Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin
Arbeitsgruppe Softwareengineering

Planungsbasierte Zuordnung dynamisch erzeugter Prozesse mit MPI-2

Paul Nierenz
Matrikelnummer: 5315126
paul.nierenz@fu-berlin.de

Betreuer: Barry Linnert
Eingereicht bei: Barry Linnert
Zweitgutachter: Prof. Dr.-Ing. Jochen Schiller

Berlin, 4. Januar 2024

Zusammenfassung

Anwendungen mit besonders hohem Ressourcenbedarf, wie beispielsweise Wetter- und Klimasimulationen, erfordern leistungsfähige High-Performance-Computing-Systeme. Diese verfügen üblicherweise über ein Verwaltungssystem, welches den Anwendungen für einen bestimmten Zeitraum Ressourcen (CPUs, Speicher etc.) zur Verfügung stellt. Bei planungsbasiertem Ressourcenmanagement wird für jede Anwendung ein Plan erstellt, der vorgibt, welche Ressourcen in welchem Zeitraum von der Anwendung genutzt werden sollen. In dieser Arbeit wird untersucht, wie die Prozesse einer MPI-Anwendung bestimmten Knoten eines HPC-Systems gemäß eines Ausführungsplans zugewiesen werden können. Dazu werden die Abläufe beim Start einer MPI-Anwendung vorgestellt und ein Lösungsansatz präsentiert. Dieser wird in Form eines Plugins für Open MPI prototypisch implementiert.

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit handelt von der planungsbasierten Zuordnung von dynamisch erzeugten MPI-Prozessen auf HPC-Systemen. Bevor die konkrete Aufgabenstellung in Kapitel 2 erläutert wird, folgen zunächst ein paar einleitende Abschnitte zu den Themen High-Performance-Computing (HPC) und Ressourcenverwaltung.

1.1. HPC-Anwendungen

In vielen Bereichen des täglichen Lebens werden Computerprogramme zur Lösung von Problemen eingesetzt. Obwohl Computersysteme in den letzten Jahrzehnten viel kleiner und leistungsfähiger geworden sind, gibt es Anwendungen, die besonders hohe Anforderungen an die Hardware, auf der sie ausgeführt werden, stellen. Sie benötigen mehr Ressourcen, als ein einzelner Computer zur Verfügung stellen kann. Solche Anwendungen werden im Folgenden als *HPC-Anwendungen* bezeichnet.

HPC-Anwendungen werden beispielsweise für Genomforschung eingesetzt, für die Erstellung von Wettervorhersagen und Klimamodellen sowie für Crash-Test-Simulationen. Ein weiteres Beispiel ist die Simulation von Windgeschwindigkeiten im urbanen Umfeld [6], welche in Abbildung 1 veranschaulicht wird.

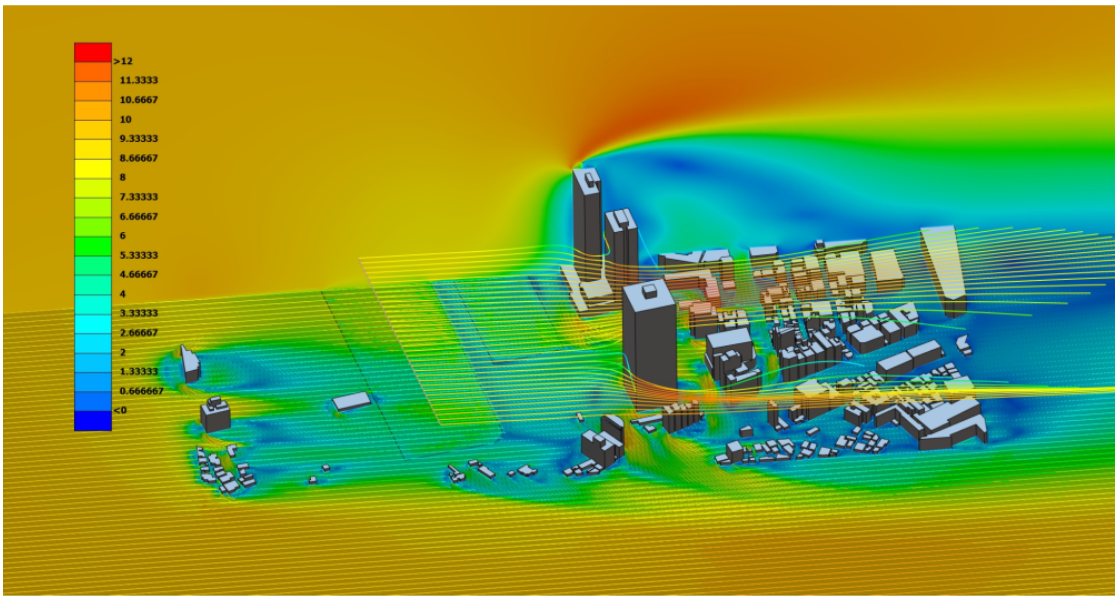

Abbildung 1: Beispiel einer HPC-Simulation zur Berechnung von dreidimensionalen Windgeschwindigkeiten zwischen Gebäuden [6]

Solche Simulationen ermöglichen es, Vorhersagen über die Zukunft zu treffen oder teure Produkte wie Rotorblätter von Windkraftanlagen zu simulieren, ohne sie dazu herstellen

zu müssen. Dadurch lassen sich potentiell Kosten und Zeit sparen.

1.2. HPC-Systeme und Grid Computing

HPC-Anwendungen benötigen zur Ausführung leistungsfähige Systeme. Hier kommen High-Performance-Computing-Systeme (HPC-Systeme) zum Einsatz. Diese bestehen aus vielen miteinander vernetzten Servern, die im weiteren Verlauf auch als *Knoten* bezeichnet werden. Ein Beispiel für ein HPC-System ist in Abbildung 2 dargestellt.

Abbildung 2: Das Lichtenberg II Cluster der TU Darmstadt [9]

Um die Rechenleistung mehrerer Knoten nutzen zu können, werden HPC-Anwendungen als parallele Programme implementiert. Zur Laufzeit werden dazu viele Prozesse erzeugt, die auf die Ressourcen des HPC-Systems verteilt werden. Durch die Parallelisierung kann die Rechenzeit der Anwendungen verkürzt werden.

Da die Anschaffung, der Betrieb und die Wartung von HPC-Systemen sehr kostenintensiv sind, lohnt sich ein eigenes System für einen Nutzer nur dann, wenn er es dauerhaft auslasten kann. Alternativ können mehrere Nutzer ihre Anwendungen auf einem gemeinsamen System oder auf einem System, das von einem externen Anbieter bereitgestellt wird, ausführen.

Für den Fall, dass eine HPC-Anwendung mehr Ressourcen benötigt, als ein einzelnes HPC-System zur Verfügung stellen kann, könnten Ressourcen auch zwischen HPC-Systemen geteilt werden. Ein solches System, welches aus mehreren vernetzten HPC-Systemen besteht, wurde von Foster und Kesselman als Grid-System vorgeschlagen [2]. Das vorgeschlagene System soll es ermöglichen, dass ein Nutzer benötigte Rechenressourcen ohne detailliertes Wissen über die zugrundeliegende Hardware nutzen kann.

1.3. Ressourcenmanagement auf HPC-Systemen

Sollen mehrere Anwendungen auf einem System ausgeführt werden, muss entschieden werden, welche Anwendung über welchen Zeitraum welche Ressourcen des HPC-Systems nutzen darf. Diese Aufgabe wird von einem Ressourcen-Verwaltungssystem durchgeführt und als Scheduling bezeichnet.

Heutzutage nutzen die meisten HPC-Ressourcen-Verwaltungssysteme ein Warteschlangen-basiertes Scheduling [4, Kap. 2.1]. Dabei werden eingereichte Jobs (Kombination von Anwendung und dazugehörigen Eingabeparametern) nach Priorität in eine Warteschlange eingereiht und ausgeführt, sobald die nötigen Ressourcen zur Verfügung stehen. Um die Auslastung des Systems zu verbessern, werden sogenannte Backfilling-Strategien angewendet, durch die Jobs mit geringerem Ressourcenbedarf vorgezogen werden können, wenn dadurch höher priorisierte Jobs nicht verzögert werden.

Für den Nutzer eines HPC-Systems kann es wichtig sein, die genaue Antwortzeit seines Jobs zu kennen. Das ermöglicht zum Beispiel die Reservierung von Netzwerkressourcen im Voraus, um das Resultat einer Berechnung für die weitere Nutzung zu übertragen.

Allerdings kann ein Warteschlangen-basiertes System mit Backfilling-Mechanismen nur Worst-Case-Deadlines garantieren und keine präzise Vorhersage über die Antwortzeit liefern. Das liegt zum Beispiel daran, dass durch das Vorziehen eines Jobs dessen Antwortzeit unvorhergesehen verkürzt werden könnte. Um Antwortzeiten garantieren zu können, wurde eine neue Scheduling-Strategie vorgeschlagen: planungsbasiertes Scheduling [4].

Im Gegensatz zum Warteschlangen-basierten Scheduling wird beim planungsbasierten Scheduling nicht nur der aktuelle Zustand des Systems für Scheduling-Entscheidungen in Betracht gezogen. Stattdessen wird für die Zukunft ein Plan ausgearbeitet, der beschreibt, wann und wie lange welcher Job ausgeführt wird und welche Ressourcen ihm dazu zur Verfügung gestellt werden. Dieser Plan setzt sich zusammen aus den Ausführungsplänen für die einzelnen Jobs.

2. Problemstellung

Wurde für eine HPC-Anwendung ein Ausführungsplan erstellt, ist es wichtig, dass das Ressourcen-Verwaltungssystem auch in der Lage ist, diesen umsetzen zu können. Beispielsweise muss sichergestellt werden, dass die Prozesse der Anwendung auf den für sie vorgesehenen Knoten des HPC-Systems ausgeführt werden. Im Rahmen dieser Arbeit wird untersucht, wie das für MPI-Anwendungen umgesetzt werden kann.

2.1. Message Passing Interface (MPI)

MPI-Anwendungen sind in dieser Arbeit HPC-Anwendungen, die das *Message-Passing Interface* (MPI) nutzen.

Dieses Interface definiert Funktionen für den Nachrichtenaustausch zwischen parallel laufenden Prozessen einer Anwendung. Es legt dabei lediglich die Schnittstelle mit Syntax und Semantik der Operationen fest, enthält jedoch keine konkrete Implementierung. MPI bietet für Anwendungs-Programmierer den Vorteil, dass Anwendungen durch Nutzung von MPI portabel sind, d. h. unabhängig von einer konkreten Rechner- und Netzwerkarchitektur. Dies vereinfacht zudem die Nutzung von Interprozesskommunikation.

Die erste Veröffentlichung des MPI-Standards war im Jahr 1994 [7]. Seitdem wurde die MPI Spezifikation stetig weiterentwickelt. Mit MPI-2 wurde die Möglichkeit eingeführt, zusätzliche MPI-Prozesse zur Laufzeit zu erzeugen [8]. Diese Prozesse werden im Folgenden als *dynamisch erzeugte Prozesse* bezeichnet. Zuvor gab es bei MPI nur die Möglichkeit, initial beim Start des Programms Prozesse zu erzeugen (*initial erzeugte Prozesse*).

Die Verwendung dynamisch erzeugter Prozesse kann sinnvoll sein, um zusätzliche Rechenlast auf mehrere Prozesse zu verteilen. Eine solche Last kann entstehen, wenn eine Simulation zur Laufzeit lokal verfeinert werden soll (adaptive mesh refinement). Zum Beispiel kann bei einer Simulation eines Atmosphärenmodells, in dem starke Turbulenzen auftreten, ein bestimmter Bereich genauer untersucht werden.

2.2. Verwandte Arbeiten

Die Umsetzung eines planungsbasierten Mappings für MPI-Anwendungen wurde bereits in [5] untersucht. Dabei wurde festgestellt, dass Open MPI für die initialen Prozesse einer Anwendung bereits die Möglichkeit bietet, diese bestimmten Ressourcen zuzuweisen.

Für dynamisch erzeugte Prozesse wurde eine Möglichkeit aufgezeigt, wie nach dem Start der Prozesse deren Aufgaben untereinander getauscht werden können. Dies ermöglicht es bei einem heterogenen System, also einem System mit unterschiedlich leistungsstarken Knoten, diese gleichmäßiger auszulasten. Jedoch wurde nicht untersucht, wie die Prozesse bereits vor der Erzeugung bestimmten Knoten zugewiesen werden können.

2.3. Aufgabenstellung

In dieser Arbeit wird untersucht, wie dynamisch erzeugte Prozesse bereits vor der Ausführung mittels planungsbasiertem Mapping bestimmten Knoten zugewiesen werden können.

Es wird angenommen, dass diese Zuordnung aus einem Ausführungsplan für die Anwendung hervorgeht, der durch ein planungsbasiertes Ressourcen-Verwaltungssystem erstellt wurde. Es werden keine Anforderungen an die Struktur des Plans oder die Art, in der der Plan vorliegt, gestellt. Es wird lediglich vorausgesetzt, dass das Verwaltungssystem eine Schnittstelle bereitstellt, über die das Mapping von zu startenden Prozessen abgefragt werden kann.

Im Folgenden soll eine prototypische Umsetzung einer Lösung erarbeitet werden, die folgende Anforderungen erfüllt:

- Es soll möglich sein, dynamisch erzeugte Prozesse beliebigen, für die Anwendung bereits allozierten Knoten eines HPC-Systems zuzuweisen. Diese Zuweisung erfolgt dabei gemäß eines Ausführungsplans.
- Der Quellcode von MPI-Anwendungen, für die das Mapping erfolgen soll, darf nicht verändert werden.

3. Grundlagen

Als MPI-Implementierung wird in dieser Arbeit Open MPI in der Version 4.1.4 verwendet. Dabei handelt es sich um quelloffene Software, die eine Architektur aufweist, die das Ziel hat, möglichst flexibel einsetzbar und erweiterbar zu sein. [3]

Open MPI 4.1.4 implementiert die Version 3.1 des MPI-Standards.

In diesem Kapitel werden einige Grundlagen zu Open MPI und zur Ressourcenzuweisung behandelt.

3.1. Architektur von Open MPI

Die Architektur von Open MPI nennt sich *Modular Component Architecture* (MCA). Diese beinhaltet verschiedene *Frameworks*, die jeweils für einen bestimmten Teilbereich der Funktionalität von Open MPI zuständig sind. Ein Beispiel für ein Framework ist das `btl`-Framework (Byte Transfer Layer), welches den Nachrichtenaustausch über verschiedene Netzwerke regelt. Innerhalb der Frameworks gibt es eine oder mehrere *Komponenten* (auch *Add-Ons* oder *Plugins* genannt), die diese Funktionalität implementieren (siehe Abbildung 3). Im `btl`-Framework gibt es beispielsweise eine TCP-Komponente. Das Framework gibt dazu eine Schnittstelle vor, der die Komponenten entsprechen müssen. Auf diese Weise ist es möglich, dieselbe Funktionalität auf verschiedene Arten bereitzustellen. [3, 10]

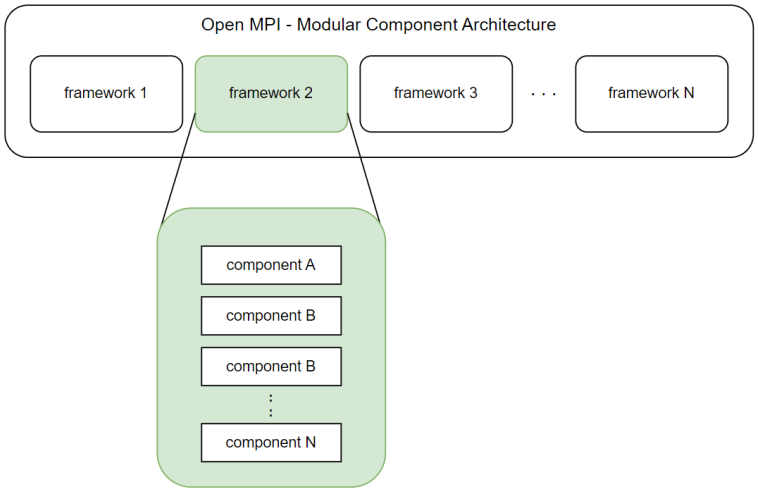

Abbildung 3: Die Modular Component Architecture von Open MPI beinhaltet mehrere Frameworks, deren Funktionalität in einer oder mehrerer Komponenten implementiert ist. Abbildung nach [3, Abb. 1]

Welche Komponenten zur Laufzeit verwendet werden, hängt von verschiedenen Faktoren ab, kann aber in der Regel vom Nutzer ausgewählt werden. Beispielsweise wird eine Komponente für ein bestimmtes (externes) Ressourcen-Verwaltungssystem nur dann verwendet, wenn dieses auf dem HPC-System genutzt wird.

Open MPI erlaubt es dem Nutzer, interne Parameter (*MCA-Parameter*) zur Laufzeit anzupassen. Dadurch können Komponenten ausgewählt werden und deren Verhalten beeinflusst werden. Beispielsweise kann man für die oben erwähnte TCP-Komponente einstellen, welche Netzwerkschnittstellen genutzt werden sollen. Dadurch kann das Verhalten einer Open MPI Installation präzise auf ein bestimmtes HPC-System abgestimmt werden, ohne Open MPI neu zu kompilieren zu müssen. [10]

MCA-Parameter können auf vielfältige Weise gesetzt werden – unter anderem beim Aufruf von `mpirun` (siehe Kapitel 3.2.1) mittels `--mca <parameter> <wert>`, über Umgebungsvariablen der Form `OMPI_MCA_<paramter>=<wert>` und in Konfigurationsdateien (sowohl systemweit als auch für den lokalen Benutzer). Außerdem kann mittels `--tune` beim Aufruf von `mpirun` eine Liste von Dateien angegeben werden, in denen MCA-Parameter gesetzt werden. [11]

In der Implementierung des Prototyps werden MCA-Parameter genutzt, um eine Verbindungs-Konfiguration festzulegen, anhand welcher eine Verbindung zum Ressourcen-Verwaltungssystem hergestellt werden kann (siehe Kapitel 4.2.1).

3.2. Open Run-Time Environment (OpenRTE)

Für die Ausführung eines MPI-Programms wird eine Laufzeitumgebung benötigt, die die Prozesse der Anwendung startet, diese während der Ausführung überwacht und gegebenenfalls beendet. Open MPI hat dafür das `rte`-Framework, in welches eine Laufzeitumgebung mittels einer Komponente eingebunden werden kann. *Open Run-Time Environment* (OpenRTE) [1] ist eine solche Laufzeitumgebung, die standardmäßig in Open MPI verwendet wird. Die Nutzung von OpenRTE wird im Folgenden vorausgesetzt. Wie Open MPI verwendet auch OpenRTE die *Modular Component Architecture* und beinhaltet diverse Plugins für externe Ressourcen-Verwaltungssysteme.

Die nachfolgenden Abschnitte behandeln zum einen den Start einer MPI-Anwendung mittels `mpirun` und zum anderen das Starten weiterer Prozesse mit `MPI_Comm_spawn`. Beide Vorgänge werden jeweils zunächst aus Sicht des Nutzers beschrieben. Anschließend wird auf die Zuweisung der Ressourcen in OpenRTE eingegangen.

Grundsätzlich sind für die Ressourcenzuweisung zwei Schritte notwendig. Zuerst werden die Ressourcen, die für eine Anwendung zur Verfügung stehen, ermittelt. Dies wird hier als *Allocation* bezeichnet. Anschließend werden die zu startenden Prozesse der Anwendung den Ressourcen zugeordnet. Dieser Schritt wird als *Mapping* bezeichnet.

3.2.1. Start eines MPI-Programms mit `mpirun`

Um eine MPI-Anwendung über OpenRTE zu starten, verwendet man das Kommando `orterun` oder die symbolischen Links `mpirun` oder `mpiexec`. Der Aufruf hat dabei üblicherweise folgende Form:

```
mpirun [ <options> ] <program> [ <args> ]
```

wobei `program` das auszuführende Programm ist und `args` eine Liste von Argumenten, die beim Start des Programms übergeben werden sollen. Mit den `options` kann das Verhalten von Open MPI gesteuert werden. Hierbei kann unter anderem die Anzahl der zu startenden Prozesse spezifiziert werden sowie die Liste der Knoten, die der Anwendung zur Verfügung stehen. Zudem können Regeln angegeben werden, nach denen die Zuweisung der Prozesse erfolgen soll (*Mapping-Policies*).

Es gibt, neben der oben gezeigten Form, auch die Möglichkeit, mehrere Programme mit einem einzelnen Aufruf von `mpirun` gemeinsam zu starten. Dabei werden die Programme (mit ihren Argumenten) jeweils durch einen Doppelpunkt voneinander getrennt nacheinander angegeben.

Die Angabe der zu verwendenden Knoten kann auf verschiedenen Wegen erfolgen:

`--host` Hiermit wird eine Liste von Knoten angegeben und optional für jeden Knoten die Anzahl an Prozessen, die auf dem Knoten gestartet werden können (Anzahl der *Slots*), bspw. `--host node1:2,node2:2` für jeweils 2 Slots auf den Knoten node1 und node2.

`--hostfile` Die Namen der Knoten können auch in einer Datei abgelegt werden. Auch hier kann die Anzahl der Slots angegeben werden.

`--default-hostfile` Das Default-Hostfile wird verwendet, falls für ein Programm keine Knoten angegeben wurden.

`--rankfile` Ein Rankfile enthält für jeden zu startenden Prozess den Namen des Knotens, auf dem dieser ausgeführt werden soll.

Wenn eine MPI-Anwendung über `mpirun` gestartet wird, wird zunächst der *Head Node Process* (HNP) von OpenRTE gestartet. Der HNP ermittelt zunächst die Liste der Knoten, die für den Job zur Verfügung stehen (*Allocation*), und startet anschließend auf jedem dieser Knoten einen OpenRTE-Daemon. Diese erkunden ihr jeweiliges System und schicken eine Nachricht mit der Topologie des Knotens zurück an den HNP. Im nächsten Schritt nimmt der HNP die Zuweisung der Prozesse auf die Knoten vor (*Mapping*), wobei er die Topologie-Informationen der Daemons verwenden kann. Anschließend sendet der HNP eine *Launch-Message* an alle Daemons, aus der hervorgeht, welcher Prozess auf welchem Knoten gestartet werden soll. Die Daemons starten daraufhin die ihnen zugewiesenen Prozesse und melden schließlich dem HNP den erfolgreichen Start ihrer Prozesse.

Für die Zuweisung der Prozesse zu bestimmten Knoten sind insbesondere die Schritte „Allocation“ und „Mapping“ interessant, weshalb diese im Folgenden näher betrachtet werden.

Allocation Im Allocation-Schritt wird die Liste der zur Verfügung stehenden Knoten und die jeweilige Anzahl an Prozessen, die auf den Knoten gestartet werden kann, ermittelt. Letzteres wird in Open MPI als die Anzahl der *Slots* bezeichnet, wobei ein Slot eine Ressourceneinheit für einen Prozess darstellt [11]. Wie die Allocation geschieht, hängt davon ab, welches Verwaltungssystem auf dem HPC-System verwendet wird und ob dieses von Open MPI unterstützt wird. Gibt es ein solches System, werden die Informationen über die zu verwendenden Knoten über ein entsprechendes Plugin im `ras`-Framework (OpenRTE Resource Allocation Subsystem) abgerufen.

Zur Ermittlung der Ressourcen wird die Funktion `orte_ras_base_allocate` aufgerufen. Hier wird zunächst geprüft, ob eines der Plugins Informationen über zu verwendende Knoten bereitstellen kann. Ist das nicht der Fall, wird der Reihe nach geprüft, ob beim Start

- ein Rankfile angegeben wurde,
- Knoten mittels `--host` angegeben wurden,
- ein Hostfile angegeben wurde oder
- ein Default-Hostfile angegeben wurde.

Trifft einer der Fälle zu, werden die entsprechenden Knoten in die Liste der verfügbaren Knoten aufgenommen und die Suche beendet. Wurde bis dahin nichts gefunden, wird lediglich der Knoten, auf dem der HNP ausgeführt wird, in die Liste aufgenommen.

Eine Besonderheit des Allocation-Schritts ist es, dass dieser nur einmal beim Start mit `mpirun` ausgeführt wird und bei weiteren Aufrufen der Funktion `orte_ras_base_allocate` übersprungen wird. Das betrifft insbesondere dynamisch erzeugte Prozesse, die mit `MPI_Comm_spawn` gestartet werden.

Mapping In diesem Schritt werden die Anwendungsprozesse auf die zur Verfügung stehenden Knoten gemäß einer Mapping-Policy verteilt. Standardmäßig werden unter anderem folgende Mapping-Policies unterstützt:

- Es gibt verschiedene Round-Robin-Policies, die jeweils auf einer bestimmten Ebene agieren. So verteilt `--map-by node` die Prozesse gleichmäßig über alle Knoten, während `--map-by slot` erst alle Slots auf einem Knoten auffüllt, bevor mit dem nächsten Knoten fortgefahren wird.
- Mit `--map-by ppr:N:<object>` wird im einfachsten Fall auf allen Knoten die gleiche Anzahl an Prozessen gestartet. Dies kann auch proportional zur Anzahl an Cores, Sockets, L3-Caches oder ähnlichem erfolgen.

- Mit `--map-by sequential` werden die Prozesse in der gleichen Reihenfolge auf die verfügbaren Knoten verteilt, in der die Knoten mittels `--host`, `--hostfile` oder `--default-hostfile` angegeben wurden.
- Mithilfe eines Rankfiles kann für jeden MPI-Prozess ein bestimmter Knoten, auf dem der Prozess ausgeführt werden soll, angegeben werden.

Wie schon in [5] gezeigt wurde, ist es mit diesen Mitteln möglich, die initialen Prozesse einer MPI-Anwendung bestimmten Knoten eines HPC-Systems zuzuweisen.

Für das Mapping ist das `rmaps`-Framework (OpenRTE Resource Mapping Subsystem) zuständig. Die Komponenten des Frameworks setzen jeweils eine bestimmte Mapping-Policy um. Beim Start von OpenRTE werden zunächst alle verfügbaren Mapping-Komponenten ermittelt und initialisiert. Dabei geben alle Komponenten eine Priorität an, nach der die Komponenten anschließend sortiert werden. Soll das Mapping für eine Anwendung berechnet werden, wird die Komponente mit der höchsten Priorität aufgerufen. Kann diese das Mapping nicht erstellen (z. B. weil eine andere Mapping-Policy vorgegeben wurde), kann die Komponente `ORTE_ERR_TAKE_NEXT_OPTION` zurückgeben, woraufhin die Komponente mit der zweithöchsten Priorität aufgerufen wird usw.

3.2.2. Prozesserzeugung mit `MPI_Comm_spawn`

Mit Version 2 der MPI-Spezifikation wurden zwei neue Funktionen hinzugefügt, die das Starten von weiteren Prozessen zur Laufzeit ermöglichen: `MPI_Comm_spawn` und `MPI_Comm_spawn_multiple`. Beide Funktionen starten neue MPI-Prozesse und richten die Kommunikation mit diesen ein. Mit `MPI_Comm_spawn_multiple` kann man im Gegensatz zu `MPI_Comm_spawn` mehrere Programme gleichzeitig starten oder dasselbe Programm mit verschiedenen Eingabedaten.

```
MPI_COMM_SPAWN(command, argv, maxprocs, info, root, comm,
    intercomm, array_of_errcodes)
```

Folgende Parameter werden beim Aufruf von `MPI_Comm_spawn` übergeben: `command` ist der Name des Programms und `argv` dessen Eingabedaten. `maxprocs` gibt an, wie viele Prozesse gestartet werden sollen[1]. Im `info` Argument können Paare von Schlüsseln und Werten abgelegt werden, die das Verhalten des Ressourcenmanagers beeinflussen, um Implementierungs-spezifische Funktionen nutzen zu können. `root` gibt den Rank des Prozesses in `comm` an, bei dem die vorherigen Argumente verarbeitet werden. `comm` ist der Kommunikator der Eltern-Prozesse und `intercomm` der Kommunikator zwischen Eltern- und Kind-Prozessen. In `array_of_errcodes` wird für jeden Kind-Prozess angegeben, ob der Start erfolgreich war.

[1] Obwohl die Bezeichnung eine Obergrenze suggeriert, wird eine Fehlermeldung erzeugt, wenn nicht genau `maxprocs` Prozesse erzeugt werden können. Dies wird als „hard spawn“ bezeichnet und ist das Standardverhalten. MPI-Implementierungen können zusätzlich einen „soft spawn“ anbieten, bei dem auch weniger Prozesse erzeugt werden dürfen.

Das `info` Argument ermöglicht es, Umgebungs-spezifische Funktionen zu nutzen. Hierzu werden Paare von Schlüsseln und Werten abgelegt. Einige Schlüssel sind von MPI reserviert, das heißt, wenn eine MPI-Implementierung sie unterstützt, müssen sie einer vorgegebenen Semantik folgen.

Für das Mapping sind insbesondere folgende Schlüssel in Open MPI relevant:

host Liste von Knoten, auf denen die Anwendung gestartet werden soll.

hostfile Das zu verwendende Hostfile für Anwendung.

add-host Fügt neue Knoten hinzu, falls diese noch nicht im Allocation-Schritt der Eltern-Prozesse angegeben wurden und nutzt diese für die Anwendung.

add-hostfile Fügt Knoten aus einem Hostfile hinzu und verwendet diese beim Mapping.

mapper Gibt an, welche Mapping-Komponente für das Mapping genutzt werden soll.

map_by, npernode, pernode, ppr Angabe bestimmter Mapping-Policies.

Man kann hiermit also auch für Prozesse, die mittels `MPI_Comm_spawn` erzeugt wurden, festlegen, welche Knoten genutzt werden sollen. Diese müssen allerdings bereits im ursprünglichen Allocation-Schritt ermittelt worden sein oder über `add-host` oder `add-hostfile` hinzugefügt werden. Zudem müssen die Knoten über genügend freie Slots verfügen, damit ihnen dynamisch erzeugte Prozesse zugewiesen werden können. Die Anzahl verfügbarer Slots auf bereits bekannten Knoten lässt sich in Open MPI nicht modifizieren, was insbesondere bei Nutzung eines Rankfiles relevant ist, denn hier entspricht die Anzahl verfügbarer Slots genau der Anzahl der genutzten Slots.

Wie viele Prozesse auf diesen Knoten gestartet werden sollen, hängt von der Mapping-Policy und der verwendeten Mapping-Komponente ab. Ist eine beliebige Zuordnung gewünscht, lässt sich diese mit dem `seq`-Mapper (sequentielles Mapping) realisieren. Dabei werden die Knoten in der Reihenfolge, die durch `host` oder `hostfile` vorgegeben ist, verteilt.

Wenn ein MPI-Prozess die Funktion `MPI_Comm_spawn` aufruft, wird die Anfrage zum Starten neuer Prozesse zunächst an den lokalen OpenRTE-Daemon weitergereicht, der diese an den HNP schickt. Ab diesem Punkt werden mit Ausnahme des Allocation-Schritts, die gleichen Abläufe durchgeführt wie bei `mpirun` (siehe Kapitel 3.2.1). Für das Mapping stehen also grundsätzlich nur die Knoten zur Verfügung, die beim Allocation-Schritt der Eltern-Prozesse ermittelt wurden. Das Starten neuer OpenRTE-Daemons wird nur dann durchgeführt, wenn mittels `add-host` oder `add-hostfile` neue Knoten hinzugefügt wurden.

4. Umsetzung

In diesem Kapitel werden einige Ansätze vorgestellt, wie man die Zuweisung der Prozesse beeinflussen kann. Anschließend wird die Implementierung des Prototyps sowie die Testumgebung vorgestellt.

4.1. Lösungsansätze

Wie in Kapitel 3.2.2 erläutert, kann das Mapping dynamisch erzeugter Prozesse durch Einträge im `info`-Argument von `MPI_Comm_spawn` beeinflusst werden. Zum einen können die zu verwendenden Knoten angegeben werden und zum anderen eine geeignete Mapping-Policy ausgewählt werden, die den Knoten jeweils bestimmte Prozesse zuweist. Hierfür eignet sich besonders die sequentielle Mapping-Policy. Es wäre also naheliegend, die entsprechenden Einträge im `info`-Argument vorzunehmen, um ein bestimmtes Mapping zu erreichen.

Dieser Ansatz würde allerdings eine Modifikation des Quellcodes der MPI-Anwendung erfordern, was den Anforderungen aus Kapitel 2.3 widerspricht. Das Eingreifen in das Mapping kann frühestens in der Funktion `MPI_Comm_spawn` erfolgen und muss abgeschlossen sein, bevor die Launch-Message, die die Zuweisung der Ressourcen enthält, für die OpenRTE-Daemons erstellt wird. Grundsätzlich wäre es denkbar, die erwähnten Einträge an anderer Stelle zu ergänzen bzw. zu modifizieren, was vor der Berechnung des Mappings durch den HNP erfolgen müsste.

Bereits existierende Ressourcen-Verwaltungssysteme werden in OpenRTE aktuell über Plugins in mehreren Frameworks unterstützt: Mit dem `ras`-Framework werden die für den Job vorgesehenen Knoten abgerufen. Im `plm`-Framework (Process Lifecycle Management Subsystem), wird das Starten der OpenRTE-Daemons mit den Mechanismen des jeweiligen Verwaltungssystems umgesetzt. Das `ess`-Framework (Environment-Specific Services) bietet eine Möglichkeit, die Initialisierung von OpenRTE an das jeweilige Verwaltungssystem anzupassen. Ein planungsbasiertes Verwaltungssystem könnte auf die gleiche Weise, über eine `ras`-Komponente, verwendet werden. Es wäre dann jedoch darauf beschränkt, einmalig die Liste der verfügbaren Knoten festzulegen. Eine solche Komponente wird bei dynamisch erzeugten Prozessen gar nicht ausgeführt (siehe Abschnitt 3.2.1). Dieser Ansatz ist also nicht zielführend.

Ein weiterer Ansatz für die Umsetzung eines planungsbasierten Mappings ist das Schreiben einer Komponente für das in Kapitel 3.2.1 beschriebene `rmaps`-Framework. Eine solche Komponente weist den Prozessen einer MPI-Anwendung Knoten zur Ausführung zu. Dadurch ist eine beliebige Zuordnung der Prozesse auf eine Weise möglich, die mit der Architektur von Open MPI kompatibel ist und keine Änderungen an vorhandenem Code erfordert. Daher wird dieser Ansatz im folgenden Abschnitt umgesetzt.

4.2. Planungsbasierte Mapping-Komponente

Bevor das Mapping berechnet werden kann, müssen im Allocation-Schritt die zur Verfügung stehenden Knoten ermittelt worden sein und für jeden Knoten die Anzahl der Prozesse, die ihm zugeordnet werden können. Theoretisch können für dynamisch erzeugte Prozesse neue Knoten hinzugefügt werden. In der Praxis hat das auf dem Testsystem (siehe Kapitel 4.3) leider nicht funktioniert. Aus diesem Grund wird vorausgesetzt, dass die initial ermittelte Menge von Knoten vollständig ist und im Laufe der Ausführung der Anwendung nicht geändert wird.

Das `rmaps`-Framework gibt als Schnittstelle für Plugins zwei Funktionen an: `map_job` und `assign_locations`. Erstere wird durch den HNP aufgerufen und legt für jeden Prozess den Knoten fest, auf dem dieser ausgeführt werden soll. `assign_locations` wird sowohl vom HNP als auch von den OpenRTE-Daemons ausgeführt, nachdem diese die Launch-Message erhalten haben. Hier wird jedem Prozess eine bestimmte Position in der Topologie des Knotens zugewiesen. Das kann beispielsweise ein bestimmter CPU-Kern sein oder ein beliebiger CPU-Kern, der einen bestimmten Cache nutzt. Die meisten Mapping-Komponenten führen diesen Schritt bereits in der `map_job`-Funktion aus.

Im Rahmen der prototypischen Implementierung wird das Mapping lediglich auf der Ebene der Knoten durchgeführt. Einzelne CPU-Kerne oder verschiedene Cache-Level werden hier nicht betrachtet.

Die Implementierung der planungsbasierten Mapping-Komponente basiert auf der Implementierung der bestehenden `seq`-Komponente für sequentielles Mapping. Der Unterschied besteht im Wesentlichen darin, wie die Komponenten die Listen der Knoten beziehen, die den Prozessen zugewiesen werden sollen. Die `seq`-Komponente arbeitet bei initial erzeugten Prozessen mit der Liste aus `--host` oder `--hostfile` und im Fall von `MPI_Comm_spawn` mit `host` oder `hostfile` aus dem `info`-Argument. Die planungsbasierte Komponente bezieht ihre Liste der Knoten hingegen vom planungsbasierten Ressourcen-Verwaltungssystem.

4.2.1. Abfrage der Knoten vom planungsbasierten Verwaltungssystem

Für die Übermittlung der Liste der Knoten gibt es mehrere Möglichkeiten. Denkbar wäre das Einlesen aus einer Datei, das Einlesen bestimmter Umgebungsvariablen oder eine Abfrage an das Verwaltungssystem mittels Interprozesskommunikation. In dieser Arbeit wird der letzte Fall implementiert. Dies erlaubt es dem Verwaltungssystem, den Ausführungsplan nach dem Start einer MPI-Anwendung zu ändern, da die Abfrage der zu verwendenden Knoten zum spätest-möglichen Zeitpunkt erfolgt. Zudem wird das Verwaltungssystem auf diesem Weg über den beabsichtigten Start neuer Prozesse informiert.

Für die Abfrage der Knoten wird in dieser Implementierung eine TCP-Verbindung zum Verwaltungssystem hergestellt. Es wird angenommen, dass das Verwaltungssystem eine

solche Schnittstelle bereitstellt. Die IP-Adresse und der Port, unter dem das Verwaltungssystem erreichbar ist, werden der planungsbasierten Mapping-Komponente über die MCA-Parameter `rmaps_planbased_rms_host` und `rmaps_planbased_rms_port` mitgeteilt.

Für die prototypische Implementierung wird ein sehr simples Protokoll mit zwei Schritten verwendet. Zunächst stellt die Mapping-Komponente eine Verbindung zum Verwaltungssystem her und übermittelt Daten über die zu startende Anwendung. Das Verwaltungssystem antwortet anschließend mit einer Liste von Knoten, wobei für jeden Prozess genau ein Knoten angegeben wird. Sollen mehrere Prozesse auf einem Knoten gestartet werden, ist dieser mehrmals anzugeben.

Der Prototyp übermittelt im ersten Schritt die `jobid` und die `vpid` des Prozesses, welcher beim Aufruf von `MPI_Comm_spawn` im `root`-Argument angegeben wurde (bei initial erzeugten Prozessen wird stattdessen `INVALID` übermittelt) sowie die `jobid` der zu startenden Anwendung. Die Werte `jobid` und `vpid` werden innerhalb von OpenRTE dazu genutzt, MPI-Prozesse und OpenRTE-Prozesse zu identifizieren. Die Abfrage der Knoten ist in der Funktion `get_vrm_node_list` implementiert.

4.2.2. Umsetzung des Mappings

In diesem Abschnitt wird auf die Implementierung der zu Beginn des Kapitels beschriebenen `map_job`-Funktion eingegangen.

Die `map_job`-Funktion der planungsbasierten Mapping-Komponente ruft zunächst die Liste der zu verwendenden Knoten mittels `get_vrm_node_list` ab. Im Anschluss werden den Prozessen der Reihe nach diese Knoten zugewiesen.

Dazu wird die Funktion `orte_rmaps_base_setup_proc` aufgerufen. Hier wird für einen MPI-Prozess eine neue Datenstruktur vom Typ `orte_proc_t` angelegt. In das Feld `parent` wird hierbei die `vpid` des OpenRTE-Daemons eingetragen, der den Prozess starten soll. Die `vpid` ist in `node->daemon->name.vpid` hinterlegt, wobei `node` die Datenstruktur ist, die den Knoten beschreibt, auf dem der Prozess ausgeführt werden soll.

Jeder OpenRTE-Daemon führt eine Liste mit lokalen MPI-Prozessen (`orte_local_children`), für die er verantwortlich ist. Nachdem der HNP die Launch-Message an die Daemons geschickt hat, prüfen diese für alle neuen Prozesse, ob im `parent`-Feld die `vpid` des jeweiligen Daemons steht. In diesem Falle werden die entsprechenden Prozesse in die `orte_local_children`-Liste übernommen. Anschließend werden alle neuen Prozesse dieser Liste gestartet.

4.3. Testaufbau

Zum Testen der planungsbasierten Mapping-Komponente wurden die nicht zugangsbeschränkten Linux Compute-Server des IT-Dienstes des Fachbereichs Mathematik und Informatik der Freien Universität Berlin genutzt[2].

Für die Überprüfung der Funktionalität werden zwei Programme benötigt: ein Serverprogramm, welches das planungsbasierte Verwaltungssystem bei der Beantwortung der Anfragen der Mapping-Komponente simuliert, und eine (beliebige) MPI-Anwendung, die mittels `MPI_Comm_spawn` oder `MPI_Comm_spawn_multiple` neue Prozesse erzeugt.

4.3.1. Serverprogramm

An die Stelle des Verwaltungssystems tritt ein einfacher TCP-Server, der Anfragen von der planungsbasierten Mapping-Komponente entgegennimmt und diese mit einer Liste von Knoten beantwortet. Der Server unterstützt aktuell nur eine MPI-Anwendung gleichzeitig. Als Eingabe wird eine Zeichenkette der Form

```
<parent-jobid>;<parent-vpid>;<child-jobid>
```

erwartet. Zurückgegeben wird eine Liste der Form

```
<node1>,<node2>, ...
```

Welche Liste zurückgegeben wird, kann für jeden Prozess der MPI-Anwendung, der `MPI_Comm_spawn` aufruft, individuell festgelegt werden. Dazu wird in einem Server-Hostfile eine Liste von Prozess-Kennungen zusammen mit der jeweiligen Liste von Knoten gespeichert. Die Kennung ergibt sich aus der `vpid` des Prozesses und den `vpid`s seiner Vorfahren. Für initiale Prozesse wird ein imaginärer Eltern-Prozess mit der `vpid` `init` angenommen. Eine Zeile des Server-Hostfiles könnte beispielsweise folgendermaßen aussehen:

```
init.2.0: node0,node2,node4,node2
```

Die Zeile sagt aus, dass die Liste `node0,node2,node4,node2` zurückgegeben werden soll, wenn der Prozess mit `vpid` 0, dessen Eltern-Prozess der initiale Prozess mit `vpid` 2 ist, `MPI_Comm_spawn` oder `MPI_Comm_spawn_multiple` aufruft.

Zum Testen der Zuordnung wurden die Angaben im Server-Hostfile variiert, um verschiedene Zuordnungen der Prozesse zu erreichen.

[2] Eine Beschreibung der Hardware ist auf der Webseite des IT-Dienstes unter `https://www.mi.fu-berlin.de/w/IT/ComputeServer` zu finden.

4.3.2. Beispielanwendung

Mit der planungsbasierten Mapping-Komponente wird das Mapping sowohl für initiale Prozesse als auch für dynamisch erzeugte Prozesse berechnet. Um zu zeigen, dass das Mapping für initiale Prozesse umgesetzt wird, reicht es, mit `mpirun` ein beliebiges Programm (z. B. `hostname`) zu starten, welches den Namen seines Knotens ausgibt.

Für die Überprüfung des Mappings dynamisch erzeugter Prozesse braucht man eine Anwendung, die mindestens einmal `MPI_Comm_spawn` oder `MPI_Comm_spawn_multiple` aufruft. Zu diesem Zweck wurde ein simples MPI-Programm geschrieben, welches weitere Instanzen von sich selbst durch einen Aufruf von `MPI_Comm_spawn` erzeugt. Ob die neuen Prozesse ebenfalls `MPI_Comm_spawn` aufrufen, wird dabei durch einen Eingabeparameter gesteuert, mit dem die Anzahl der Aufrufe festgelegt wird. Alle Prozesse geben dabei den Namen ihres Knotens aus.

Beim Bauen und bei der Ausführung der Beispielanwendung ist darauf zu achten, dass auf allen Knoten die korrekte Version von Open MPI (mit der planungsbasierten Mapping-Komponente) installiert ist und verwendet wird. Damit die Mapping-Komponente eine Verbindung zum Serverprogramm herstellen kann, muss dieses vor der MPI-Anwendung gestartet werden. Zudem müssen der Knotenname (oder die IP-Adresse) sowie der Port, unter dem das Serverprogramm erreichbar ist, mittels MCA-Parametern bekannt gegeben werden (siehe Kapitel 4.2.1 und 3.1).

5. Evaluation

Bei den Tests mit der in Kapitel 4.2 vorgestellten planungsbasierten Mapping-Komponente wurden die Prozesse jeweils auf den für sie vorgesehenen Knoten gestartet. Nutzeranwendungen müssen für den Einsatz der Mapping-Komponente nicht angepasst werden. Somit sind die Anforderungen aus Kapitel 2.3 erfüllt.

Unter anderem aufgrund des prototypischen Charakters der Implementierung sind einige Randbedingungen bei der Verwendung der planungsbasierten Mapping-Komponente zu beachten. Diese werden im nächsten Abschnitt behandelt.

5.1. Randbedingungen

Im Mapping-Schritt stehen nur die Knoten zur Verfügung, die zuvor im Allocation-Schritt ermittelt wurden, welcher nur einmal beim Start einer MPI-Anwendung durchgeführt wird. Daher ist es erforderlich, alle Knoten, die für initiale Prozesse und für dynamisch erzeugte Prozesse genutzt werden sollen, bereits beim Start der Anwendung anzugeben.

Bei der vom planungsbasierten Verwaltungssystem zurückgegebenen Liste von Knoten muss die Anzahl der Knoten mindestens so groß sein, wie die Anzahl der zu startenden Prozesse. Andernfalls ist für die verbleibenden Prozesse keine Zuordnung möglich.

Über das `info`-Argument der `MPI_Comm_spawn`-Funktion können Anwendungs-Programmierer unter anderem angeben, durch welche Mapping-Komponente das Mapping erfolgen soll. Die planungsbasierte Mapping-Komponente fragt diesen Eintrag ab, falls dieser gesetzt wurde, und berechnet nur dann ein Mapping, wenn keine andere Mapping-Komponente angegeben wurde. Dieses Verhalten wird auch von allen anderen Mapping-Komponenten implementiert. Das heißt, dass für die Verwendung der planungsbasierten Komponente keine andere Komponente angegeben werden darf.

5.2. Mögliche Auswirkungen planungsbasierter Ressourcenverwaltung

Die hier vorgestellte Lösung ermöglicht es einem planungsbasierten Ressourcen-Verwaltungssystem, MPI-Prozesse bestimmten Knoten zuzuweisen und somit Ausführungspläne für MPI-Anwendungen umsetzen zu können. Im Folgenden werden mögliche Vorteile aufgezeigt, die durch die Verwendung eines solchen Verwaltungssystems für die Nutzer und Betreiber von HPC-Systemen sowie für die Entwickler von HPC-Anwendungen entstehen.

Nutzer von HPC-Anwendungen können von zugesicherten Antwortzeiten ihrer Anwendungen profitieren, da diese nun verlässlich in externe Arbeitsabläufe eingeplant werden können. Zudem ist keine detaillierte Kenntnis der verwendeten Hardware notwendig, da

das Verwaltungssystem das Mapping selbständig vornimmt und dabei unterschiedliche Anforderungen der Prozesse an die Hardware berücksichtigen kann.

Anwendungsentwickler müssen ihre Programme für die Nutzung auf einem planungsbasiert verwalteten HPC-System nicht modifizieren, da die nötigen Schritte für die Umsetzung des Mappings im Hintergrund ablaufen. Eine genaue Kenntnis der zugrundeliegenden Hardware ist auch hier nicht zwingend nötig.

Durch planungsbasierte Ressourcen-Verwaltung ist potentiell eine effizientere Nutzung eines HPC-Systems möglich, da Ressourcen, die einer HPC-Anwendung zugeordnet wurden, aber nicht während der gesamten Ausführungszeit der Anwendung genutzt werden, zwischenzeitlich an andere Anwendungen vergeben werden können. Dadurch kann möglicherweise die Antwortzeit anderer Anwendungen gesenkt werden und die Auslastung des HPC-Systems erhöht werden, wodurch sowohl Nutzer als aus Betreiber eines HPC-Systems profitieren.

6. Fazit

Ein planungsbasiertes Ressourcen-Verwaltungssystem muss in der Lage sein, seine Ausführungspläne auch umsetzen zu können. Insbesondere müssen Prozesse von HPC-Anwendungen auf den für sie vorgesehenen Knoten gestartet werden können. In dieser Arbeit wurde untersucht, wie dies für die Prozesse einer MPI-Anwendung durchgeführt werden kann.

Dazu wurden die Abläufe bei die Ressourcenzuweisung in Open MPI und dessen Laufzeitumgebung OpenRTE untersucht und eine Lösung für das Mapping-Problem vorgeschlagen, die prototypisch implementiert wurde. Die Ergebnisse der Tests zeigen, dass eine planungsbasierte Zuordnung von MPI-Prozessen mit der vorgeschlagenen Lösung umsetzbar ist.

7. Ausblick

Die hier vorgeschlagene Lösung wurde gemäß der Aufgabenstellung prototypisch implementiert. Für die Verwendung auf einem Produktivsystem wären weitere Maßnahmen vorzunehmen, die eine umfängliche Fehlerbehandlung und eine effiziente Ressourcennutzung sicherstellen. Der Prototyp könnte zudem um die Fähigkeit erweitert werden, eine präzisere Zuordnung der Prozesse einer Anwendung vorzunehmen, etwa auf der Ebene von CPU-Kernen.

Ein weiterer Schritt wäre die Integration in ein planungsbasiertes Verwaltungssystem, welches dadurch ein Werkzeug erhält, um Ausführungspläne umsetzen zu können. Hierbei ist eine geeignete Schnittstelle zu definieren und zu implementieren.

Nicht zuletzt ist die Erstellung von Ausführungsplänen für HPC-Anwendungen, also insbesondere die Frage nach einer optimierten Zuweisung der Prozesse, ein Thema für zukünftige Forschung.

A. Anhang

A.1. Repositories der Implementierung

Der Quellcode der planungsbasierten Mapping-Komponente ist in folgendem git-Repository zu finden:

`https://git.imp.fu-berlin.de/paun51/ompi-planbased-mapping`

Die Implementierung des Servers und der Beispielanwendung aus Kapitel 4.3 ist unter folgendem Link zu finden:

`https://git.imp.fu-berlin.de/paun51/planbased_mapping_demo`

Literatur

[1] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett und G. E. Fagg. „The Open Run-Time Environment (OpenRTE): A Transparent Multi-Cluster Environment for High-Performance Computing“. In: *Proceedings, 12th European PVM/MPI Users' Group Meeting*. Sorrento, Italy, Sep. 2005.

[2] I. Foster und C. Kesselman, Hrsg. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[3] R. L. Graham, T. S. Woodall und J. M. Squyres. „Open MPI: A Flexible High Performance MPI“. In: *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*. Poznan, Poland, Sep. 2005.

[4] M. Hovestadt, O. Kao, A. Keller und A. Streit. „Scheduling in HPC Resource Management Systems: Queuing vs. Planning“. In: *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, S. 1–20.

[5] F. Korthals. „Entwurf und Implementierung einer Prozesszuordnungsstrategie für MPI-Prozesse in planungsbasierten Cluster-Verwaltungssystemen“. Bachelorarbeit. Freie Universität Berlin, Sep. 2021.

[6] L. Krüger. *Wind Field Simulation for Operation of UAS in Urban Environments*. 2022. URL: `https://www.hkhlr.de/en/projects/2773` (besucht am 04. 01. 2023).

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 1.0*. Mai 1994. URL: `https://www.mpi-forum.org/docs/mpi-1.0/mpi-10.ps`.

[8] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 2.0*. Juli 1997. URL: `https://www.mpi-forum.org/docs/mpi-2.0/mpi-20.ps`.

[9] T. Reimann. *Lichtenberg II Cluster Darmstadt*. o.D. URL: `https://www.hkhlr.de/en/clusters/lichtenberg-ii-cluster-darmstadt` (besucht am 04. 01. 2023).

[10] J. M. Squyres. „Open MPI“. In: *The Architecture of Open Source Applications. Structure, Scale, and a Few More Fearless Hacks*. Hrsg. von A. Brown und G. Wilson. Bd. 2. Self published, Apr. 2012. Kap. 15.

[11] The Open MPI Project. *Open MPI logo mpirun(1) man page (version 4.1.6)*. URL: `https://www-lb.open-mpi.org/doc/v4.1/man1/mpirun.1.php` (besucht am 04. 01. 2024).

Abbildungsverzeichnis