specification*, 1.3 edition, 2004.

[6] OMG Technical Document orbos/98-05-05. *CORBA Messaging Specification*, 1998.

[7] A. J. Page and T. J. Naughton. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 6*, page 189.1, Washington, DC, USA, 2005. IEEE Computer Society.

[8] A. J. Page and T. J. Naughton. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artif. Intell. Rev.*, 24(3-4):415–429, 2005.

[9] S. Schuch, C. Clauss, T. Arens, S. Lankes, and T. Bemmerl. Entwurf und Implementierung einer Windows-spezifischen, TCP/IP-basierten Geräteschnittstelle für MP-MPICH. In *Tagungsband zum Workshop KiCC05*, TU Chemnitz, Germany, November 2005.

[10] S. Schuch and M. Pöppe. MP-Cluma - A CORBA Based Cluster Management Tool. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2004)*, Las Vegas, USA, June 2004.

[11] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.

[12] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *J. Parallel Distrib. Comput.*, 47(1):8–22, 1997.

[13] J. Worringen. *Effizienter Nachrichtenaustausch auf speichergekoppelten Rechnerverbundsystemen mit SCI Verbindungsnetz*. PhD thesis, RWTH Aachen, 2003.

[14] H. Yu, D. C. Marinescu, A. S. Wu, and H. J. Siegel. A genetic approach to planning in heterogeneous computing environments. *ipdps*, 00:97a, 2003.