Fachbereich Mathematik und Informatik
Arbeitsgruppe Software Engineering

Erweiterung der Jobplanung eines Workloadmanagers für HPC-Systeme

Autor
Sören Hübner
soerenh@inf.fu-berlin.de

Betreuer
Dipl.-Inf. Barry Linnert

Zweitgutachter
Dr.-Ing. Jochen Schiller

Zusammenfassung

Die Arbeit behandelt ein Thema aus dem Bereich des High-Performance-Cluster-Computings. In dieser wird die Benutzung eines Workloadmanagers eines solchen Cluster-Computers um zusätzliche Funktionalität erweitert. Speziell wird es um das OpenCCS-Projekt gehen. Statt bei einer Jobplanung neben der Ablaufdauer auch explizit die benötigten Ressourcen anzugeben, kann man nun stattdessen ein Programmablaufmodell übergeben, aus welchem die Ressourcen automatisch abgeleitet werden und danach der Job wie üblich bearbeitet wird. Das Programmablaufmodell und die Ableitung der benötigten Ressourcen aus diesem werden in dieser Arbeit als gegeben angesehen, der Fokus liegt auf der Kapselung einer Schnittstelle.

28. Februar 2018

Eigenständigkeitserklärung

Hiermit erkläre ich, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Genutzte Grafiken wurden selbst erstellt. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Sören Hübner
Berlin, 28. Februar 2018

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit erkundet einen neuen Ansatz, um das Job-Scheduling für HPC-Cluster-Systeme zu optimieren. Das Thema bildet die Grundlage für weitere Forschung und bettet sich ein in aktuelle Forschungsbemühungen zur Verbesserung momentaner Produktivsysteme. Ziele sind eine bessere Auslastung von Clustersystemen, die Entlastung von ProgrammiererInnen und die generelle Erhöhung der Benutzbarkeit und Dienstgüte („Quality of Service“). Im Zuge dieser Arbeit wird eine Schnittstelle für ein bestehendes Projekt geschaffen, um diese Ziele zu verfolgen.

Nach der Einleitung in das Thema und der Problembeschreibung in Abschnitt (1), beschreibt diese Arbeit in (2) zunächst das erweiterte Projekt anhand von dessen Besonderheiten und Problemen. Danach wird in (3) die Implementierung der Schnittstelle beschrieben, die im nachfolgenden Abschnitt (4) ausgewertet wird. Als letztes folgt in (5) ein Ausblick auf weiterführende Forschungsmöglichkeiten. Die Liste der verwendeten Literatur findet sich in (6) und im Anhang (7) ist unter Anderem der Quellcode der Schnittstelle zu finden.

In diesem Zusammenhang ist die Rede von HPC, also High-Performance-Computing. Ein HPC-System unterscheidet vom üblichen Heimcomputer oder einer Workstation über die massive Rechenkapazität und die physikalische Größe, denn ein HPC-System umfasst hunderte oder tausende Knoten mit jeweils einer Vielzahl von Prozessoren in mehreren Serverschränken [1].

Noch spezieller geht es um HPC-Cluster-Computer [2] in Abgrenzung zu HPC-Massively-Parallel-Processor-Arrays (MPPA) [3]. Cluster bestehen meist aus tausenden, über ein Hochgeschwindigkeitsnetzwerk eng gekoppelter, vollständiger Maschinen und ein MPPA im Gegensatz dazu, ist eine einzelne Maschine mit speziellen, eng gekoppelten Prozessoren und besonderer Speicherstruktur. Da Cluster aus herkömmlicher Verbraucherhardware bestehen, sind diese bei ähnlicher Performance um einiges wirtschaftlicher [4]. MPPA ist neben Clustersystemen die einzige Supercomputerarchitektur, die das TOP500-Projekt [1] für die 500 stärksten HPC-Systeme auflistet.

Ein Cluster-Computer ist eine Ansammlung von möglichst homogenen Computern und anderen Ressourcen wie GPUs, Speichersystemen, Netzwerken, Lizenzservern, usw. [2]. Jeder dieser Computer wird als „Knoten“ bezeichnet und auf jedem Knoten läuft ein eigenes und vollständiges Betriebssystem. Untereinander sind die Knoten über ein Hochgeschwindigkeitsnetzwerk in spezieller Topologie verbunden. Die Planung und Ausführung von Jobs wird über Software geregelt, durch sogenannte Workloadmanager. Diese Workloadmanager reduzieren einerseits die technische Administration durch

Menschen und bieten den BenutzerInnen eine einheitliche Ansicht mit einem möglichst leicht zu bedienenden Interface.

Die zwei Hauptaufgaben von Workloadmanagern sind also:

- Das Abstrahieren von der zugrundeliegenden Hardware durch das Bereitstellen einer virtuellen Maschine. Konkret heißt das, dass dem Workloadmanager Jobs inklusive Anweisungen übergeben werden, die dann auf freien Knoten abgearbeitet werden. Bei wiederholter Ausführung des gleichen Jobs kann dieser auf den gleichen oder komplett anderen Knoten berechnet werden, die Menge der einzelnen Maschinen kann wie ein einzelnes System benutzt werden.
- Die Ressourcenverwaltung anhand von Informationen über das Cluster und die Jobs. Damit ist die genaue Zuordnung von Job auf vorhandene Ressourcen wie Knoten, Verbindungsleitungen, usw. gemeint.

Im Bereich des Cluster-Computings ist ein Job eine Aufgabe, ein Programm oder eine Berechnung, die ausgeführt wird bzw. werden soll. Jobs lassen sich meist in eine Menge von Prozessen oder Threads aufteilen, um auf mehreren Prozessoren oder Knoten bearbeitet zu werden. Dies ähnelt dem Scheduling auf Multi-Core-Prozessoren moderner Computer, z.B. auf dem normalen Heimcomputer oder einem einzelnen Knoten in einem Cluster-Rechner. Diese verteilte Bearbeitung einzelner Jobs ist das Hauptargument für Cluster-Computer, neben der gleichzeitigen Ausführung einer Vielzahl von Jobs.

Während der Jobplanung wird entschieden wann welcher Job mit wie vielen Ressourcen ausgeführt wird. Ressourcen sind in diesem Fall z.B. Knoten, Prozessoren, Arbeitsspeicher, GPUs, Netzwerkbandbreite, Festplattenspeicher oder Softwarelizenzen [5].

1.1 Problembeschreibung

Die vorliegende Arbeit greift Forschungsbemühungen um den „Virtual Resource Manager“ (VRM) auf [6] [7]. Dabei geht es um die transparente Bereitstellung virtueller Ressourcen nach einem vorher festgelegtem „Vertrag“, in diesem Fall dem Programmablaufmodell. Diese Ressourcen-Virtualisierung bedeutet, dass Ressourcen von NutzerInnen unbemerkt zusammengelegt oder aufgeteilt werden, um geforderten Anfragen gerecht zu werden. Ziele sind der Ausbau einer serviceorientierten Struktur mit höherer Benutzbarkeit, aber auch besserer Auslastung und hoher Fehlertoleranz. Das soll erreicht werden mit Abstraktion und dem Einhalten von „Verträgen“. Beides wird durch die unten beschriebenen Programmablaufmodelle erzielt.

Dies soll im OpenCCS realisiert werden, das schon über vorhandene Ressourcen-Transparenz gewährleistet, dass die benötigten Ressourcen bereit stehen und genutzt werden, um den übergebenen Job auszuführen. Trotzdem gibt es noch Schwierigkeiten mit „weichen“ Anfragen, die an verschiedenen Zeitpunkten der Berechnung unterschiedliche Ressourcen benötigen oder in der Menge der Ressourcen flexibel sind. Ein Programmablaufmodell bietet mehr Informationen als eine einfache Angabe von Ressourcen, wodurch diese Probleme angegangen werden können.

Das betrachetete Thema ist seit einiger Zeit Bestandteil der Forschung, zuvor ist es allerdings noch nicht zu einer Umsetzung davon in einem funktionierenden Produktivsystem gekommen. Dies soll mit der vorliegenden Arbeit am OpenCCS-Projekt geändert werden.

Mit dieser Arbeit soll eine Funktion gekapselt werden, die die Auswertung und Anwendung der formalisierten Beschreibungen von Jobs ermöglichen soll. Hierfür sollen sogenannte Programmablaufmodelle genutzt werden. Ziel der Arbeit ist die Kapselung der einer effektiven Schnittstelle, die flexibel an die Ansprüche spezieller Ablaufmodelle angepasst werden kann.

1.2 Abgrenzung des Problems

Es folgt die Beschreibung zweier Probleme, die für diese Arbeit wichtig sind, aber an anderer Stelle gelöst werden. Das erste Problem ist das Zuordnungsproblems, das z.B. in [8] oder in [9] beschrieben wird. Das zweite ist die Definition eines speziellen Programmablaufmodells, siehe hierzu zum Beispiel [10], [11] und [12] für Lösungsansätze.

1.2.1 Das Zuordnungsproblem

Das Zuordnungsproblem ist ein Optimierungsproblem aus der Graphentheorie. Bei dem Problem gibt es zwei Ansammlungen von Elementen: eine Menge von „Arbeitern“ und eine Liste von zu erledigenden „Aufgaben“. Die Arbeiter haben verschiedene Fertigkeiten und die Aufgaben haben unterschiedliche Anforderungen. Das Problem ist es die optimale Aufteilung von Aufgaben auf Arbeiter, sodass die gesamte Arbeit schnellstmöglich erledigt wird.

In dem Zusammenhang mit Cluster-Computing sind die „Arbeiter“ die Knoten des Rechnerverbundes und die „Aufgaben“ sind Teile des zu berechnenden Jobs. Es soll bestimmt werden in welche Teilstücke ein Job für ein bestimmtes Cluster zerlegt werden muss und welche Knoten diese dann bearbeiten. Die Heterogenität der Knoten ergibt sich aus deren Ausstattung, z.B. mit oder ohne GPU, ein oder zwei CPUs, etc. Jeder Job benötigt unterschiedlich viel Rechenkapazität und dazu eventuell noch weitere Ressourcen

wie Netzwerkbandbreite oder Softwarelizenzen [8] [9]. Da die Aufgabenverteilung auch diese Ressourcen beachten sollte, wird das Zuordnungsproblem NP-schwer.

Für eine geringe Anzahl Aufgaben und Knoten wäre die Aufgabe leicht zu lösen. Moderne Cluster-Computer bestehen aber aus tausenden von Knoten, sodass Heuristiken angewandt werden müssen, deren Entwicklung und Implementierung aktueller Forschungsgegenstand sind [13]. Die vorliegende Arbeit bietet die Grundlage für die Möglichkeit einer solchen Implementierung, in der sich optimale Zerlegung aus einem Programmablaufmodell ergibt, das detaillierte Informationen über einen Job und die benötigten Ressourcen enthält.

1.2.2 Programmablaufmodelle

Ein Programmablaufmodell ist eine formalisierte Beschreibung der genutzten Ressourcen eines Jobs. In diesem Modell könnte z.B. vermerkt sein, wann im Quellcode eines Jobs Threads oder Prozesse abgezweigt oder zusammengeführt werden und wie viel Arbeitspeicher gebraucht wird. Das Programmablaufmodell lässt sich dann nutzen, um das Zuordnungsproblem effektiver zu lösen, d.h. welcher Teil eines Jobs auf welchen Knoten mit wie vielen Ressourcen laufen sollte. In herkömmlichen Systemen wird nur eine sehr grobe Beschreibung der benötigten Ressourcen und der Maximallaufzeit übergeben, doch aus einem Programmablaufmodell lassen sich viel genauere Informationen ableiten.

Ein solches Programmablaufmodell muss von Menschen während der Planungsphase für Software eigens erstellt werden. Es handelt sich hierbei um eine besondere Umsetzung eines Algorithmus, die automatisch geparst werden kann. Für bestehende Programme kann ein Programmablaufmodell vermutlich nur mit großem Aufwand nachträglich erstellt werden und eine maschinelle Erzeugung ist nicht möglich. Ablaufpläne von Programmen kann man sich wie Flow-Charts [14] vorstellen, ergänzt mit Informationen über die gebrauchten Ressourcen. Es existiert schon eine Vielzahl verschiedener Modellarten für verschiedene Anwendungszwecke [10] [11] [12].

1.3 Bewertungskriterien der Erweiterung

Das wichtigste Kriterium ist die problemlose Erweiterbarkeit des Quellcodes, denn aufbauend auf der vorliegenden Implementierung sollen weitere Forschungsarbeiten ermöglicht werden. Hierzu ist eine gute Kapselung der entsprechenden Schnittstelle notwendig.

Eine möglichst einfache Bedienbarkeit ist vorzuziehen. Die Erweiterung benutzt die bestehende Bedienoberfläche mit und sollte sich ähnlich den schon

vorhandenen Funktionalitäten einbetten. Lesbarkeit des Codes und Güte der Dokumentation sind ebenfalls hilfreich, genauso wie Robustheit und aussagekräftige Fehlermeldungen in allen ungewollten Fällen.

Da der Ausbau eines schon vorhandenen Projekts betrachtet wird, sollte sich dieser möglichst gut in die vorhandene Codebasis einfügen, indem bestenfalls wenige neue globale Variablen und Funktionen eingeführt werden. Stattdessen ist die Nutzung von vorhandenen Methoden, Fehlerbehandlungmechanismen usw. wünschenswert. Die Orientierung an schon vorhandenem Code ist nötig. Es sollte auch die vorhandene Dokumentation in für das Projekt üblicher Weise angepasst werden, so z.B. auch die Hilfsfunktion.

An folgenden sechs Kriterien orientiert sich Qualitätsabschätzung:

1. Ist es möglich Programmablaufmodelle zu parsen und dafür Ressourcen zu reservieren? Kann man alternative Programmablaufmodelle ergänzen?
2. Ist der Code lesbar und ausreichend kommentiert und dokumentiert?
3. Sind die Ausgaben, v.a. der Fehler, lesbar und zielführend? Wie robust wird mit Argumenten umgegangen?
4. Wie viele neue Dateien, globale Variablen und Funktionen wurden erstellt oder verändert? Gibt es dabei Konfliktpotential?
5. Wurden neue Bibliotheken genutzt? Ergeben sich daraus neue Abhängigkeiten?
6. Wie vergleicht sich die Erweiterung mit dem schon Vorhandenen?

2 Das OpenCCS-Projekt

Diese Arbeit behandelt die Erweiterung eines bestehenden Projekts, des OpenCCS [15], um zusätzliche Funktionalität. Das OpenCCS stellt als planungsbasierter Workloadmanager für BenutzerInnen eine Schnittstelle für die Benutzung eines HPC-Cluster-Systems und für AdministratorInnen eine für die Verwaltung eines solchen bereit. „CCS“ steht dabei für „Computing Center Software“ und ist veröffentlicht unter der Open-Source-Lizenz GPL [16], weshalb es auch als „OpenCCS“ bezeichnet wird [17]. Die Auswahl des Projekts ergab sich aus der Aufgabenstellung, denn es ist Gegenstand von Forschungsprojekten an der Freien Univsersität Berlin.

Laut offiziellen Dokumenten sind die Designziele des OpenCCS die Bereitstellung einer homogenen Bedienoberfläche für eine Ansammlung verschiedener High-Performance-Systeme und Möglichkeiten der Beschreibung, Organisation und Verwaltung eines solchen HPC-Systems für die AdministratorInnen [17].

Nachfolgend wird die historische Entwicklung und der Einsatz des OpenCCS beschrieben. Es wird geklärt, dass der Vergleich mit ähnlichen Projekten aufgrund des planungsbasierten Schedulings des OpenCCS schwerfällt. Nach der Beschreibung einer Grobstruktur des OpenCCS wird der Abschnitt mit einer Bestandsaufnahme des vorhandenen Codes und der Dokumentation beendet.

2.1 Einordnung

Das Projekt wird momentan hauptsächlich vom Projektmanager Axel Keller [18] betreut. Der meiste Code stammt aus seiner Arbeit oder aus Studierendenprojekten. Die Codebasis ist an einigen Stellen recht alt, die erste wissenschaftliche Veröffentlichung dazu wurde von Axel Keller und Alexander Reinefeld im Jahr 1998 veröffentlicht [19] und beschreibt, dass seit dem Projektstart in 1992 zu dem Zeitpunkt ungefähr zwanzig Leute an dem Projekt gearbeitet hatten. Ansonsten sind bei weiteren Papern zum CCS insgesamt etwa zehn Autoren genannt, alle entweder von der TU Berlin, Universität Paderborn oder dem Paderborn Center for Parallel Computing. 2009 wurde das Projekt für das planungsbasierte Scheduling und Mapping überarbeitet, das für die vorliegende Arbeit von Bedeutung ist [20].

OpenCCS wird im Parallel Computing Center in Paderborn eingesetzt und entwickelt [21]. Der OCuLUS-Cluster in Paderborn hat 9920 Prozessorkerne, fast 50 Grafikkarten und 45 TB Arbeitsspeicher in 12 Serverschränken, verteilt auf über 600 Knoten, die jeweils über InfiniBand miteinander verbunden sind [22] [23]. Da das Projekt eine Open-Source-Lizenz hat und frei verfügbar ist, könnte die Software noch an anderen Orten im Einsatz sein. Allerdings wird es laut dem TOP500-Projekt in keinem der 500 leistungsstärksten Systeme eingesetzt [1].

Der verbreitetste Workloadmanager für Clustersysteme in den TOP500 ist Slurm [24], das laut Angaben der Entwickler auf 6 der 10 schnellsten TOP500-Rechner läuft, darunter dem zweitschnellsten Tianhe-2 und dem schnellsten Sunway TaihuLight. Letzterer hat allerdings keinen Vermerk über die Verwendung von Slurm auf der Website [25]. Slurm wird generell als fehlertolerant, sehr performant und gut skalierend beschrieben [26], ist jedoch warteschlangenbasiert, was den Vergleich zu OpenCCS schwierig macht. Der Projektumfang ist bei Slurm auch um einiges großer, denn ein Team von acht festangestellten ProgrammiererInnen kümmert sich darum, zusätzlich

zu hunderten freiwillig Beitragenden. OpenCCS wäre vergleichbar mit in ähnlichem Umfang betreuten, kleineren Projekten mit Forschungskontext, es gibt aber auch dort keine mit rein planungsbasierten Job-Scheduling. Ähnlich ist das von ForscherInnen der Universität Jerusalem [27] entwickelte EASY++, das zwar warteschlangenbasiert ist, aber Laufzeitabschätzungen zum Backfilling kurzer Jobs benutzt. Anders als das OpenCCS stützt sich EASY++ auf Job-Warteschlangen, wie alle anderen verbreiteten Workloadmanager.

2.2 Scheduling

Das Alleinstellungsmerkmal vom OpenCCS ist, dass es vollständig planungsbasiertes Job-Scheduling betreibt, im Gegensatz zu den gängigen warteschlangenbasierten Systemen [28]. Bei planungsbasierten Workloadmanagern bekommt jeder Job bzw. jede Ressourcenanfrage eine Startzeit zugewiesen [29]. Dies geschieht entweder explizit durch die BenutzerInnen oder wird implizit aus der Menge der benötigten und vorhandenen Ressourcen abgeleitet. Die Besonderheiten entstehen daraus, dass der Job-Scheduler nicht nur den momentanen Zustand betrachtet, sondern auch zukünftige Auslastungen in einem Zeitplan speichert, der nachträglich erweitert werden kann.

So sind im Falle des OpenCCS flexible Anfragen auf Ressourcen möglich, wie z.B. „bis zu 32 CPUs“ oder „mindestens 128 GB Arbeitsspeicher“. Diese skalierbaren Anfragen können zu höherer Auslastung des Clusters führen, da kleine und somit schwer nutzbare Reste von Ressourcen auf größeren Aufgaben verteilt werden können. Die Jobs, die nun weitere Ressourcen erhalten haben, werden tendenziell schneller ihr Ergebnis errechnen, womit dann wieder Platz für größere Aufgaben geschaffen ist. Auch können individuelle Deadlines einfach realisiert werden. Im Gegensatz dazu braucht es bei warteschlangenbasierten Systemen eine globale Maximalzeit für Jobs, damit Ressourcen nicht endlos durch fehlerhafte Jobs belegt werden.

Ein Nachteil des planungsbasierten Job-Schedulings ist die Notwendigkeit, eine geschätzte Laufzeit mitangeben zu müssen. Ist diese zu kurz, werden Jobs möglicherweise kurz vor Ende gestoppt, was Ressourcen verschwendet, da die Berechnung eventuell zu keinem endgültigen Ergebnis geführt hat. Ist die angegebene Laufzeit allerdings zu lang, dann werden Ressourcen entweder unnötig lange belegt, obwohl sie ungenutzt sind, oder ein neuer Zeitplan muss errechnet werden. BenutzerInnen neigen dazu im Mittel die Laufzeit zwei- oder dreifach so hoch einzuschätzen, als der Job eigentlich bräuchte, aufgrund des Risikos, die eine zu kurze Laufzeit bietet [30]. Etwa ein Zehntel aller NutzerInnen überschätzt die Laufzeit sogar um das Zehnfache oder mehr und mehr als ein Zehntel unterschätzt die Laufzeit, sodass es zu Jobabbrüchen kommt. [31]

Warteschlangenbasierte Systeme andererseits haben meist mehrere Schlangen unterschiedlicher Maximallaufzeiten, um den Zugriff von Jobs auf Ressourcen zu regeln. Nachdem eine Berechnung vollendet ist, wird geprüft, ob ein neuer Job die frei gewordenen Ressourcen nutzen kann. Zur Anwendung kommen übliche Scheduling- und teils auch Backfilling-Algorithmen. Eine Obergrenze für die Ausführungszeit von Jobs ist notwendig, wenn man das Risiko von permanent verbrauchten Ressourcen durch fehlerhafte oder extrem lange Jobs umgehen will [20].

2.3 Aufbau und Architektur

Das OpenCCS besteht aus vielen miteinander verknüpften Komponenten, die zum Teil wenig mit der vorliegenden Arbeit zu tun haben. In der folgenden Darstellung werden deshalb zwei davon weggelassen und einige andere unter einer einzelnen Kategorie zusammengefasst. Für das Verständnis sind die folgenden fünf Komponenten entscheidend [17] [20].

UI — User Interface Die Bedienoberfläche des Programms, also die Kommandozeile oder die grafische Oberfläche.

AM — Access Manager Dies ist der Authorisierungsservice, der Zugriffsrechte verwaltet und die Eingaben der BenutzerInnen weitergibt.

PM — Planning Manager Hier werden die Ressourcenanfragen geplant und auf das Cluster abgebildet.

MM — Machine Manager Der Knotenmanager stellt maschinenspezifische Features bereit wie z.B. Knotenmanagement oder Partitionierung und kommuniziert direkt mit der Hardware.

JM — Job Management Unter dieser Kategorie wurden alle Services zusammengefasst, die bei der Erstellung eines Jobs einzig für diesen initialisiert werden.

In dem folgenden Diagramm sind diese fünf Komponenten mit ihren Beziehungen untereinander dargestellt.

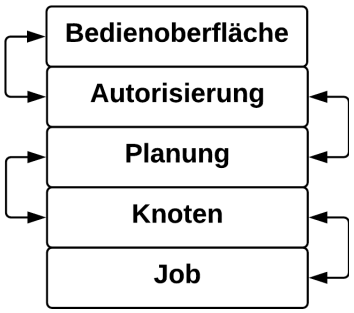

Abbildung 1: Vereinfachtes Diagramm der OpenCCS-Architektur

Eingaben werden von der Bedienoberfläche aus durch die Ebenen nach unten gereicht und das Ergebnis eines Jobs bewegt sich in umgekehrter Richtung. Diese Grafik wird später genutzt, um den Eingriffspunkt der geplanten Schnittstelle zu illustrieren.

2.4 Status der Dokumentation

Auf der offiziellen Webseite des Projekts [15] befinden sich einige hundert Seiten Dokumentation, aufgeteilt auf ein etwa 150 Seiten starkes „User Manual“ [32] und 230 zusätzliche Seiten „Administrator Manual“ [17]. Die enthaltenen Informationen sind zum Teil einige Jahre alt, was sich z.B. am Copyright einiger Code-Beispiele aus dem Jahr 2012 ablesen lässt, das Deckblatt ist jeweils datiert auf den 5. Oktober 2017.

Auf der Webseite befindet sich ein kurzes FAQ mit weniger als zehn Einträgen. Eine Wikiseite, der Projektplan, ein Ticket-System und ein Journal sind auf der offiziellen Webseite vorhanden, jedoch sind alle ungenutzt [15]. Der Ordner für Anleitungsseiten der Installation („Man-Pages“) ist leer.

Anleitungsseiten im html-Format wurden zusätzlich mit dem Code des Projekts und weiteren kurze Anweisungen per E-Mail zugesendet. Über die Suche mit einer herkömmlichen Suchmaschine wurden im Internet zwei Broschüren gefunden, die Konfiguration und Bedienung des OCuLUS-Clusters für Studierende der Universität Paderborn erklären [21] [5] und auch für das OpenCCS allgemein hilfreiche Informationen bieten.

Im Zuge der Einarbeitung in das OpenCCS-Projekt lag der Fokus besonders auf der Installationsanleitung. Diese ist über mehrere Dokumente und jeweils darin über mehrere Kapitel verteilt, außerdem fehlten anschauliche Beispiele. Benötigte Pakete und Bibliotheken sind unvollständig aufgelistet.

Weiter wurde die Orientierung in der Dokumentation dadurch erschwert, dass Dateinamen und -pfade mit speziellen Variablen gekennzeichnet sind,

ohne aufschlüsselnde Tabelle. Die Namen der tatsächlichen Dateien haben oft keine Gemeinsamkeiten mit den Variablennamen, was zu Verwirrung und erschwertem Fehlerfinden führt. Im Administrationshandbuch heißt es, dass man sich bei der Benennung an übliche Muster hielte, allerdings wird z.B. für die Benennung von Variablen sowohl CamelCase als auch snake_case verwendet.

2.5 Status des Codes

Die Codebasis wurde direkt vom momentanen Betreuer des Projekts Axel Keller [18] per E-Mail bereitgestellt. Zusätzlich zum neuesten Code des Projekts war auch ein Ordner mit zusätzlichen Hinweisen und Man-Pages enthalten. Normalerweise benutzt das Projekt Apache Subversion (svn), um die aktuelle Version des Codes auf der Projektwebsite bereitzustellen, diese enthält auch eine kurze Anleitung hierfür [15]. Kritikpunkte sind ein veraltetes und selbst unterschriebenes SSL-Zertifikat, was den Download über Subversion erschwert und darüber hinaus ein Sicherheitsproblem darstellt.

Bei dem Teil des Codes, der für die vorliegende Arbeit erweitert werden sollte, handelte sich vor allem um C-Code mit einigen Aufrufen über Perl-Skripte, das restliche Projekt enthält auch C++-Code.

2.5.1 Fehler in der Codebasis

Entgegen der Erwartungen war das Projekt in bereitgestellter Form durch einige Fehler im Code nicht lauffähig. Es gab sowohl Programmierfehler, als auch schwerwiegendere strukturelle Unstimmigkeiten. Dies führte noch vor dem erfolgreichen Kompilieren zu Problemen. Ein offensichtlicher Fehler, ein einzelnes nicht geschlossenes Kommentar, legt die Vermutung nahe, dass die momentane Version des OpenCCS zuvor nicht getestet wurde.

2.5.2 Undokumentierte Bibliotheken

Die für das Kompilieren benötigten Pakete waren schlecht dokumentiert und teils schwer zu finden. Es gibt eine Menge solcher undokumentierter Pakete und Bibliotheken, die installiert werden mussten. Um welche Pakete es sich im speziellen handelt kann aus dem Dockerfile im Anhang entnommen werden. Meist wurde das Fehlen nur aus den Fehlern beim Kompilieren abgeleitet, da die Liste in der Installationsanleitung unzureichend ist. Was die Installation angeht, erweckt die Dokumentation einen sehr veralteten und unvollständigen Eindruck, zusätzlich sind Kapitelsprünge verwirrend und nicht bedienungsfreundlich [17].

2.5.3 Alter des Quellcodes

Die Codebasis ist an manchen Stellen alt und wurde von verschiedenen Personen entwickelt, teils für Universitätsprojekte und -abschlussarbeiten [19]. Das führt dazu, dass einige vom OpenCCS benutzte Bibliotheken nun veraltet sind. Damit sind diese nicht mehr ohne Probleme aus einem offiziellen Repository herunterzuladen, weil sie schon vor längerem durch neuere Versionen ersetzt wurden. Dies hatte einen nicht unerheblichen Suchaufwand zur Folge und der Code musste vor dem Kompilieren an mehreren Stellen angepasst werden, da wenige moderne Versionen von Bibliotheken komplett inkompatibel waren mit der Codebasis und einige Dependencies waren nicht mehr in der benötigten, veralteten Form auffindbar. Zusätzlich wurden Paketnamen im Laufe der Jahre umbenannt, was die Suche noch erschwerte. Diese Probleme wurden vor Beginn der eigentlichen Arbeit über die Erstellung einer geeigneten Entwicklungsumgebung gelöst, worauf im Abschnitt (3.1) näher eingegangen wird.

Die moderne Version des GUI-Frameworks Qt 5 wird nicht unterstützt, sondern das alte Qt 4, was dazu führt, dass die Software nicht ohne Anpassungen auf Systemen laufen kann, die das neue Framwork benutzen, das von vielen Programmen gefordert wird. Es gab auch Versionsprobleme mit dem Tokenizer `flexer.l` in dem für den Inputstream zuständigen Pointer `yyin`, da die genutzten `istreams` in der neuen Bibliothek anders gehandhabt werden. In allen anderen Fällen konnte ein Konflikt beim Kompilieren durch die Benutzung älterer Pakete umgangen werden, der uneingeschränkte Betrieb der Software ist aber fraglich und bedarf einigen Tests.

2.5.4 Fehlerausgaben

Die Fehlerausgaben und -logs waren, wenn vorhanden, häufig schwer zu interpretieren, denn es gab kein einheitliches Schema und Fehlermeldungen wurden oft durch viele Dateien gereicht, bevor sie verarbeitet wurden. Diese Fehler sind auch in keiner öffentlich zugänglichen Stelle dokumentiert, was vermutlich ein Ergebnis der Diversität und Anzahl der beteiligten ProgrammiererInnen ist.

2.5.5 Konfiguration und Installation

Auch das Konfigurieren und anschließende Installieren der Software kostet einiges an Zeit. Während der Installation werden im Code feste Pfade abgesucht, in denen Konfigurationsdateien in bestimmter Form liegen müssen, damit OpenCCS diese findet und anwenden kann. Welche Konfigurationsdateien benötigt werden erschließt sich nicht intuitiv, außerdem gibt es bei den meisten Dateien und Werten keine Standardvariablen, die bei schlechter Konfiguration trotzdem zu einer einfachen Entwicklungsumgebung führen

könnten. Hier waren einige Versuche und das Interpretieren der Fehlerlogs nötig.

Ähnlich wie beim Kompilieren, führt die Konfigurationsanleitung durch das komplette Administartorenhandbuch und ist dabei unvollständig und unübersichtlich. Auffallend sind überraschend viele überholte bzw. deprecated Optionen, die während der Konfiguration nicht benutzt werden sollten.

Alleine für die Ressourcenbeschreibung des Clusters über die RSD-Konfiguration („Resource and Service Description“) gab es zehn Seiten Beschreibung für etwa 30 Optionen inkl. Listen und Verschachtelungen. Zum einfacheren Verständnis sollte ein siebenseitiges komplexes Beispiel vom RSD des OCuLUS-Clusters verhelfen. Auf dieses im Anhang befindliche Beispiel wurde nicht verwiesen.

2.5.6 Verbesserungsvorschläge

Die Dokumentation benötigt an einigen Stellen eine Überarbeitung, besonders bei den Anleitungen zur Installation und Konfiguration. Vorteilhaft wäre es, wenn zusammengehörende Abschnitte nahe beisammen stünden oder zumindest aufeinander verweisen würden. Unkomplizierte Beispiele wären außerdem eine große Hilfe.

Die Organisation der Projektseite ist nicht optimal, denn sie enthält eine Reihe ungenutzter Dokumentationsmöglichkeiten, z.B. Ticketsystem oder Wiki. Zusätzlich erfordert die Versionsverwaltung über Subversion ein aktuelles SSL-Zertifikat, das nicht vorhanden ist. Diese Probleme ließen sich lösen, wenn sich Projekt für eine Git-basierte Lösung entscheiden würde, wie GitHub oder GitLab. Das Wiki, die Handbücher usw. könnten auch dahin umziehen, das bündelt die Dokumentation und macht sie anpassbar für alle, nicht nur für einzelne Mitglieder des Projekts. Das verteilte Arbeiten am Code und das Abzweigen neuer Entwicklungszweige, z.B. für Studierendenprojekte, wird vereinfacht, zudem ist Git in den meisten Fällen performanter als Subversion [33] [34].

3 Implementierung der Schnittstelle

Die eigentliche Aufgabe dieser Arbeit ist die Implementierung einer neuen Allozierungsfunktion, die eine alternative Ressourcenanfrage über ein Programmablaufmodell verarbeiten kann. Zum Zeitpunkt der Aufgabe lagen drei andere Allozierungsmöglichkeiten vor: über die Anzahl der Knoten (`--nodes`), die Anzahl der Prozessoren (`--cores`) oder eine spezielle Ressourcenbeschreibung über die Kommandozeile (`--res`). An letzterer orientiert sich die Implementierung.

In diesem Abschnitt wird zunächst die Vorbereitung der Codebasis und die Schaffung der Entwicklungsumgebung beschrieben. Danach werden die Details der Implementierung der neuen Schnittstelle aufgeführt.

3.1 Entwicklungsumgebung

Da der Code unerwarteterweise in der ausgelieferten Form nicht ausführbar war, war vor der Implementierung der erwünschten Funktionalitäten die Schaffung einer Entwicklungsumgebung nötig. Dies ist nicht trivial, da die vorliegende Codebasis für ein ganzes Computer-Cluster gedacht ist und nicht einen Heimrechner. Die einzufügende Schnittstelle soll in das bestehende Projekt integriert werden, das vor und nach Implementierung lauffähig sein sollte. Um das umfangreiche Projekt kompilieren zu können, mussten einige Probleme behoben werden.

3.1.1 Designentscheidungen

Die direkte Einrichtung auf einem laufenden modernen System kam aus oben genannten Gründen nicht in Frage. Außerdem bietet eine möglichst flexible Entwicklungsumgebung Vorteile, z.B. sind Konflikte mit inkompatiblen Paketen unwahrscheinlicher und leichter zu beheben und Fehler können auch in den schlimmsten Fällen keinen ernsthaften Schaden verursachen.

Die wichtigste Designentscheidung war die Frage, wo die Erweiterung eigebettet werden sollte. Wie oben erwähnt, orientiert sich diese an einer verwandten Art der Ressourcenspezifikation und macht wie diese Gebrauch von Variablen und Funktionen aus Datei `alloc.c`. Es ist deshalb eine einfache Wahl, die Schnittstelle ebenso in diese Stelle einzupflegen, wie die anderen Möglichkeiten zur Ressourcenspezifikation auch. Die grundlegende Architektur des Projektcodes bleibt dabei unberührt, da keine neuen Dateien erstellt werden, sondern nur bestehende ergänzt. Da die Schnittstelle eine ähnliche Aufgabe hat, wie einige schon vorhandene Funktionen und zugleich deren Funktionen und Variablen mitnutzen kann, empfiehlt sich die Einführung eines neuen Moduls nicht, obwohl die Auslagerung eine einfachere Anpassung für andere Programmablaufmodelle ermöglichen würde. Jedoch verletzt dies die Architektur und könnte dadurch zu Verwirrung führen, weil ähnliche Methoden an unterschiedlichen Orten definiert würden.

Die neue Methode könnte, statt eigene Funktionalität zu beinhalten, einfach eine andere Allozierungsfunktion aufrufen. Aber auch das würde die bestehende Architektur verletzten, die anderen Allozierungsfunktionen müssten dann allerdings nicht angepasst werden. Es wurde sich gegen diese Möglichkeit entschieden.

Als Übergabeparameter soll der Dateiname eines Programmablaufmodells festgelegt werden, nach welchem dann in einem speziellen Ordner gesucht

wird. Um auf die Angabe eines Namens und die Festlegung auf einen einzelnen Ordner zu verzichten, könnte man auch eine Funktion implementieren, die z.B. das komplette Projektverzeichnis nach einer bestimmten Endung oder einem bestimmten Inhalt absucht, um das Programmablaufmodell zu finden. Diese Performancekosten rechtfertigen jedoch nicht die etwas erleichterte Bedienung der ansonsten leicht berechenbaren Funktion.

Den BenutzerInnen wird eine zusätzliche Möglichkeit des Gebrauchs der Software angeboten, deshalb ergänzt die Schnittstelle natürlicherweise die Bedienungsoberfläche, in diesem Fall die Möglichkeiten der Kommandozeile. Im folgenden Diagramm kann man erkennen, dass die Erweiterung in die Komponente „Bedienoberfläche“ einbettet wird. Auf diese Ebene wird direkt zugegriffen, um Ressourcenanfragen zu stellen.

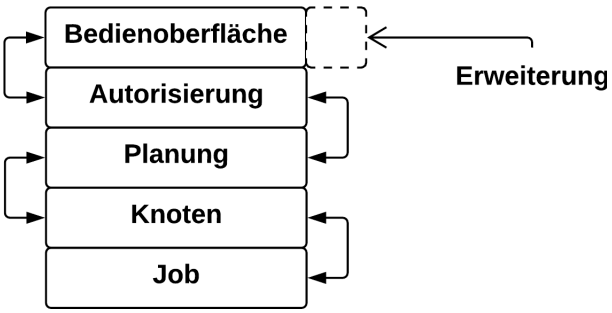

Abbildung 2: Ansatzpunkt der Schnittstelle

Während der Implementierung der Schnittstelle wurden an den einigen Berührpunkten mit der ursprünglichen Software nicht-kritische Erweiterungen der Fehlerbehandlung eingeführt.

Für die Umsetzung der Entwicklungsumgebung wurden drei Möglichkeiten abgewägt:

- Die **direkte Installation** auf einer eigens eingerichteten veralteten Betriebssystemversion mit der Verknüpfung an Archive für Legacy-Pakete. Das Einrichten einer neuen Maschine ist allerdings aufwendig, schwer zu präsentieren und fast unmöglich zu teilen.
- Eine **virtuelle Maschine**. Fehler z.B. beim Kompilieren, Besorgen der Pakete oder der Konfiguration können vorherige Arbeitsschritte zunichtezumachen. Das Sichern von Zuständen kann viel Zeit in Anspruch nehmen. Virtuelle Maschine bieten im Vergleich zu den anderen beiden Möglichkeiten Performance-Einbußen.

- Die Erstellung eines **Docker**-Containers. Aufgrund der Schichtenstruktur von Docker werden erfolgreiche Aktionen automatisch gespeichert und ein Datenverlust ist nicht möglich. Über das Dockerfile behält man einen Überblick über vergangene Arbeitsschritte. Weiter unten im Text wird auf diese Technologie weiter eingegangen.

Es wurde sich aus den oben genannten Gründen für die Verwendung eines Docker-Containers entschieden. Bestehende Erfahrung und die einfache Benutzung, Anpassung und Vervielfältigung für spätere Anwendungen beeinflussten die Auswahl zusätzlich.

3.1.2 Docker

Docker ist eine Virtualisierungstechnologie ohne den bei virtuellen Maschinen sonst üblichen Overhead, was über besondere Features des Linux-Kernels erreicht wird. Eine Betriebssysteminstanz wird nach einer speziellen Anleitung, dem Dockerfile, initalisiert und danach gestartet, wobei diese Instanz isoliert und parallel zum eigentlichen Betriebssystems des Hostrechners läuft. Da ein Docker-Container ein komplettes Betriebssystem enthält, ist es weitgehend unabhängig vom Hostsystem und kann flexibel auf verschiedenen Maschinen mit gleichem Resultat gestartet werden [35].

Im vorliegenden Fall kann mithilfe von Docker die vollständig konfigurierte Entwicklungsumgebung überall errichtet werden, das Docker-Image eignet sich allerdings nicht dazu, auf den Knoten eines Clusters zur Bildung eines Produktivsystems verteilt zu werden. Einige dafür notwendige Optionen wurden deaktiviert.

Es liegen zusätzlich noch zwei bash-Skripte vor, um das Image zu bauen und dann mit dem daraus entstehenden Container eine Entwicklungsumgebung zu starten.

Listing 1: build

```
1 #!/bin/bash
2 docker build \
3 -t openccs \
4 -f Dockerfile.openccs .
```

Das Bau-Skript sucht nach der Anleitungsdatei `Dockerfile.openccs`, um aus diesem ein instanziierbares Images zu bauen und es mit dem Namen `openccs` zu versehen. Dieses Skript muss nur aufgerufen werden, nachdem am Dockerfile, oder an Dateien, die in das Image kopiert werden Änderungen vorgenommen wurden oder die Entwicklungsumgebung zum ersten Mal auf dem jeweiligen Hostsystem gestartet werden soll.

Listing 2: start-node

```
#!/bin/bash
docker run \
-h oculus \
--name openccs-node \
-it --rm \
openccs
```

Dieses Start-Skript öffnet eine interaktive Kommandozeile auf der durch Docker bereitgestellten Betriebssysteminstanz, setzt den Hostnamen auf `oculus`, benennt den Container zu `openccs-node` um und legt fest, dass der Arbeitsspeicher bereinigt werden soll, falls der Container geschlossen wird. Dieses Skript sollte immer dann gestartet werden, wenn man die Entwicklungsumgebung nutzen möchte.

Die Entwicklungsumgebung dient dem Testen der vom OpenCCS bereitgestellten Funktionalität ohne das tatsächliche Ausführen eines Jobs. Es kann so getestet werden, ob neu implementierte Funktionen das korrekte Verhalten zeigen. Durch Docker kann die Entwicklungsumgebung auf jedem 64bit-Linux-System ausgeführt und benutzt werden.

3.2 Implementierung

Mit der eingerichteten Entwicklungsumgebung konnte dann die eigentliche Erweiterung implementiert werden. Eine zusätzliche Funktion nach Vorbild der bestehenden Allozierungsfunktionen wurde eingebunden. Wie beim restlichen Code des Projekts sind die Variablennamen und Kommentare im Quellcode in englischer Sprache gehalten.

Für spezielle Anwendungsfälle benötigte Schemata von Programmablaufmodellen können problemlos integriert werden. Hierzu sollte man den über Kommentare auffällig markierten Block in `alloc.c` ersetzen. Zu Testzwecken befindet sich nun dort als Freihalter Code zum Parsen eines sehr vereinfachten Programmablaufmodells. Das eigentliche Ausführen und Schedulen passiert wie bei der Angabe der Ressourcen über die Kommandozeile. Es folgt also dem normalen Ablauf, den das OpenCCS bereitstellt.

Die neue Methode liest die Ressourcen aus einer Datei ein. Der Dateiname wird als Parameter für das neu eingeführte Argument `-p` oder `--program-behaviour` übergeben, wonach im Pfad `$CCS/etc/$CCS_ISLAND` gesucht wird. Im vorliegenden Beispielsfall schlüsselt sich dieser auf als `/usr/local/openccs/etc/HAWAII`, wo auch ein Testmodell mit dem Namen `tiapbm`, kurz für „this is a program behaviour model“, liegt. Die Implementierung orientiert sich stark an der `--res`-Funktion, nur liest es die

angefragten Ressourcen aus einer Datei statt von der Kommandozeile, aber das Ergebnis und die Ausgabe der beiden Funktionen sind gleich. Das Beispielmodell enthält das `--res`-Beispiel aus der Dokumentationsfunktion. Zusätzlich wird das eingelesene Programmablaufmodell auf der Kommandozeile ausgegeben, um das korrekte Einlesen zu bestätigen. Dies ist im Quellcode offensichtlich markiert und könnte im Produktivsystem entfernt werden.

Die Funktionalität wurde wie alle anderen Allozierungsfunktionen in `alloc.c` eingeflochten. Die Erweiterung sitzt somit also an der gleichen Stelle wie die anderen Möglichkeiten zur Ressourcenspezifikation.

Die neu implementierte Funktion ist wie alle anderen vom User zu benutzenden Funktionen über -h oder –usage dokumentiert. Die dafür nötigen Ergänzungen der Datei options.h sind die folgenden:

Listing 3: options.h

```
1 #define PROG_BEHAVIOUR_S   "p"
2 #define PROG_BEHAVIOUR_L   "program-behaviour"
3 #define DEF_ARG_PROG_BEHAVIOUR \
4    arg_str0(PROG_BEHAVIOUR_S, PROG_BEHAVIOUR_L, \
5    "<program behaviour file>", \
6    "Uses a program behaviour file for resource calculation.\n" \
7    " The file should be in $CCS/etc/$CCS_ISLAND. "  \
8    )
```

In der ersten und zweiten Zeile werden die Argumente zum Aufruf der implementierten Funktion definiert und in den Zeilen 5-7 die Beschreibung, die durch die Hilfsfunktion ausgegeben wird.

Bei der Implementierung der Fehlerausgabe wurde sich auch am bestehenden Format der veränderten Datei orientiert. Es wird immer dann ein Fehler geworfen, wenn sich an dem abgesuchten Ort kein gültiges Programmablaufmodell mit dem als Parameter übergebenen Namen befindet. Als Hilfe wird neben der eigentlichen Meldung auch der abgesuchte Pfad ausgegeben.

Listing 4: Fehlermeldung Pfad

```
1 pc2Error("Can't find a program behaviour model at this place:%s
2   \nProgram behaviour models should be located at %s\n\n", path, dir);
```

Die Fehlerbehandlung beim Parsen der Ressourcenanforderungen wird über die schon vorhandenen Methoden gemacht. Die anderen Allozierungsfunktionen benutzen diese auch.

Listing 5: Fehlermeldung Spezifikation

```
pc2Syntax (1, RES_L, "Error in resource description'%s': %q",(char *) line);
```

Zusätzlich wird nun an einigen Stellen über `strerror` der `errno`-Fehlerinhalt ausgegeben, was unaussagekräftige Fehlermeldungen ergänzt. Für das Testen der neuen Funktion wurden verständlicherere Fehler benötigt und da dies auch für den normalen Betrieb von Vorteil sein kann, wurde diese Ergänzung nicht entfernt.

Die anderen drei Allozierungsfunktionen wurden im Zuge der Implementierung auch minimal angepasst. Dies geschah ausschließlich ganz am Anfang der jeweiligen Funktion, da dort über `if`-Abfragen eine korrekte Bedienung erzwungen wird, ohne mehrfache Spezifikation der Ressourcen. Die Anpassung beschränkte sich auf das Hinzufügen einer einzelnen Anfrage pro Allozierungsfunktion.

Listing 6: Ausschluss redundanter Anfragen

```
if (MS_cores->count > 0)
  pc2Error("Do not know what to do:
    you used options '-%s' and '-%s'", PROG_BEHAVIOUR_S, CORES_S);
if (MS_nodes->count > 0)
  pc2Error("Do not know what to do:
    you used options '-%s' and '-%s'", PROG_BEHAVIOUR_S, NODES_S);
if (MS_res->count > 0)
  pc2Error("Do not know what to do:
    you used options '-%s' and '--%s'", PROG_BEHAVIOUR_S, RES_L);
```

Hinweis: `ccs_strtools.c` wirft bei der Ausführung aller Allozierungsfunktionen einen Fehler, aufgrund fehlender UI-Anbindung. Diese Anbindung wird von Docker nativ nicht unterstützt und birgt Sicherheitsrisiken, ist aber weder für Tests noch den Betrieb des OpenCCS notwendig. Die genaue Fehlerausgabe befindet sich im Anhang.

4 Evaluation der Ergebnisse

Nachfolgend wird die erstellte Schnittstelle anhand der sechs Merkmalen aus dem Abschnitt „Bewertungskriterien der Erweiterung“ geprüft. Zur besseren Übersicht sind die Kriterien hier noch einmal aufgelistet.

1. Ist es möglich Programmablaufmodelle zu parsen und dafür Ressourcen zu reservieren? Kann man alternative Programmablaufmodelle ergänzen?

2. Ist der Code lesbar und ausreichend kommentiert und dokumentiert?

3. Sind die Ausgaben, v.a. der Fehler, lesbar und zielführend? Wie robust wird mit Argumenten umgegangen?

4. Wie viele neue Dateien, globale Variablen und Funktionen wurden erstellt oder verändert? Gibt es dabei Konfliktpotential?

5. Wurden neue Bibliotheken genutzt? Ergeben sich daraus neue Abhängigkeiten?

6. Wie vergleicht sich die Erweiterung mit dem schon Vorhandenen?

Hier stehen nun die Einschätzungen über die Schnittstelle nach den obigen Kriterien.

1. Die Schnittstelle kann Programmablaufmodelle parsen und ist für die Verarbeitung anderer Schemata problemlos erweiterbar. Die Reservierung von Ressourcen funktioniert genauso wie bei den anderen Arten der Ressourcenanfragen.

2. Der Code umfasst nur einige Dutzend Zeilen, wovon beinahe jede zum leichteren Verständnis kommentiert wurde. Auch eingeführte Variablen werden im Code dokumentiert. Beim Betrieb des erweiterten OpenCCS lässt sich auf die übliche Weise eine Hilfe zuer Bedienung der Schnittstelle ausgeben.

3. Wird das Ablaufmodell an der gesuchten Stelle nicht gefunden wurde, wird zusätzlich zur Fehlermeldung ein Hinweis ausgegeben. Dadurch lässt sich erkennen, wo die Schnittstelle das Modell erwartet und unter welchem Dateinamen. Die Anfrage kann daraufhin entsprechend verändert werden, um das gewünschte Verhalten zu erzielen. Ist in der eingelesenen Datei kein gültiges Programmblaufmodell vorhanden, wird dies beim Parsen der Ressourcenanforderungen bemerkt. Dort wird der Fehler über die gleiche Fehlerfunktion behandelt wie bei den anderen Allozierungsfunktionen. Diese Fehlerausgabe wurde zusätzlich ergänzt.

4. In `alloc.c` wurde genau eine neue globale Methode eingeführt, die die gewünschte Schnittstelle umsetzt und in `options.h` wurden die nötigen Variablen für die Hilfsfunktion gesetzt. Diese Struktur ist identisch mit denen schon vorhandener Methoden, es gibt dort kein Konfliktpotential. Programmablaufmodelle werden jeweils in einer eigenen Datei beschrieben, abgesehen davon wurden aber keine neuen Dateien eingeführt.

5. Für das Einlesen von Dateininhalten wurde zusätzlich die C-Standardbibliothek `stdio.h` eingebunden, die mit jeder C-Installation ausgeliefert wird. Das OpenCCS-Projekt benutzt diese Bibliothek selbst an anderen Stellen, deshalb sind keine neuen Abhängigkeiten entstanden.

6. Die Integration der neuen Schnittstellen orientiert sich an ähnlichen Funktionen. Sie basiert vor allem auf der `--res`-Funktion, so ist auch das Parsen von Ressourcen aus dieser übernommen worden. Individuelle Funktionalitäten zum Parsen alternativer Programmablaufmodelle können problemlos integriert werden. Die hierfür anzupassenden Stellen sind im Code markiert.

Hinweis: Das Dockerfile könnte vielleicht verkürzt werden, wenn man sich mehr mit den benötigten Paketen beschäftigt. Es wurde nicht darauf geachtet, ob unnötigerweise Paketsammlungen installiert werden, obwohl einzelne Pakete daraus reichen würden. So etwas vergrößert das Docker-Image, was das Bauen und Verteilen verlangsamen kann.

Insgesamt war die Implementierung erfolgreich und hat zu einer lauffähigen Schnittstelle geführt. Die vorhandene Bedienoberfläche wurde hierzu erweitert und verfügt nun über eine weitere Möglichkeit zur Ressourcenspezifikation. Durch eine vorsichtige Einbettung wurden offensichtliche Konflikte vermieden und die vorherige Architektur beibehalten. Bestehender Code wurden nur an den unbedingt nötigen Berührpunkten angepasst und alte Funktionalitäten können genauso wie zuvor benutzt werden.

5 Ausblick

Mit dieser Arbeit wird eine Schnittstelle bereitgestellt, um eine alternative Ressourcenanfrage über das OpenCCS an ein Cluster-System zu stellen. Dies schafft die Option genauere Anfragen und auch solche mit dynamischem Ressourcengebrauch zu stellen. Die Ziele dabei sind sowohl das Erreichen einer höheren Auslastung mit positiven Effekten auf alle Jobs, als auch das engere Einbinden vom Bedarf der NutzerInnen. Kaum ein Job benötigt alle verfügbaren Ressourcen über die komplette Laufzeit und dies soll über Programmablaufmodelle, für eine effiziente Verarbeitung virtueller Ressourcen, in das Scheduling miteinbezogen werden. Weitere Forschung kann das OpenCCS und ähnliche Systeme ausbauen, um moderne und produktivere Cluster-Systeme zu realisieren.

Fragen stellen sich noch nach der Repräsentation, der Erstellung und dem Parsen von effektiven Programmablaufmodellen, die genügend Informationen zum verbesserten Scheduling enthalten, aber gleichzeitig auch zuverlässig für größere Jobs angefertigt werden können. Für spezifische Anwendungsfälle

sind verschiedenste Schemata denkbar, die letztlich alle eine strukturierte Ansammlung von Beschreibungen benötigter Ressourcen sind.

Abschließend lässt sich sagen, dass heutzutage und in naher Zukunft Clustersysteme den wichtigsten Teil des HPC-Bereichs ausmachen [36] [37]. Vorstellbare Optimierungen und eine steigende Zahl an Anforderungen aus unterschiedlichen Anwendungsdomänen von Personen auch ohne Informatik-Hintergrund, wie z.B. Chemie, Genetik oder Astronomie [36], machen es erforderlich Jobs so flexibel und damit effizient wie möglich bearbeiten zu können. Der Fokus sollte hierbei auch auf der Benutzbarkeit liegen und möglichst bedienungsfreundlich auf unterschiedlichste Aufträge einzugehen und diese gemeinsam auf einem Cluster-System verarbeiten. Gute Auslastungsquoten aller Ressourcen und die Öffnung des Cluster-Computings für neue Anwendungsbereiche ebnen den Weg für das HPC der Zukunft.

6 Literatur

[1] "Top500 List - November 2017," https://www.top500.org/list/2017/11/, Überprüft am: 11.02.2018.

[2] M. Baker, "Cluster computing white paper," 2000.

[3] M. Butts, "Synchronization through communication in a massively parallel processor array," 2007.

[4] D. A. Bader and R. Pennington, "Cluster computing: Applications," 2001.

[5] Paderborn Center for Parallel Computing, "HTC Brief Instructions," https://groups.uni-paderborn.de/pc2/documents/htc_brief_instructions.pdf, version: 19.12.2017.

[6] L.-O. Burchard, B. Linnert, F. Heine, M. Hovestadt, O. Kao, and A. Keller, "A Quality-of-Service Architecture for Future Grid Computing Applications," 2005.

[7] L.-O. Burchard, M. Hovestadt, O. Kao, and A. Keller, "The Virtual Resource Manager: An Architecture for SLA-aware Resource Management," 2004.

[8] W. Domschke, *Transport: Grundlagen, lineare Transport- und Umladeprobleme.* Oldenbourg, 2007.

[9] S. Dempe and H. Schreier, *Transportoptimierung.* Teubner, 2006.

[10] P. J. Denning, "The working set model for program behavior," 1968.

[11] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," 2001.

[12] J. P. Kitajima and B. Plateau, "Modelling parallel program behaviour in alpes," 1994.

[13] Y. H. Kang, W. J. Choi, B. C. Kim, and J. M. Kim, "On tradeoff between the two compromise factors in assigning tasks on a cluster computing," 2014.

[14] L. A. Schultheiss and E. M. Heiliger, "Techniques of flow-charting," 1963.

[15] "OpenCCS web site," https://www.openccs.eu/core/, Überprüft am: 11.02.2018.

[16] "GNU General Public License," https://www.gnu.org/licenses/gpl-3.0.en.html, Überprüft am: 26.02.2018.

[17] Paderborn Center for Parallel Computing, "OpenCCS Administrator Manual 0.9.8-2," https://groups.uni-paderborn.de/pc2/documents/openccs_admin_manual.pdf, Stand: 05.10.2018.

[18] "Axel Keller - Paderborn Center for Parallel Computing - Staff," https://pc2.uni-paderborn.de/about-pc2/staff-board/staff/axel-keller/, Überprüft am: 17.02.2018.

[19] A. Keller and A. Reinefeld, "Ccs resource management in networked hpc systems." IEEE, 1998.

[20] A. Keller, "A Data Structure for Planning Based Workload Management of Heterogeneous HPC Systems."

[21] Paderborn Center for Parallel Computing, "OCuLUS Brief Instructions," https://groups.uni-paderborn.de/pc2/documents/oculus_brief_instructions.pdf, version: 19.12.2017.

[22] ——, "OCuLUS Technical Description," http://pc2.uni-paderborn.de/hpc-services/available-systems/oculus/, Überprüft am: 11.02.2018.

[23] TOP500, "OCuLUS Technical Description," https://www.top500.org/system/178076, Stand: 06.2013.

[24] "SchedMD - Slurm Support and Development," https://schedmd.com/, Überprüft am: 14.02.2018.

[25] "Sunway TaihuLight - Software," http://www.nsccwx.cn/wxcyw/soft.php?word=soft&i=47, Überprüft am: 14.02.2018.

[26] Y. Gong, M. E. Pierce, G. C. Fox, E. Frachtenberg, and U. Schwiegelshohn, *Job Scheduling Strategies for Parallel Processing: 14th International Workshop, JSSPP 2009, Rome, Italy, May 29, 2009. Revised Papers.* Springer-Verlag Berlin Heidelberg, 2009.

[27] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," 2007.

[28] "Comparison of cluster software Wikipedia," https://en.wikipedia.org/wiki/Comparison_of_cluster_software, Überprüft am: 11.02.2018.

[29] M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in hpc resource management systems: Queuing vs. planning." Springer Berlin Heidelberg, 2003.

[30] A. Streit, "A self-tuning job scheduler family with dynamic policy switching." Springer-Verlag, 2002.

[31] D. Tsafrir, Y. Etsion, and D. G Feitelson, "Backfilling Using Runtime Predictions Rather Than User Estimates," 2018.

[32] Paderborn Center for Parallel Computing, "OpenCCS User Manual 0.9.8-2," https://groups.uni-paderborn.de/pc2/documents/openccs_user_manual.pdf, Stand: 05.10.2018.

[33] "Git versus svn popularity statistics," https://softwareengineering.stackexchange.com/questions/136079/are-there-any-statistics-that-show-the-popularity-of-git-versus-svn, Überprüft am: 20.02.2018.

[34] "Comparison of version control software," https://en.wikipedia.org/wiki/Comparison_of_version_control_software, Überprüft am: 20.02.2018.

[35] "What is docker?" https://www.docker.com/what-docker, Überprüft am: 27.02.2018.

[36] D. A. Reed and J. Dongarra, "Exascale Computing and Big Data," 2015.

[37] A. Geist and D. A. Reed, "A survey of high-performance computing scaling challenges," 2017.

7 Anhang

7.1 Dockerfile für OpenCCS und Dokumentation dazu

Oben im Text befinden sich Anweisungen, wie man den Container baut und startet. Der Container ist in erster Linie dazu gedacht, eine passende Entwicklungsumgebung für das OpenCCS-Projekt bereit zu stellen und es kann damit in der vorliegenden Version kein Produktivsystem erstellt werden. Das Dockerfile ist kommentiert.

Listing 7: Dockerfile

```
#### USE UBUNTU AS THE OS ####
FROM ubuntu:xenial

#### INSTALL ALL THE NEEDED DEPENDENCIES AND LIBRARIES ####
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
apt-get install --no-install-recommends -y --assume-yes \
#### THE FOLLOWING PACKAGES WILL BE INSTALLED ####
apt-utils \
autoconf \
automake \
bison \
bsd-mailx \
build-essential \
flex \
g++-5-multilib \
gcc-5-multilib \
gpp \
lib32ncurses5-dev \
libargtable2-dev \
libexplain-dev \
libeztrace0 \
libmotif-dev \
libpcp3-dev \
libqt4-dev \
libreadline-dev \
libstdc++-5-dev \
libtool \
libxext-dev \
libxpa-dev \
ncurses-base \
pcp \
perl \
perl-byacc \
perl-tk \
perlqt-dev \
qt4-dev-tools \
```

```
ssh \
subversion \
sudo \
tk-dev \
uuid-dev \
wget \
nano

#### COLLECT THE OPENCCS SOURCE CODE ####
#RUN svn --non-interactive --no-auth-cache --trust-server-cert co \
#https://openccs.eu/svn/openccs/trunk openccs-trunk
#ADD openccs-trunk /opt/openccs
ADD OpenCCS-0.9.8-1 /opt/openccs
WORKDIR /opt/openccs/src

#### SET THE ROOT PASSWORD (NEEDED FOR DEBUGGING) ####
RUN passwd -d root

#### CREATE THE USER CCS WITH PASSWORD AND SUDOER RIGHTS ####
RUN groupadd ccsadmin \
&& useradd --create-home --groups ccsadmin --shell /bin/bash ccs \
&& echo 'ccs:1234' | chpasswd \
&& adduser ccs sudo
ADD sudoers /etc/sudoers
RUN chmod 440 /etc/sudoers
USER ccs

#### INSTALL NEEDED PERL MODULES FOR THE USER CCS ####
RUN sudo apt-get install --no-install-recommends -y \
libmailtools-perl \
libproc-processtable-perl \
#### CONFIGURE OPENCCS ####
&& sudo libtoolize --force \
&& sudo aclocal \
&& sudo autoheader \
&& sudo automake --force-missing --add-missing \
&& sudo autoconf \
&& sudo /opt/openccs/src/configure \
--enable-ccs-account=ccs:ccs \
--enable-thread-support=yes \
--enable-develop-mode=yes

#### ADD CORRECTED FILES ####
ADD Makefile /opt/openccs/src/shared/evt/Makefile
ADD evt_com_xdr.c /opt/openccs/src/shared/evt/evt_com_xdr.c
#### COMPILE AND INSTALL OPENCCS ####
RUN sudo ln cis/om_api.h shared/evt \
&& sudo make \
&& sudo make install
```

```
87 ADD ccs_postinstall /opt/openccs/src/util
88 RUN sudo util/ccs_postinstall
89
90 #### SET THE NECECESSARY ENVIRONMENT OPTIONS ####
91 ENV CCS=/usr/local/openccs
92 ENV CCS_DIR="$CCS"
93 ENV CCS_ISLAND="HAWAII"
94 ENV CCS_UI_DEF_ISLAND="$CCS_ISLAND"
95 ENV CCS_UI_DEF_GROUP="ccsadmin"
96 ENV BIN_DIR="$CCS/bin"
97 ENV SBIN_DIR="$CCS/sbin"
98 ENV CCS_ARCH="LINUX64"
99 ENV SBIN_ARCH_DIR="$SBIN_DIR/$CCS_ARCH"
100
101 ENV MANPATH="$CCS/man:$MANPATH"
102 ENV PATH="$CCS/bin:$CCS/sbin:$CCS/bin/$CCS_ARCH:$PATH"
103 ENV LD_LIBRARY_PATH="$CCS/lib:$LD_LIBRARY_PATH"
104
105 #### ADJUST THE PERMISSIONS ####
106 RUN sudo chown root:ccs $SBIN_ARCH_DIR/ccs_sudo \
107 && sudo chmod 110 $SBIN_ARCH_DIR/ccs_sudo \
108 && sudo chmod u+s $SBIN_ARCH_DIR/ccs_sudo \
109 && sudo chmod -R 777 $CCS
110
111 #### COPY THE EXAMPLE CONFIGURATION FILES INTO THE CONTAINER ####
112 WORKDIR /usr/local/openccs
113 RUN mkdir -p etc/$CCS_ISLAND \
114 && sudo cp examples/* etc/$CCS_ISLAND
115
116 ADD resource.conf etc/$CCS_ISLAND
117 ADD nodefile-oculus.txt etc/$CCS_ISLAND
118 ADD aal.conf etc/$CCS_ISLAND
119 #### ADD AN EXAMPLE PROGRAM BEHAVIOUR MODEL FILE ####
120 ADD this_is_a_program_behaviour_model etc/$CCS_ISLAND/tiapbm
121
122 RUN sudo chmod -R 777 $CCS
123
124 ENV ISLAND_RESOURCE_FILE="$CCS/etc/$CCS_ISLAND/resource.conf"
125 ENV CCS_UI_DEF_ISLAND="$CCS_ISLAND"
126
127 RUN mkdir -p /usr/local/openccs/tmp/$CCS_ISLAND/AM \
128 && mkdir -p /usr/local/openccs/tmp/$CCS_ISLAND/PM
129
130 #### CONFIGURE WHAT HAPPENS WHEN THE CONTAINER STARTS ####
131 ADD bootstrap.sh .
132 RUN sudo chmod +x bootstrap.sh
133 ENTRYPOINT ["/bin/bash", "bootstrap.sh"]
```

7.2 Codeanpassungen

Hier folgt die Funktion, durch die die neue Schnittstelle in `alloc.c` implementiert wurde.

Listing 8: alloc.c

```
/* use program behaviour file */
if (MS_prog_behaviour->count > 0)
  {
    if (MS_cores->count > 0)
      pc2Error("Do not know what to do: you used options '-%s' and '-%s'", PROG_BEHAVIOUR_S, CORES_S);
    if (MS_nodes->count > 0)
      pc2Error("Do not know what to do: you used options '-%s' and '-%s'", PROG_BEHAVIOUR_S, NODES_S);
    if (MS_res->count > 0)
      pc2Error("Do not know what to do: you used options '-%s' and '--%s'", PROG_BEHAVIOUR_S, RES_L);

    FILE *prog_behaviour_model;  // handle for the program behaviour model file
    char *line = NULL;      // points to each individual line of the model after another
    size_t len = 0;         // the length of each individual line
    ssize_t read;           // error code (or EOF) for the getline method
    char path[128];         // path to the program behaviour model
    char dir[128];          // directory for program behaviour models; mostly used for error messages

    // construct the searched directory out of the environment variables
    sprintf(dir, "%s/etc/%s/", getenv("CCS"), getenv("CCS_ISLAND"));
    // add the user given file name to the path, to get the absolute path of the program behaviour model
    strcpy(path, dir);
    strcat(path, MS_prog_behaviour->sval[0]);
    // try to open the program behaviour model in read mode
    prog_behaviour_model = fopen(path, "r");
    // if this program could open the program behaviour model
    if (prog_behaviour_model != NULL)
      {
        /////// THE CODE IN THE FOLLOWING PART IS ONLY AN EXAMPLE ///////
        char msg_top[56];   // hold a Message which will be displayed on top of the output
        sprintf(msg_top, "\n###### PROGRAM BEHAVIOUR MODEL ######\n");
        printf("%s", msg_top);
        // while the end of the program behaviour model is not reached, inspect another line
        while ((read = getline(&line, &len, prog_behaviour_model)) != EOF)
          {
            // prints each read line to show a representation of the program behaviour model
            printf("%s", line);

            /////// THIS IS ONLY AN EXAMPLE RESOURCE PARSING ///////
            // copied from the resource description option
            MS_resDsc.rSet = ccs_rset_build (MS_resDsc.rSet, (char *) line);
            if (NULL == MS_resDsc.rSet)
              pc2Syntax (1, RES_L, "Error in resource description'%s': %q",(char *) line);
            ccs_str_append(&MS_resStr, " ", (char*) line);
            /////// END OF EXAMPLE RESOURCE PARSING ///////
          }
        // close the access to the program behaviour model file
        fclose(prog_behaviour_model);
        // if there is still a pointer to the memory (only in an error case)
        if(line)
          free(line);

        // print a line of # as long as the top message
        printf("%.*s\n\n", strlen(msg_top)-2, "##########################################################");

```

```
55          /////// END OF EXAMPLE CODE ///////
56        }
57      else
58        {
59          pc2Error("Can't find a program behaviour model at this place: %s
60              \nProgram behaviour models should be located at %s\n\n", path, dir);
61        }
62    }
```

7.3 Fehlerausgabe wegen fehlender UI-Anbindung

Listing 9: Fehlerausgabe UI-Anbindung

```
1 UI_2583f8c7369f_1459(23:35:36):[E]:(evt_com_clnt.c, 902):
2 Invalid argument(s) given:
3 Too few arguments given (thrown by 'ccs_strtools.c:398')
4
5 UI_2583f8c7369f_1459(23:35:36):[E]:(evt_com_clnt.c, 907):
6 Invalid argument(s) given:
7 Too few arguments given (thrown by 'ccs_strtools.c:398')
```

7.4 Installationstipp für OpenCCS

Eine Liste der benötigten Dependencies und Pakete für ein neues Ubuntu-System kann man dem Dockerfile entnehmen. Auch für eine beispielhafte Konfiguration kann der Rest des Dockerfiles als Ablaufplan herangezogen werden, falls grundsätzliche Docker-Kenntnisse vorhanden sind.