Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Entwicklung eines Verwaltungssystem für Cluster, auf welchem virtuelle Maschinen und High-Performance-Computing-Anwendungen gemeinsam ausgeführt werden.

Martin Görick
Matrikelnummer: 4207873
martin.goerick@zedat.fu-berlin.de

Erstgutachter: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr.-Ing. Jochen Schiller

Berlin, 10.01.2017

Zusammenfassung

In Firmen und Universitäten werden derzeit Cluster für verschieden Verwendungszwecke genutzt. Darunter zählt die Nutzung mit virtuellen Maschinen, welche Dienste, wie E-Mail-Server, Druckerserver oder Webserver, zur Verfügung stellen. Außerdem werden Cluster in Universitäten sowie großen Instituten auch für High-Performance-Computing verwendet, um rechenintensive Berechnungen durchzuführen, wie beispielsweise Klimaberechnungen oder Berechnungen von DNA-Sequenzen. Dazu wird ein Cluster entweder für virtuelle Maschinen oder High-Performance-Computing verwendet. In dieser Arbeit geht es um die Kombination dieser beiden Arten der Nutzung von Clustern, welche zusammen auf einem Cluster laufen sollen. Dazu wird ein Prototyp entwickelt, welcher ein Cluster verwalten soll, auf welchem virtuelle Maschinen und High-Performance-Computing-Anwendungen gemeinsam laufen sollen. Mit dem Prototyp wird getestet, ob eine Kombination beider Arten der Nutzung auf einem Cluster möglich ist. Dazu wird die Machbarkeit dieser Kombination ausgewertet und auf entstehende Voraussetzungen und Einschränkungen eingegangen.

Abstract

Companies and universities use clusters for different purposes. On the one hand they run virtual maschines to privide services like email, print or web server. High performance computing is another kind of usage, for CPU-intensive tasks like calculation of climate or DNA sequences. This thesis is about the combination of both kinds of cluster usage. Therefor was developed a prototyp to manage a cluster to run virtual machines and high performance computing together on it. The proto- typ is used to test this combination. The feasibility of this combination are evaluated and discuss the conditions and setting.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, 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. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

10.01.2017

Martin Görick

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation 1
1.2 Struktur dieser Arbeit 1

2 Definitionen **3**
2.1 Virtuelle Maschinen 3
2.1.1 OpenStack 5
2.1.2 Vorteile von virtuellen Maschinen 6
2.1.3 Nachteile von virtuellen Maschinen 7
2.2 High-Performance-Computing 8
2.2.1 Slurm 9
2.3 Unterschiede VMs und HPC-Anwendungen 10
2.4 Cluster 10
2.4.1 High-Availability- oder Failover-Cluster 11
2.4.2 Load-Balance-Cluster 13
2.4.3 High Availability und Load-Balance Cluster 13
2.4.4 Cluster für virtuelle Maschinen 14
2.4.5 Cluster für HPC-Anwendungen 14
2.4.6 Unterschiede VM- und HPC-Cluster 16

3 Aufgabe **18**
3.1 Beschreibung der Aufgabe 18
3.2 Voraussetzungen 18
3.3 Bestehende Probleme 19
3.4 Anforderungen von HIVICMA 21

4 Lösungsansatz **23**
4.1 HPC-Informationen 23
4.1.1 Timer erstellen 26
4.2 Informationen von der VM-Verwaltung 27
4.2.1 Information für Timer 28
4.3 Aufbau von HIVICMA 29
4.3.1 HPC-Modul 29
4.3.2 VM-Modul 30
4.4 Programmabläufe 32

5 Tests von HIVICMA **35**
5.1 Aufbau der Testumgebung 35
5.2 Einstellungen und Voraussetzungen 36
5.2.1 Slurm 36
5.2.2 OpenStack 37
5.3 Beschreibung der Tests 38
5.3.1 HPC-Informationen ermitteln 39
5.3.2 Knoten der HPC-Verwaltung ermitteln und manipulieren 39
5.3.3 VM-Informationen ermitteln und manipulieren 40
5.3.4 Knoten der VM-Verwaltung ermitteln und manipulieren 41
5.3.5 Kombination des HPC- und VM-Management-Systems 41
5.4 Testläufe 43
5.4.1 HPC-Modul Tests 43
5.4.2 VM-Modul Tests 45
5.4.3 HPC-VM-Kombination Tests 48
5.5 Auswertung der Testläufe 49
5.5.1 HPC-Modul Testläufe 49
5.5.2 VM-Modul Testläufe 50
5.5.3 HPC-VM-Kombination Testläufe 52

6 Fazit und Aussicht **53**
6.1 Fazit 53
6.2 Ausblick 54

7 Anhang **56**

1 Einleitung

In Universitäten und Firmen stehen Cluster für unterschiedliche Anwendungsarten zur Verfügung. Beispiele für deren Nutzung sind virtuelle Maschinen und High-Performance-Computing. In dieser Arbeit wird sich mit der Kombination dieser beiden Arten der Nutzung befasst. Dazu wird ein Prototyp für ein Cluster-Management-System entwickelt. Mit Hilfe dieses Prototyps soll gezeigt werden, dass die Möglichkeit vorhanden ist, dass diese beiden Arten der Clusternutzung gemeinsam auf einem Cluster umgesetzt werden können.

1.1 Motivation

Die derzeitige Infrastruktur der Freien Universität Berlin des Fachbereich Mathematik und Informatik besteht aus mehreren unterschiedlichen Clustern. So ist beispielsweise ein Cluster für die Bereitstellung von virtuellen Maschinen [20] vorhanden. Mit diesem werden virtuelle Maschinen für diverse Forschungen oder der Lehre zur Verfügung gestellt. Außerdem ist das Cluster "allegro"[11] für High-Performance-Computing vorhanden, um rechenintensive Anwendungen auszuführen.
Im Hinblick auf den aktuellen Stand ist es interessant, ob eine Kombination beider Arten der Nutzung von Clustern möglich ist. Die erfolgreiche Kombination könnte beispielsweise genutzt werden, um die Kosten für laufende Cluster zu verringern. Es ist nicht mehr notwendig, dass verschiedene Cluster, für beide Nutzungen, eventuell auch an unterschiedlichen Orten, unterhalten werden müssen. Der Unterhalt eines gemeinsames Clusters würde zudem auch den Verwaltungsaufwand für das Personal sowie Wartungsaufwand, wenn ein Cluster nur noch an einem Ort steht, reduzieren.

1.2 Struktur dieser Arbeit

Als erstes werden, in Abschnitt 2, die Begrifflichkeiten, wie virtuelle Maschinen, High-Performance-Computing und Cluster sowie deren Zusammenhänge miteinander beschrieben. Dazu werden die Vor- und Nachteile sowie die Unterschiede von virtuellen zu High-Performance-Computing sowie die in dieser Arbeit verwendete Software geschildert. Danach wird in Abschnitt 3 die genaue Aufgabe dieser Arbeit erläutert. Außerdem werden die Voraussetzung für die Umsetzung dieser Aufgabe, bestehende Probleme bei bisherigen

Umsetzungen auf diesem Gebiet sowie die Anforderungen an den zur Aufgabe gehörenden Prototypen dargestellt. Im Anschluss wird der Lösungsansatz für die Umsetzung der Aufgabe in Abschnitt 4 beschrieben. Dabei wird erläutert, welche Aufgaben die einzelnen Programmteile erfüllen müssen, um das Cluster zu verwalten. Im Abschnitt 5 werden danach die Tests mit dem Prototypen dargestellt. Dazu wird die gegebene Testumgebung beschrieben, welche Voraussetzungen gemacht und angenommen werden, welche Tests durchgeführt wurden, um die Aufgabe zu erfüllen, sowie die Abläufe der Tests. Danach werden die Tests kurz in Hinblick auf die Aufgabe und vorher gestellten Anforderungen ausgewertet. Am Ende folgt in Abschnitt 6 ein Fazit zur Machbarkeit der Kombination beider Cluster sowie ein Ausblick auf zukünftige Entwicklungen in diesem Bereich.

2 Definitionen

In diesem Abschnitt geht es um die Definitionen und Beschreibungen der Grundlagen dieser Arbeit. Zuerst wird in der Sektion 2.1 beschrieben, was virtuelle Maschinen sind und wie diese eingesetzt werden. Dazu wird eine Verwaltungsanwendung beschrieben, mit welcher virtuelle Maschinen betrieben werden können und welche in dieser Arbeit verwendet wird. Außerdem wird dargestellt, welche Vorteile und Nachteile der Einsatz von virtuellen Maschinen beinhaltet. Im Anschluss folgt im Teil 2.2 die Erklärungen für High-Performance-Computing sowie ein kurzer Überblick über deren Einsatz. Dabei wird die Anwendung für das Verwalten von HPC-Clustern kurz beschrieben, welche in dieser Arbeit zum Einsatz kommen. Danach werden im Abschnitt 2.3 die Unterschiede von virtuellen Maschinen und HPC-Anwendungen erläutert. Zum Abschluss wird im Abschnitt 2.4 erklärt, was ein Cluster ist sowie deren verschiedenen Typen, wie sie im heutigen Umfeld vorkommen, und wozu diese verwendet werden. Dazu werden die spezifischen Unterschiede der Cluster für die Verwendung mit virtuelle Maschinen und HPC-Anwendungen betrachtet.

2.1 Virtuelle Maschinen

Eine allgemeine Definition einer virtuellen Maschine (VM) wurde von Gerald J. Popek und Robert P. Goldberg in ihrer Arbeit "Formal Requirements for Virtualizable Third Generation Architectures." definiert. Diese lautet wie folgt.

> "Eine virtuelle Maschine ist ein effizientes, isoliertes Duplikat einer realen Maschine."[1]

Dabei gibt es verschiedene Architekturen, um die virtuellen Maschinen zu betreiben. Die virtuellen Maschinen werden dabei mittels eines Hypervisors oder auch Virtuell-Maschine-Monitor (VMM) verwaltet. In der Doktorarbeit von Robert P. Goldberg "Architectural Principles for Virtual Computer Systems"[29] definiert dieser zwei Typen von Hypervisoren.

[1] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974

- **Typ I**

  Beim Typ I arbeitet der Hypervisor direkt auf der Hardware der unterliegenden Maschine. Die virtuellen Maschinen werden dann auf diesem betrieben.

- **Typ II**

  Der Typ II Hypervisor läuft als Anwendung auf einem Betriebssystem. Dieses kann neben des Hypervisores auch eigene Programme ausführen.

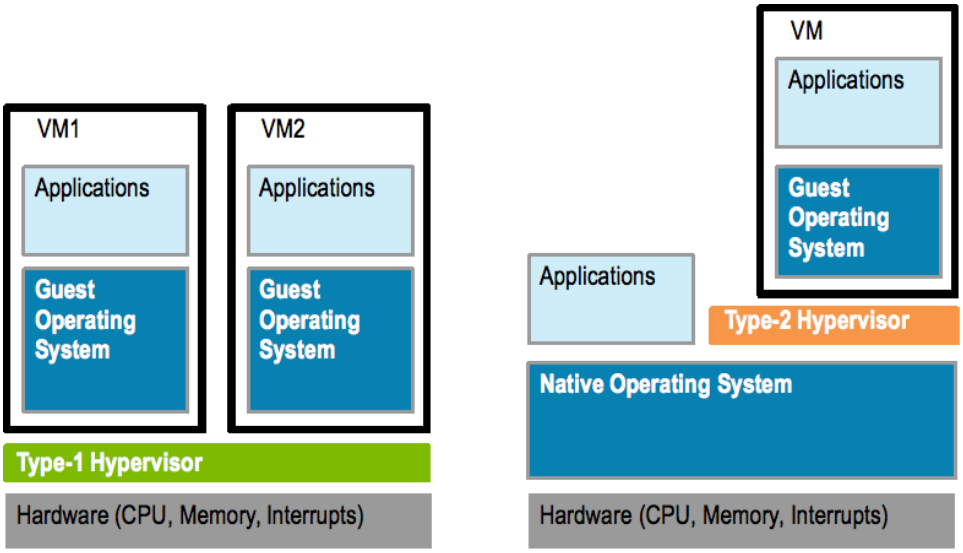

Abbildung 1: Übersicht über Type I und Type II Hypervisoren [4]

In der Abbildung 1 sind beide Typen von Hypervisoren veranschaulicht.
Je nach Hersteller sind Implementierungen für beide Typen vorhanden. Zum Beispiel verwendet Microsoft in seiner Implementierung *Hyper-V* [35] den Typ I, genauso wie die Umsetzung von VMware mittels *ESX/ESXi* [21] oder *Xen* [22] der Firma Citrix Systems, Inc. [8].
Dazu gibt es Umsetzungen von Typ II Hypervisoren bei verschiedenen Herstellern. Beispiele dazu sind von VMware die Anwendungen *VMware Workstation* [21] oder von *Oracle* mit ihrer *VirtualBox* [37].
Um die virtuellen Maschinen auf den Hypervisoren zu verwalten, wird eine Software benötigt. Ein Beispiel für solch eine Software zum Verwalten von virtuellen Maschinen ist OpenStack. [15]. Diese wird folgend genauer beschrieben, da sie für in dieser Arbeit verwendet wird.

2.1.1 OpenStack

"OpenStack ist eine Software zum Verwalten von vielen Ressourcen für Berechnungen, Speicherung und Netzwerken in einem Rechenzentrum, welches über das OpenStack-Dashboard oder die OpenStack API verwaltet wird." [2]

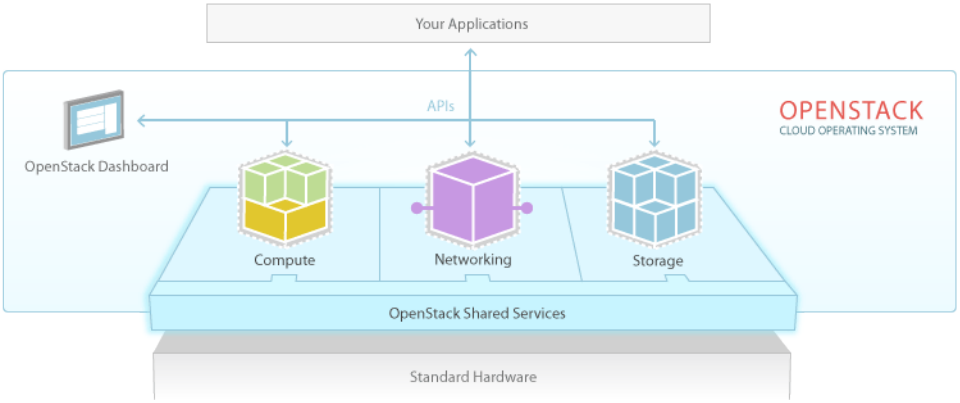

Abbildung 2: Aufbau von OpenStack [15]

Eine Übersicht über den Aufbau von OpenStack zeigt die Abbildung 2. Dabei kann OpenStack mit dem *NOVA*-Modul als Typ II Hypervisor arbeiten und verwaltet selber die Knoten, auf denen die virtuellen Maschinen laufen. Es besteht auch die Möglichkeit, mit anderen Typ I Hypervisoren, wie *Hyper-V* [35], *ESX/ESXi* [21] oder *Xen* [22], zu arbeiten, welche die VMs ausführen. Die zu verwaltenden Knoten (Hypervisoren) werden dabei in OpenStack in Zonen unterteilt. Eine VM wird immer einer Zone zugeteilt, in welcher OpenStack diese starten und verschieben kann. Dabei ist es möglich, mit Openstack ein Cluster in verschiedene Zonen einzuteilen, um beispielsweise nach Funktionalitäten oder Verfügbarkeiten zu unterteilen.
OpenStack ist für seine Aufgaben in verschiedene Kernmodule unterteilt. Darunter befindet sich die Verwaltung von verteilten Daten mittels *SWIFT*. Das Modul *KEYSTONE* übernimmt die Authentifizierung bei OpenStack. Die Verwaltung und Manipulation der virtuellen Maschinen wird mit dem *NOVA*-Modul umgesetzt. Für die Verwaltung der virtuellen Netzwerke wird das *NEUTRON*-Modul verwendet. Daneben wird das *CINDER*-Modul verwendet, damit der Speicher beim Betrieb der VMs persistent bleibt. Dies muss sichergestellt sein, wenn die laufende VMs bei Manipulationen, wie

[2] Openstack. http://www.openstack.org/, (besucht Dezember 2016)

dem Migrieren oder Pausieren, geblockt sind. Das Bereitstellen von Images zum Erstellen der virtuellen Maschinen wird mit Hilfe von *GLANCE* umgesetzt. Daneben gibt es noch viele optionale Module wie Beispielsweise zum Bereitstellen von Datenbanken, Verwenden der direkten Hardware eines Servers oder Ermitteln von Telemetrie-Daten.
Der für diese Arbeit erstellte Prototyp nutzt die Kernmodule *KEYSTONE* und *NOVA*. Das Kernmodul *KEYSTONE* wird angesprochen, um sich beim OpenStack-System zu authentisieren. Des weiteren wird das *NOVA*-Modul verwendet. Dies wird angesprochen, nachdem sich der Prototyp bei Openstack autorisiert hat, um die VMs zu manipulieren, die Knoten zu verwalteten und Information zu den VMs abzurufen.

2.1.2 Vorteile von virtuellen Maschinen

Es gibt verschiedene Vorteile, warum virtuelle Maschinen in der Industrie, Universitäten oder Heimbereich eingesetzt werden können. Ein paar davon werden folgend dargestellt und kurz beschrieben.[31]

- **Partitionierung**

  Eine Anwendung, welche in der virtuelle Maschine ausgeführt wird, lastet eine reale Maschine nicht vollständig aus. Deswegen ist es möglich, die Ressourcen der realen Maschine, wie den Hauptspeicher, die Kerne der Prozessoren oder den eventuell vorhandenen Grafikkarten, aufzuteilen. Somit ist es möglich, dass mehrere VMs auf einer realen Maschine laufen können.

- **Kapselung**

  Jede Komponente einer virtuelle Maschine, sowie deren Betriebssystem und die dort laufenden Anwendungen werden als Dateien gespeichert. Dadurch besteht die Möglichkeit, dass virtuelle Maschinen einfach erstellt, kopiert oder verschoben werden können.

- **Isolation von Usern**

  Durch die Unabhängigkeit der einzelnen Maschinen beeinflussen sich diese nicht gegenseitig. Sollte eine virtuelle Maschine hängen bleiben oder mit einem Virus befallen ist, werden andere VMs, welche auf der selben realen Maschine laufen, nicht betroffen sein.

- **Hardwareunabhängigkeit**

  Die unterliegende Hardware ist unabhängig von der verwendeten VM, da diese nur mit dem Hypervisor zusammen arbeitet. Somit können die virtuellen Maschinen einfach verschoben werden, wenn die aktuelle Hardware, etwa durch Alterserscheinungen oder Defekte, ausgetauscht werden muss.

Damit die Vorteile genutzt werden können, kann für jede Anwendung, wie Beispielsweise Datenbanken, E-Mail-Server, Datenspeicherung oder virtuelle Desktops, eine eigene virtuelle Maschine erstellt werden. Der Vorteil der Partitionierung wird beispielsweise im universitären Betrieb eingesetzt, da ein stärkerer realer Server gekauft wird und damit mehrere virtuelle Maschinen, etwa für Arbeitsgruppen und/oder Forschung, bereitgestellt werden. Der Vorteil der Kapselung tritt bei allen Anwendungen von virtuellen Maschinen auf, da dies ein Kernkonzept dieser ist. Es werden beispielsweise Kopien der VMs als Backup erstellt. Dagegen tritt die Isolation vor allem beim Einsatz von Workspace-as-a-Service[30] auf. Dabei bekommt jeder Nutzer einen eigenen virtuellen Desktop und wenn dieser befallen oder hängen geblieben ist, sind die anderen virtuellen Desktops durch die Isolation nicht betroffen. Ein Beispiel für den Vorteil der Hardwareunabhängigkeit ist, dass die unterliegende Hardware ausgetauscht werden kann, wenn diese zu langsam wird, veraltet oder defekt ist. Die virtuelle Maschine muss nun nur übertragen werden und kann ohne Komplikationen ihren Dienst fortsetzen.

2.1.3 Nachteile von virtuellen Maschinen

Es entstehen eventuelle extra Kosten, wenn nur ältere Hardware vorhanden ist, welche nicht über die notwendige Leistung verfügt, um den Betrieb von VMs zu gewährleisten. Dazu können Probleme bei der Lizenzierung von Software, welche in der VM laufen, entstehen. Diese sind etwa an Benutzer, Betriebssystem, MAC-Adressen oder andere Hardware-Komponenten gebunden. Vor allem bei an Hardware gebunden Lizenzen kann es zu Problemen kommen, wenn die VM verschoben wird.
Des weiteren gibt es Anwendungen, welche Grafik-, Speicher und/oder Rechenintensiver sind. Dazu gehören beispielsweise High-Performance-Computing-Anwendungen. Durch die Nutzung von virtuellen Maschinen entsteht ein Overhead, welcher Ressourcen verwendet, welche nicht für die Anwen-

dung zur Verfügung stehen und diese daher nicht optimal mit allen Ressourcen arbeiten kann. In Abschnitt 3.3 wird auf bestehende Probleme von High-Performance-Computing-Anwendungen erläutert und darunter wird auch das Ausführen von diesen in VMs beschrieben.

2.2 High-Performance-Computing

Es gibt keine eindeutige Definition für High-Performance-Computing (HPC), jedoch kann vereinfacht gesagt werden, dass HPC eine Bezeichnung für komplexe Berechnungen sind, welche nicht auf typischen Desktop Computern, wie sie etwa im Heim- oder Officebereich eingesetzt werden, effizient durchgeführt werden können. Dazu werden sogenannte HPC-Cluster verwendet, welche genauer in Abschnitt 2.4.5 beschrieben werden.
HPC-Anwendungen werden zum Beispiel in den Bereichen Wissenschaft, Medizin oder in der Wirtschaft eingesetzt. Ein Beispiel dabei sind Klimaberechnungen, welche im Deutschen Klimarechenzentrum [10] durchgeführt werden. Es werden aber auch kleine rechenintensive Anwendungen oder Anwendungen zum Analysieren von Daten [33] als HPC bezeichnet. Beispiele dazu sind auch beim Zedat, dem Hochschulrechenzentrum der Freien Universität Berlin, zu finden. [23] Genauso auch in den laufenden und abgeschlossenen Projekten des Fachbereich Mathematik und Informatik der Freien Universität Berlin. [17]
Da HPC-Anwendungen oft Informationen untereinander austauschen, werden die Knoten im Cluster über ein eigenes Kommunikationsnetzwerk verbunden, um es von der Verwaltung zu trennen. Des weiteren werden HPC-Anwendungen auch als HPC-Jobs oder im Kontext dieser Arbeit nur als Jobs beschrieben. Um die Jobs zu verwalten und auf den einzelnen Knoten im Cluster zu starten, wird eine Software benötigt. Ein Beispiel für eine Software zum Verwalten von HPC-Anwendungen ist Slurm [18], welches in dieser Arbeit verwendet wird.

2.2.1 Slurm

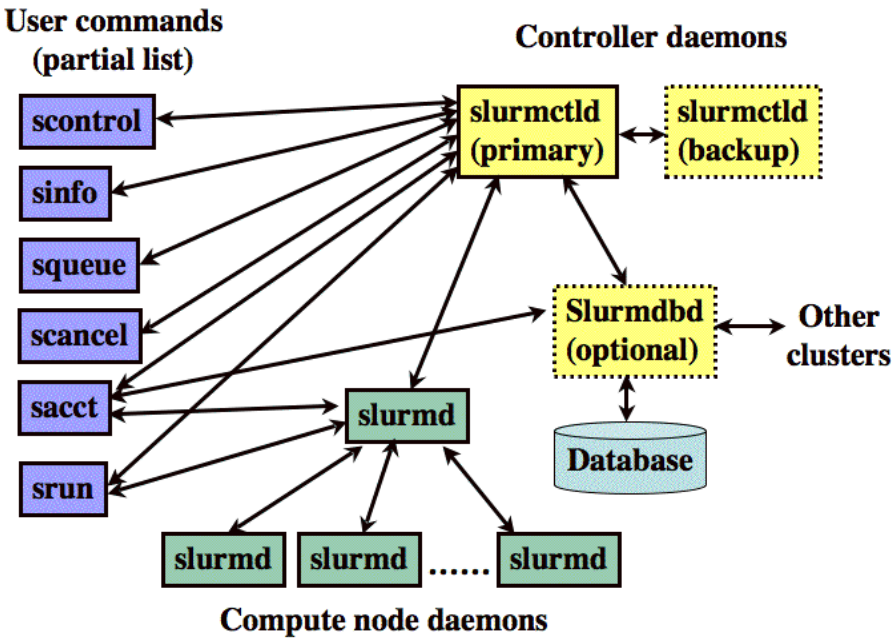

Abbildung 3: Slurm Architektur [18]

"Slurm steht für `Simple Linux Utility for Ressource Management` und ist ein Opensource, fehlertolerantes und hoch skalierbares Cluster-Management und System zum Verteilen von Jobs auf großen oder kleinen Linux-Clustern."[3] Der Aufbau von Slurm ist in Abbildung 3 dargestellt. Mit Hilfe des *slurmctld*-Daemon werden die einzelnen Knoten im Cluster verwaltet. Auf den einzelnen Knoten läuft dabei der *slurmd*-Daemon. Dieser übernimmt das Starten der HPC-Anwendungen.
Die Knoten werden von Slurm in Partitionen unterteilt. Diese können einen oder mehrere Knoten enthalten, je nachdem, wie Slurm konfiguriert ist. Jede Partition kann mehrere Warteschlangen für die Jobs besitzen, welche je nach Größe der Jobs oder benötigter Zeit sortiert sein können. Dabei werden die HPC-Jobs, meist mit einer angegebenen oder reservierten Startzeit sowie einer Endzeit oder der maximalen Dauer der Berechnung angeben.

[3] Slurm. http://slurm.schedmd.com/, (besucht Dezember 2016)

2.3 Unterschiede VMs und HPC-Anwendungen

Die Unterschiede von virtuellen Maschinen und HPC-Anwendungen liegen vor allem in ihrer Verwendung. Virtuelle Maschinen laufen meist über eine längere Zeit, wie beispielsweise ein Email-Server, SQL-Server oder Server, welche andere Dienste bereitstellen. Einige davon müssen durchgängig erreichbar sein und stellen ihre Funktionalität nur für Updates nicht zu Verfügung. Auf der anderen Seite stehen HPC-Anwendungen, welche nur für bestimmte Berechnungen und/oder für bestimmte Zeiten ausgeführt werden. Diese Zeiten werden meist fest angegeben oder können absehbar angenommen werden.
Ein anderer Unterschied ist die Auslastung der Hardware, welche die virtuellen Maschinen benötigen. Diese Belastungen sind je nach Anwendung meist konstant gering oder können etwa bei Web-Anwendungen, durch die Zugriffe auf diese und damit den Ressourcenverbrauch pro Zugriff, gemessen werden. Die Auslastung ist damit abschätzbar für virtuelle Maschinen. Dadurch ist es möglich, dass mehrere virtuelle Maschinen optimal auf einem physikalischen Server laufen und nicht nur eine virtuelle Maschine pro Server, wenn diese nicht alle Ressourcen verwendet. Die HPC-Anwendungen benötigen jedoch je nach Anwendungen für ihre Berechnungen oft alle Ressourcen, wie Recheneinheiten des CPUs, Hauptspeicher oder die Grafikkarte, eines Computers und lasten den Computer damit vollkommen aus. Daher sollte keine andere Anwendung zu dieser Zeit auf den Knoten laufen.

2.4 Cluster

Virtuelle Maschinen und High-Performance-Computing-Anwendungen werden häufig in Clustern betrieben. In dieser Arbeit geht es zudem darum, dass auf einem Cluster diese beiden Anwendungsmöglichkeiten zusammen betreiben werden. Daher erfolgt nun die Definition von Clustern, sowie deren Typen und speziell der Aufbau im Kontext von virtuellen Maschinen und High-Performance-Computing.
Im Allgemeinen betrachtet ist ein Cluster der Zusammenschluss von verschiedenen Komponenten, welche miteinander in Beziehung stehen, um mit diesen spezifische Aufgaben zu erledigen. Beschrieben wird dieses ausführlich zum Beispiel für die Zusammenarbeit von Unternehmen in dem Buch "Competitive Advantage of Nations" [39] von Michael E. Porter 1990.

Im Fall der vorliegenden Arbeit bestehen die Komponenten aus Computern, welche über ein Netzwerk verbunden sind, über welches sie kommunizieren. Zu den Aufgaben der einzelnen Computer gehört zum Beispiel das Ausführen von virtuellen Maschinen, Jobs von HPC-Anwendungen oder anderen Diensten. Die folgende Definition beschreibt die hier verwendeten Cluster.

> "Die grundlegende Form ist ein System aus zwei oder mehr Computern oder Systemen (Knoten genannt), welche eine Anwendung ausführen oder andere Aufgaben. Dabei ist das System transparent, also für einen Außenstehenden sieht es aus, als wenn er nur mit einer Ressource arbeitet. Die Schlüsselmerkmale sind dabei Zuverlässigkeit, Verteilung der Auslastung und die Performance." [4]

Dabei gibt es verschiedene Typen von Clustern, welche folgend kurz beschrieben werden. [28]

2.4.1 High-Availability- oder Failover-Cluster

Bei einem *High-Availability-* oder *Failover-Cluster* wird darauf geachtet, dass die Funktionalität der Dienste, wie beispielsweise E-Mail- oder Web-Server, im Cluster dauerhaft zur Verfügung stehen beziehungsweise erreichbar sind. Dazu werden Backup-Kopien dieser Dienste als Server oder virtuelle Maschine bereit gehalten und gestartet oder aufgeweckt, wenn die aktuell laufenden Dienste nicht mehr zur Verfügung stehen oder durch Analyse des Zustandes der Hardware festgestellt wird, dass diese bald ausfällt. Dabei stehen im Cluster mehrere Knoten bereit, welche den gleichen Dienst zur Verfügung stellen. Falls ein Knoten oder eine Anwendung ausfällt, kann dann ein anderer Knoten die Arbeit übernehmen und je nach Umsetzung ohne Datenverlust weiter arbeiten. Zum Beispiel wird dies für Datenbanken oder Nachrichten gemacht, welche immer erreichbar sein sollen.

[4] Pravin Ganore. Cluster Computing: Definition and Architecture of a Cluster. https://www.esds.co.in/blog/cluster-computing-definition-and-architecture-of-a-cluster/, März 2014 (besucht Dezember 2016)

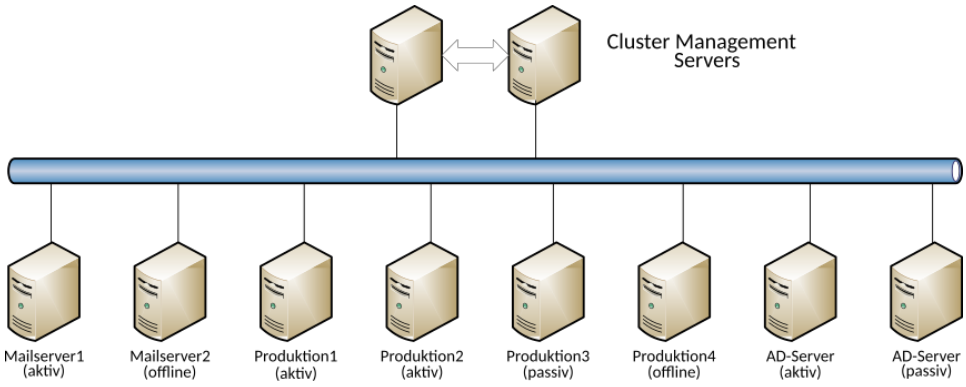

Abbildung 4: High Availability Cluster Beispiel

Die Abbildung 4 zeigt dazu ein reales Beispiel. Es sind mehrere Cluster-Server zur Verwaltung vorhanden, welche erkennen sollen, ob ein Dienst zur Verfügung steht und im Notfall die Knoten, welche ausfallen können oder ausgefallen sind, abzuschalten und die neuen Knoten zu starten oder anzubinden. In der Abbildung sind diese als aktiv *(gestartet)*, passiv *(nicht angebunden)* oder offline *(ausgeschaltet)* dargestellt sind. Beispielsweise gibt es zwei E-Mail-Server, von denen einer ständig aktiv ist. Sollte einer ausfallen, kann der zweite aktiviert werden. Da E-Mails, je nach Arbeit, weniger relevant sind, muss der zweite nicht im passiven Zustand warten und kann erst später aktiviert werden. Anderes sieht es bei dem Produktionsservern aus. Im Beispiel laufen zwei dauerhaft und einer ist auf passiv und kann im Notfall sofort verwendet werden, wenn einer der aktiven Produktionsserver ausfällt. Sollte dies der Fall sein, kann der vierte Produktionsserver, welcher bisher offline war auf passiv gestartet werden, sollte noch ein Server ausfallen. Ein reales Beispiel dazu wäre die Produktion einer Zeitung. Die Produktionsserver sind dabei die Server zum Bearbeiten der Zeitungsseiten und hierbei stehen immer zwei zur Verfügung. Wichtig sind in diesem Beispiel, Abbildung 4, auch die AD-Server (Active-Directory-Server), also die Server zum Verwalten der Benutzer im Netzwerk. Hierbei sollte immer einer aktiv sein und der zweite sofort zugeschaltet werden können. Sollte der erste Server ausfallen, kann keiner der Nutzer mehr arbeiten, da diese sich nicht anmelden können.

2.4.2 Load-Balance-Cluster

Bei den *Load-Balance-Clustern* wird darauf geachtet, dass Anfragen an die Dienste im Cluster gleichmäßig auf alle Knoten aufgeteilt wird. Damit kann ein Service von mehr Nutzern verwendet werden, als ein einzelner Server bearbeiten kann. Dies kommt vor allem bei großen Webanwendungen zum Einsatz, welche tausende Anfragen pro Sekunde bearbeiten müssen. [28] In der Abbildung 5 ist ein Beispiel für den Aufbau eines Load-Balance Clusters dargestellt. Dabei verteilt der Load-Balancer die kommenden Anfragen auf die jeweils freien Knoten.

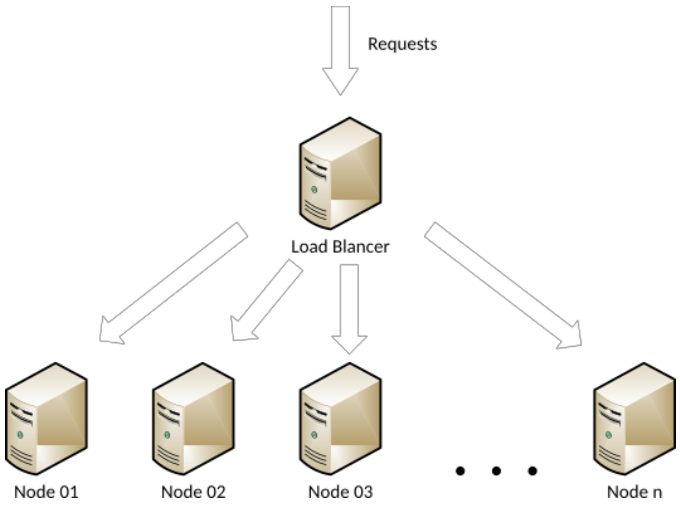

Abbildung 5: Load-Balance Cluster Beispiel Aufbau

2.4.3 High Availability und Load-Balance Cluster

Hierbei werden beide Clustertypen zusammen verwendet. Zum einen werden die Aufgaben auf mehrere Knoten verteilt und zum anderen stehen für den Ausfall von Knoten oder Anwendungen noch weitere Knoten bereit. Diese können bei Ausfall die Aufgaben übernehmen.

2.4.4 Cluster für virtuelle Maschinen

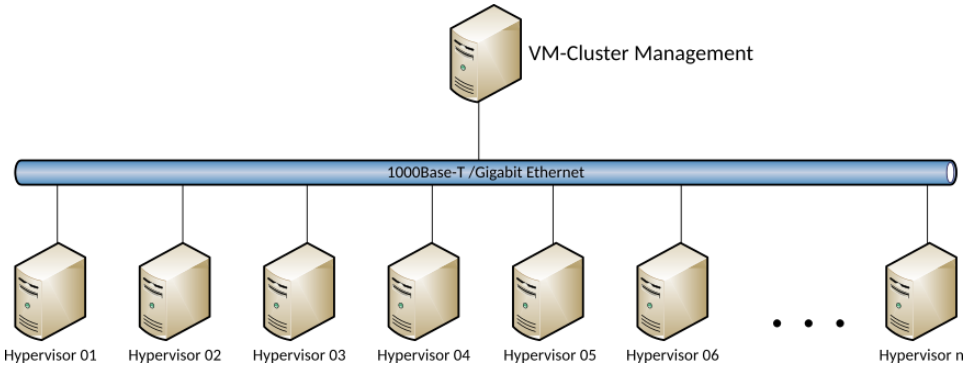

Abbildung 6: Aufbau des Clusters für virtuelle Maschinen

Die Cluster für virtuelle Maschinen sind ähnlich aufgebaut, wie im Abschnitt 2.4.1 für *High Availability* oder *Failover Cluster* und in Abbildung 6 dargestellt. Der Unterschied dabei ist, dass auf den einzelnen Knoten ein Hypervisor läuft und die Anwendungen oder Dienste laufen in einzelnen virtuellen Maschinen statt direkt in dem Betriebssystem der Knoten. Dazu gibt es einen oder mehrere Knoten, welche das Cluster verwalten. Diese sind für das Verwalten, wie dem Starten oder Stoppen sowie Migrieren, der VMs auf den jeweiligen Knoten zuständig. Dadurch können die Vorteile der VMs, welche im Abschnitt 2.1.2 beschrieben sind, genutzt werden. Zum Bespiel durch den Vorteil der Hardwareunabhängigkeit, ist es möglich ein *High-Aailability*-Cluster durch VMs umzusetzen, da jederzeit neue VMs mit dem gleichen Dienst gestartet werden können, falls Ausfälle vorkommen.

2.4.5 Cluster für HPC-Anwendungen

Die Ursprünge eines Clusters für HPC-Anwendungen finden sich im *Beowulf Project* [7]. Das Projekt wurde ab 1993 von Donald Becker and Thomas Sterling entwickelt und 1994 am Goddard Space Flight Center der NASA [1] gestartet. Beschrieben wurde es in "BEOWULF: A Parallel Workstation For Scientific Computation"[44]. Dabei wird das Cluster wie folgt definiert.

> Ein Beowulf Cluster besteht aus dedizierten PCs oder Workstations, welche High-Performance-Computing Aufgaben ausführen können. Die Knoten des Clusters stehen nicht bei Schreibtischen, sondern jeder einzelne führt nur Aufgaben des Clusters aus. Sie

sind normalerweise nur über einen einzelnen Knoten mit der Außenwelt verbunden. [5]

Bei einem HPC-Cluster sind ein oder mehrere Knoten zum Verwalten vorhanden. Diese verteilen und starten die HPC-Anwendungen auf den einzelnen Knoten im Cluster. Dazu gibt es je nach Größe des Clusters mehrere Knoten (Nodes), welche dafür ausgelegt sind, zum Beispiel durch die Möglichkeit MPI-Anwendungen zu starten, Anwendungen für HPC auszuführen.

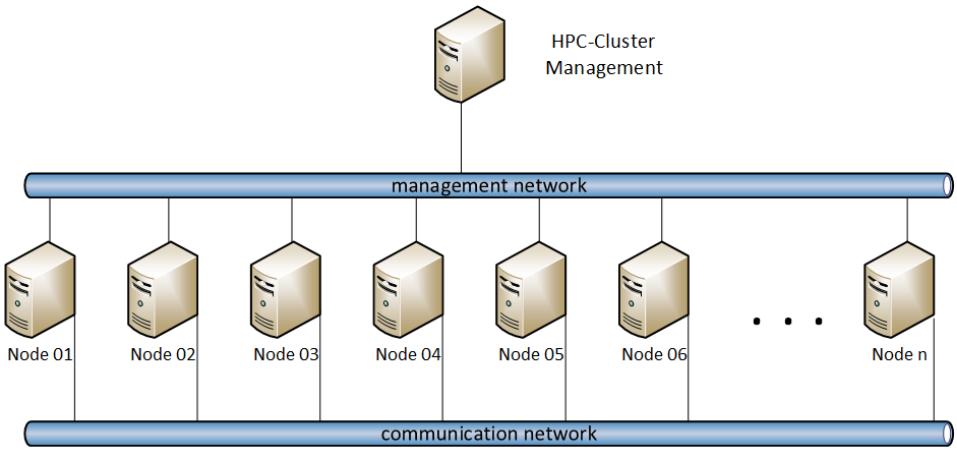

Abbildung 7: HPC Cluster Beispiel

Ein Beispiel eines Aufbaus des Clusters ist in Abbildung 7 dargestellt. Der HPC-Cluster-Management-Knoten nimmt die Aufgaben für die Berechnungen an und verwaltet die einzelnen Knoten (Node). Auf diesen werden die Berechnung durchgeführt. Diese sind zum einen über ein Verwaltungsnetzwerk *(management network)* verbunden, um mit dem Management-Knoten zu kommunizieren und die Anwendungen zu verteilen. Außerdem sind die Knoten über ein Kommunikationsnetzwerk *communiction network*, welches über Gigabit Ethernet oder InfiniBand [2] angebunden ist. Dieses wird für den Austausch von Informationen der Knoten untereinander verwendet. Mit diesem sind die Knoten mittels 10 Gigabit pro Sekunde und mehr miteinander verbunden sowie einer geringeren Latenz. Genaueres wird in Abschnitt 2.4.6 zu den Unterschieden zwischen VM- und HPC-Cluster erläutert. Das System zusammen betrachtet wird auch als Supercomputer bezeichnet, da es für Anwender transparent dargestellt ist. Für die Anwender sieht es aus,

[5] Beowulf Project. http://www.beowulf.org, (besucht Dezember 2016)

als ob nur ein Computer dahinter steht. Beispiele für Supercomputer und HPC-Cluster sind in der Abbildung 8 dargestellt, welche die Top drei unter den Top 500 von November 2016 zeigen. Dabei steht *Cores* in der Abbildung für Prozessoren. Ein Knoten der Supercomputer kann dabei mehrere Prozessoren besitzen. Weitere sind in den Top 500 [6] aufgeführt.

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209

Abbildung 8: Die ersten drei der Top 500 Supercomputer von November 2016 [6]

2.4.6 Unterschiede VM- und HPC-Cluster

Die Unterschiede beider Cluster sind vor allem bei der Netzwerkanbindung der einzelnen Recheneinheiten beziehungsweise Knoten. In einem VM-Cluster werden die einzelnen Server meist nur mit 1000Base-T oder bis 10 Gigabit Ethernet, beschrieben in *IEEE Std 802.3* [26], verbunden, welches nur bis zu 10 Gb/s Datenübertragung ermöglicht. Für die virtuellen Maschinen ist dies auch genügend, weil die Anwendungen auch selten untereinander Daten austauschen. Jedoch tauschen HPC-Anwendungen oft Daten miteinander aus. Die Datenrate sollte deshalb möglichst hoch sowie die Latenz zwischen den einzelnen Knoten möglichst gering sein. Damit können die Anwendungen ohne Verzögerungen rechnen. Dafür wird zum Beispiel InfiniBand [2] verwendet, welches 10 Gb/s und mehr Übertragungsrate ermöglicht oder 10 bis 100 Gigabit Ethernet. Eine Untersuchung zeigt, dass die Latenz zwischen Gigabit Ethernet etwa 6.8 mal größer ist als bei InfiniBand [3]. Die hohen Kosten der jeweiligen Netzwerktechnologien führen dazu, dass nicht alle HPC-Cluster auf InfiniBand setzen. Unter den Top 500 der Supercomputer etwa setzen in der Aktualisierung von Juni 2016 218 auf InfiniBand und 208 auf Gigabit Ethernet. [27]

Ein weiterer Unterschied liegt in der Anbindung von externen Speichern von beiden Systemen. Virtuelle Maschinen nutzen meist entweder NAS (Networked-Attached Storage) oder SAN (Storage-Area-Networks). Dabei wird mit NAS der Speicher per TCP/IP, beschrieben in RFC 1180 [43], angebunden und diese werden als Dateien zur Verfügung gestellt. Bei SAN wird mittels Fibre Channel [12] oder iSCSI Protokoll, spezifiziert in RFC 3720 [41], angebunden. Der Vorteil beider Speicherarten ist, dass der Speicher von mehreren Maschinen gemeinsam genutzt werden kann und das System skalierbar in Kapazität sowie Performance ist. Dagegen nutzen HPC-Cluster oft Speicherverwaltungen der Firma DDN (DataDirect Networks)[9]. Dabei ist wichtig, dass die Daten mit hoher Datenübertragung und geringer Latenz angesprochen werden können. Dazu werden die Daten nicht als Dateien sondern als Objekte zur Verfügung gestellt.

	Object Storage	File Storage
Performance	Performs best for big content and high storage throughput	Performs best for smaller files
Geography	Data can be stored across multiple regions	Data typically needs to be shared locally
Scalability	Can scale infinitely to petabytes and beyond	Potentially scales up to millions of files, but can't handle more
Analytics	Customizable metadata, not limited to number of tags	Limited number of set metadata tags

Abbildung 9: Unterschiede zwischen Object-Storage und File-Storage [32]

Die Unterschiede zwischen Datei (File) oder Objekt (Object) Speicher ist in Abbildung 9 dargestellt. Dabei ist für HPC-Anwendungen vor allem die Performance und die Skalierbarkeit wichtig, da die Daten schnell zur Verfügung gestellt werden müssen und HPC-Anwendungen mit vielen Daten arbeiten.

3 Aufgabe

Nachdem, zur Verständnis der Aufgabe in dieser Arbeit, die Begrifflichkeiten definiert und erklärt wurden, wird die Aufgabe in diesem Abschnitt erläutert. Im ersten Abschnitt 3.1 wird die Aufgabe genau beschrieben. Danach werden die Voraussetzungen für die Umsetzung der Aufgabe, in Abschnitt 3.2, dargelegt. Daraufhin werden bestehende Probleme in Bezug zur Aufgabenstellung, in Teil 3.3, beschrieben. Zuletzt werden die Anforderungen an den Prototyp, welcher für die Aufgabe entwickelt wird, in Abschnitt 3.4, aufgelistet und kurz erläutert.

3.1 Beschreibung der Aufgabe

Die Aufgabe dieser Arbeit ist, die Machbarkeit zu testen, ob ein Cluster für High-Performance-Computing-Anwendungen und eines für das Betreiben von virtuellen Maschinen kombiniert werden können. Dazu wird der Prototyp HIVICMA, welches für `High Performance Computing and virtuell Maschine Combination Management System` steht, entwickelt. Mit diesem werden Tests durchgeführt, um die Machbarkeit zu zeigen. Des Weiteren wird bei diesen Tests über die Machbarkeit dieses Zusammenspiels diskutiert und welche Voraussetzungen dafür notwendig sind, dieses umzusetzen. Danach wird der Lösungsansatz für den Prototyp beschrieben und folgend die Beschreibung der Tests, welche mit diesem durchgeführt werden. Dann erfolgt eine Auswertung der Testergebnisse sowie die Erläuterung eventueller Probleme, welche auftreten können. Zum Schluss wird die Aufgabe der Machbarkeit der Kombination der Clusters, mit Hilfe des Prototyps bewertet und eine Aussicht auf weiterführende Entwicklungen in diesem Gebiet gegeben.

3.2 Voraussetzungen

Als eine Voraussetzung für die Verwaltung des Clusters wurde vorher festgelegt, dass HPC-Anwendungen priorisiert werden sollen. Dies wurde festgelegt, da die Ergebnisse der Berechnungen der HPC-Anwendungen zeitnah benötigt werden und daher wenn möglich sofort durchgeführt werden sollten. Daraus folgt, dass laufende virtuellen Maschinen eventuell ausgeschaltet oder verschoben werden müssen, wenn diese auf Knoten laufen, welche von den HPC-Anwendungen verwendet werden. Aus dieser Voraussetzung folgt

die nächste Voraussetzung. Es wird ein Bereich der vorhandenen Knoten benötigt, welcher nur für virtuelle Maschinen genutzt wird. Dieser Bereich ist notwendig, da nicht alle VMs ausgeschaltet werden können und die laufenden VMs ihre Funktionalität weiterhin behalten sollen. Es wird dabei erst einmal vernachlässigt, dass der Bereich überlastet sein kann, wenn zu viele VMs darauf laufen.
Als Softwarevoraussetzungen für die Umsetzung ist zum einem Slurm [18] vorgegeben, für die Verwaltung der HPC-Anwendungen. Zum anderen ist für die Verwaltung der virtuellen Maschinen die Software OpenStack [15] vorgegeben. Diese Softwarevoraussetzungen sind jedoch nur für diesen Prototypen bindend und könnten auch durch ähnliche Anwendungen ausgetauscht werden. Beide Anwendungen wurden gewählt, da sie an der Freien Universität Berlin im Fachbereich Informatik eingesetzt werden. Eine genaue Beschreibung dieser Anwendungen befinden sich in den jeweiligen Abschnitten in den Definitionen, in Abschnitt 2.

3.3 Bestehende Probleme

Derzeit werden Cluster zum Beispiel entweder für virtuelle Maschinen oder HPC-Anwendungen verwendet. Es ist zwar möglich, dass die HPC-Anwendungen in den virtuellen Maschinen laufen, jedoch können diese, je nach Implementierung und Aufgabe nicht effizient arbeiten. Durch die Virtualisierungsschicht mit dem Hypervisor entsteht ein Overhead. Dieser zusätzlicher Ressourcenverbrauch, welcher entsteht, kann nicht für die Aufgaben verwendet werden. Dies ist zum Beispiel durch VMware in der Arbeit "Virtualizing HPC and Technical Computing with VMware vSphere" [5] gezeigt. Dabei wird zwar eine Verbesserung ihrer Technologie dargestellt, jedoch zeigen sie dabei auch, dass bei Message-Passing-Interface-Anwendungen (MPI [24]) immer noch ein Overhead entsteht auch wenn dieser nur noch bei unter 10% und manchmal unter 5% liegt.
Hinzu kommt, dass viele der heutigen HPC-Anwendungen auch mit der GPU rechenintensive Berechnungen durchführen. Dies wird gemacht, da die GPU im Gegensatz zur CPU, über mehr parallele Recheneinheiten verfügen. Der Unterschied dazu wird in der Abbildung 10 aus dem Artikel "Was ist GPU computing?" [36] dargestellt.

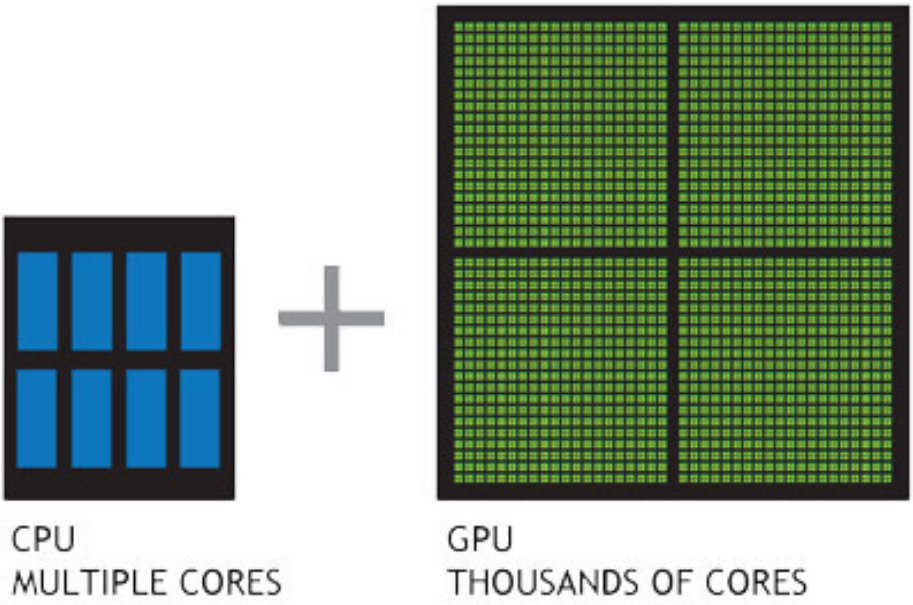

Abbildung 10: Vergleich CPU und GPU [36]

Ein Problem bei dem Verwenden einer GPU von mehreren VMs ist, dass dadurch die Isolation der virtuellen Maschinen verletzt wird und somit eine Möglichkeit entsteht, dass Informationen zwischen virtuellen Maschinen ausgetauscht werden können. Dies stellt ein Sicherheitsrisiko dar, welches den Vorteil der Kapselung von VMs verletzt. Gezeigt wird dies in der Arbeit "Confidentiality issues on a GPU in a virtualized environment" [34].
Um doch die vorhandenen Grafikkarten verwenden zu können, wurde die *GPU Pass-Through*-Methode entwickelt. Dabei wird jeder virtuellen Maschine eine eigene Grafikkarte zur Verfügung gestellt, welche nur von dieser verwendet werden kann. Dadurch entsteht zwar kein Sicherheitsrisiko über die Grafikkarte selber, jedoch muss auch sichergestellt sein, dass bei mehreren Grafikkarten in einem Rechner diese auch einen separaten Bereich im Hauptspeicher bekommen, wenn diese den verwenden kann. Ansonsten könnte darüber immer noch ein Sicherheitsrisiko entstehen. Dieses System ist außerdem sehr unflexibel, da die Grafikkarte nicht anders verwendet werden kann, wenn die VM die Grafikkarte gerade nicht verwendet.
Eine Ansatz, mit welchen VMs HPC-Anwendungen ausführen, wird in der Arbeit "Benchmarking Bare Metal Cloud Servers for HPC Applications" [40] verfolgt. Dort werden die Maschinen mittels dem OpenStack Modul

Ironic[16] als Bare Metal Maschine neu gestartet. Das bedeutet, die VM soll direkt mit der Hardware der realen Maschine arbeiten, da diese alleine auf der Maschine läuft. Dabei bleibt die Hypervisor-Schicht jedoch weiterhin bestehen und somit weiterhin der Overhead, welcher durch die zusätzliche Schicht, wie vorher beschrieben, zusammenhängt. Außerdem müssen die Hersteller der Hypervisoren spezielle Treiber dafür entwickelt haben. Mit diesen soll das geladene Image der virtuellen Maschine, den vollen Zugriff auf die Hardware erhalten. Durch die zusätzliche Hypervisor-Schicht unterscheidet sich diese Version von der in dieser Arbeit beschriebenen, da diese nicht vorhanden ist.

3.4 Anforderungen von HIVICMA

Durch die Kombination von virtuellen Maschinen und High-Performance-Computing-Anwendungen ergeben sich die folgende Anforderungen an HIVICMA. Diese wurden aufgestellt um die Kombination beider Clusterarten zu ermöglichen. Anhand dieser wurde der Prototyp entwickelt. Außerdem dienen sie im Fazit dazu, um festzustellen, ob die Aufgabenstellung erfüllt wurde.

- **Cluster verwalten**

  Das Wissen, welche Knoten im Cluster vorhanden sind, um damit zu arbeiten. Darunter sind zum einem die Knoten, welche für HPC-Anwendungen genutzt werden und zum anderen die Knoten, welche von der VM-Verwaltung genutzt werden. Dazu gehört auch das Erstellen des nur für die VMs reservierten Bereichs, falls die anderen Knoten von HPC-Anwendungen genutzt werden.

- **HPC Job Informationen beschaffen**

  Beschaffen der Information über die HPC-Anwendungen, welche aus der HPC-Verwaltung geholt werden.

- **VM Informationen beschaffen**

  Die Informationen über die virtuellen Maschinen, welche von der VM-Verwaltung verwaltet und bereitgestellt werden.

- **Analysieren der HPC-Job Informationen**

  Die Informationen, welche vorher geholt worden sind, werden analysiert, um damit das Cluster zu verwalten.

- **Verschieben von virtuellen Maschinen**

  Die Möglichkeit bereitstellen, dass VMs von einem Knoten auf einen anderen Knoten verschoben werden können.

- **Vorbereiten von Knoten im Cluster für die HPC-Jobs**

  Mit Hilfe der analysierten Information die Knoten für den Betrieb von virtuellen Maschinen zu sperren. Außerdem die, auf den benötigten Knoten, laufenden virtuellen Maschinen auf freie Maschinen verschieben oder in den reservierten Bereich.

- **Knoten wieder für virtuelle Maschinen zur Verfügung stellen**

  Nachdem HPC-Jobs abgearbeitet wurden, werden die verwendeten Knoten wieder für die Benutzung mit VMs zur Verfügung gestellt.

4 Lösungsansatz

Die Idee hinter der hier beschriebenen Lösung ist, dass einzelne Knoten im Cluster entweder für HPC-Anwendungen oder für VMs genutzt werden. Die VMs werden dafür entweder ausgeschaltet, wenn diese nicht mehr benötigt werden, oder auf Knoten verschoben, welche nicht für die HPC-Anwendung benötigt werden. Dies ist notwendig, da wie in Abschnitt 2.3 beschrieben ist, HPC-Anwendungen die volle Leistung eines Servers benötigen und daher keine virtuelle Maschine auf diesem laufen soll.
Für die Umsetzung der Verwaltungssoftware werden unterschiedliche Informationen benötigt. Dazu wird zuerst in Abschnitt 4.1 beschrieben, welche Informationen von der Software für die Verwaltung von HPC-Anwendungen benötigt und wie diese verwendet werden. Danach wird in Abschnitt 4.2 erläutert, welche Informationen von der VM-Verwaltungssoftware benötigt und wie diese verwendet werden. Im Anschluss wird der Aufbau von HIVICMA in Abschnitt 4.3 dargestellt. Zum Schluss werden die Abläufe der Programmteile von HIVICMA in Teil 4.4 beschrieben.

4.1 HPC-Informationen

Zum Verwalten des Clusters werden die Informationen der einzelnen HPC-Anwendungen benötigt. Dazu gehört erst einmal ein Überblick über die Knoten, welche von der Verwaltungssoftware für die HPC-Anwendungen verwaltet werden. Es sind auch die Informationen der einzelnen Jobs, welche aus der oder den Warteschlangen, wie beispielsweise beschrieben für Slurm, in Abschnitt 2.2.1, geholt werden müssen, nötig. Da immer wieder neue Jobs hinzugefügt werden können, muss in regelmäßigen Abständen nachgeschaut werden, ob neue Jobs hinzugekommen oder vorhandene Jobs geändert worden sind. Dazu wird gespeichert, wann die Warteschlange zuletzt aktualisiert wurde, um unnötige Neuberechnungen zu vermeiden. Je nach HPC-Verwaltungssoftware kann es möglich sein, einen Handler zu übergeben, welcher die Clusterverwaltung informiert, wenn neue HPC-Anwendungen hinzugefügt wurden.
Aus den Informationen der einzelnen HPC-Anwendungen werden die zur Verwaltung wichtigen Informationen extrahiert. Darunter befinden sich die Knoten (Node), welche der Job für seine Berechnung verwendet, sowie die Zeit, wann der Job gestartet wird und wenn möglich das voraussichtliche

Ende oder die Dauer, um damit das Ende des Jobs zu berechnen.

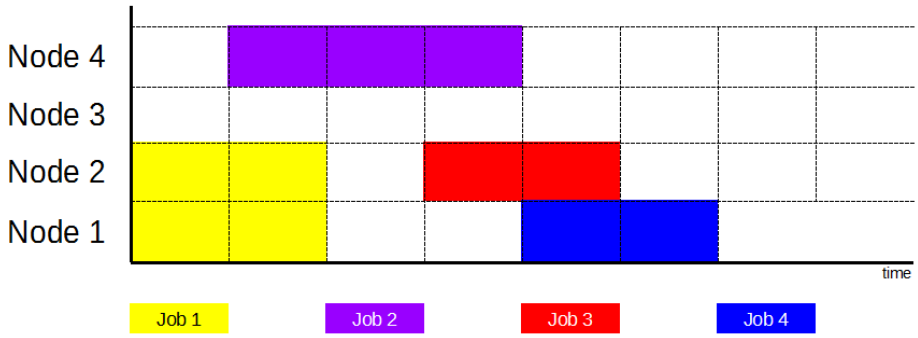

Abbildung 11: Beispiel eines Aufbaus der Events zu Knotenstruktur

Aus diesen Informationen der einzelnen Jobs wird eine Struktur wie in Abbildung 11 aufgebaut. Die einzelnen Jobs sind hierbei farbig dargestellt und die Abstände der Jobs soll die Zeiten für den Start und das Ende signalisieren. Im Beispiel sind die Zeiten in festen Zeiteinheiten eingeteilt. Dieses muss jedoch nicht zwingend notwendig sein und ist von der HPC-Verwaltungssoftware vorgegeben. Jeder Job wird dazu analysiert, um dabei zu ermitteln, auf welchen Knoten dieser gestartet werden soll oder schon gestartet ist. Anhand der Startzeiten wird dann diese Struktur aufgebaut. Die Zeiten, wann ein Job endet, können beim Aufbau vernachlässigt werden. Sie werden jedoch gespeichert, da diese für die folgende Analyse benötigt werden. Anhand dieser werden einige der benötigten Timer erstellt. Aus der aufgebauten Struktur ist es nun möglich, die Zeiten zu berechnen, an denen die Knoten für VMs verwendet werden können oder nicht. Sollten alle Knoten, welche für HPC-Anwendungen genutzt werden können, verwendet werden, müssen die virtuellen Maschinen, auf die, in den Anforderungen aus Abschnitt 3.4 beschrieben, nur für VMs reservierten Knoten verschoben werden.

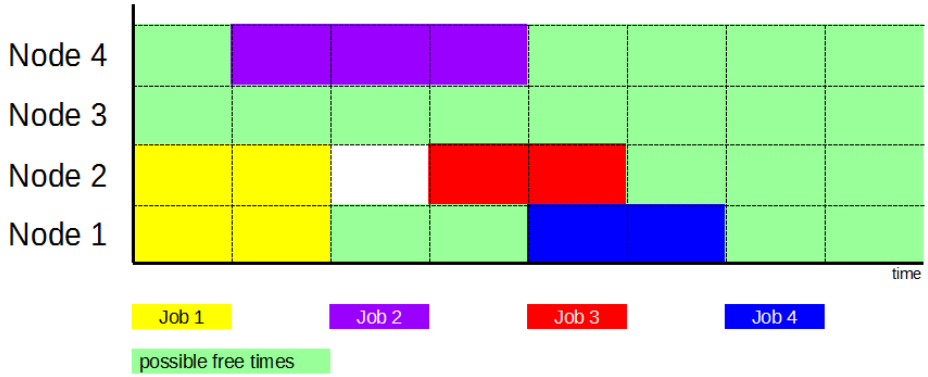

Abbildung 12: Aufbau der freien Zeiten zur Knotenstruktur

Die Struktur der freien Zeiten ist ähnlich der vorherigen und in Abbildung 12 dargestellt. Um die freie Zeiten zu errechnen, ist es notwendig zu wissen, wann ein Job gestartet wird und wann dieser abgeschlossen ist. Die minimalen freien Zeiten, müssen vorher festgelegt werden. Dazu wird vorher definiert, dass wie auf Knoten 2 dargestellt die Zeit zwischen Job 1 und 3 nicht ausreichend lange ist und daher nicht als frei markiert ist. Im Beispiel ist als weiße Fläche in der Abbildung gekennzeichnet, wenn ein Knoten nicht für den VM Betrieb freigegeben wird. Es ist beispielsweise nicht effizient, wenn zwischen zwei HPC-Anwendungen nur eine Stunde dieser Knoten nicht genutzt wird, diesen freizugeben, da er nach kurze Zeit wieder von VM befreit werden müsste. Die möglichen freien Zeiten werden als grüne Flächen dargestellt. Die Verwaltung der virtuellen Maschinen kann innerhalb dieser Zeiten die Knoten nutzen.
Es ist außerdem notwendig zu wissen, wann ein Knoten für eine HPC Anwendung benötigt wird. Vor diesem Zeitpunkt muss der Verwaltung der VMs der Knoten entzogen werden, sodass dort keine VMs mehr gestartet werden können. Danach ist es notwendig die zu verwendeten Knoten, soweit vorhanden, von den dort laufenden VMs zu befreien. Da dies einige Zeit dauern kann, muss der Vorgang dafür, je nach Menge der laufenden VMs, mehrere Minuten vorher angestoßen werden. Dies wird genauer in der Einstellung der Timer für die Tests in Abschnitt 5.3.5 erläutert.
Der Vorgang, die Jobs in der Warteschlange zu überprüfen, muss ständig wiederholt werden. Sollten nicht alle Knoten von HPC-Anwendungen genutzt werden, wir ein fester Zeitabstand gewählt, zu welchem diese Überprüfung

stattfindet. Sollten jedoch alle Knoten von HPC-Anwendungen gleichzeitig benutzt werden, wird dieser Vorgang kurz vor dem Zeitpunkt für das festgelegte Ende der kürzesten Anwendung gelegt. Dieses Ende eines Jobs beziehungsweise die Dauer und damit die Möglichkeit das Ende zu berechnen, wird, wie in der Beschreibung von Slurm, in Abschnitt 2.2.1, erläutert, beim Erstellen des Jobs festgelegt. Dies ist wichtig, um zu ermitteln, ob die Knoten danach noch einmal verwendet werden oder freigegeben werden können. Welche Werte für dieser Zeitpunkt genutzt werden, wird genauer in den Beschreibungen für die Test in Abschnitt 5.3 dargestellt.
Der Ansatz der HPC-Verwaltung einen Handler zu übergeben, wird hier nicht verfolgt, da Slurm nicht die Möglichkeit bietet. Dieser Handler würde ausgeführt, wenn ein neuer Job in die Warteschlange hinzugefügt oder geändert wird. Dadurch könnte die Auslastung reduziert werden, weil nicht ständig abgefragt werden müsste, ob neue Jobs hinzugefügt wurden.

4.1.1 Timer erstellen

Anhand der Struktur, welche in Abbildung 12 dargestellt ist, werden die folgenden Timer erstellt. Diese sind die Zeitpunkte, bis zu denen der Prototyp schlafen gelegt werden kann, da keine Aufgaben bearbeitet werden müssen. Das Listing 1 enthält den Aufbau der Timer und die möglichen Information, welche für jeden Timer gespeichert werden.

```
struct Timer {
 int - Art des Timers
 time - Zeitpunkt, bis wann HIVICMA schlafen kann
 list - Knoten, welche mit dem Timer zusammenhängen
}
```

Listing 1: Aufbau der Timer

Dabei werden drei Arten von Timern erstellt. Folgend werden diese genauer beschrieben.

- **HPC_CHECK_TIMER**

  Dieser Timer wird gesetzt, damit die Verwaltungssoftware für die HPC-Anwendungen abgefragt wird, wie der aktuelle Stand seiner Warteschlange für die Jobs ist. Das geschieht in regelmäßigen Abständen,

wenn die HPC-Anwendungen nicht alle Knoten verwenden oder sollten alle Knoten verwendet werden, wird dieser gesetzt, kurz bevor die kürzeste HPC-Anwendung endet.

- **FREE_HPC_NODES**

  Nachdem eine HPC-Anwendung beendet ist, können die verwendeten Knoten wieder für die VMs zur Verfügung gestellt werden. Um dies umzusetzen wird dieser Timer kurz nach dem Beenden der HPC-Anwendung gesetzt.

- **FREE_VM_NODES**

  Da auf den Knoten, welche eine HPC-Anwendung benötigt, eventuell VMs laufen, gibt es diesen Timer. Er wird gesetzt, damit zum einen die Knoten aus der VM-Verwaltung entfernt werden, also keine neuen VMs dort gestartet werden können. Zum anderen dient er dazu, um die dort laufenden VMs auf andere Knoten, welche von keiner HPC-Anwendungen genutzt werden, zu verschieben.

In den beiden Timern *FREE_HPC_NODES* und *FREE_VM_NODES* werden zusätzlich zum Zeitpunkt des Timers und dessen Art, die verwendeten Knoten mit gespeichert. Diese müssen für die Verwendung von VMs freigegeben beziehungsweise gesperrt und von laufenden VMs befreit werden. Die Zeiten, welche in den Timern gesetzt werden, sind in der Beschreibung der Tests, in Abschnitt 5.3, erläutert.

4.2 Informationen von der VM-Verwaltung

Für die Verwaltung werden verschiedene Information von der VM-Verwaltung benötigt. Darunter befinden sich die Knoten, welche von Openstack, der VM-Verwaltungssoftware, welche bei diesem Prototyp verwendet wird, verwaltet werden. Daraus lassen sich, mit den Knoten, welche vorher aus HPC-Verwaltung Slurm, dass in diesem Prototyp genutzt wird, ermittelt wurden, Überschneidungen finden. Sollten sich, in der Menge der Knoten von Openstack, Knoten befinden, welche nicht von Slurm verwendet werden, können diese für den in den Anforderungen 3.4 definierten Bereich genutzt werden. Ansonsten müssen aus beiden Sets von Knoten, ein paar ermittelt werden, welche nur für den Betrieb von virtuellen Maschinen genutzt werden. Diese Knoten werden benötigt, falls eine HPC-Anwendung

alle restlichen Knoten im Cluster benötigt, aber nicht alle VMs abgeschaltet werden können. Die VMs werden dann auf diese verschoben.

4.2.1 Information für Timer

Wenn der *FREE_VM_NODES*-Timer ausgeführt wird, dann werden zuerst die Knoten, welche von den VMs befreit werden müssen, aus der VM-Verwaltung entfernt, damit dort keine neuen gestartet werden. Danach werden die Informationen benötigt, welche VMs derzeit gestartet sind und auf welchen Knoten diese laufen. Dazu gehören die Informationen wie der Name der VMs in der VM-Verwaltung und der Knoten, auf denen die VMs gestartet sind. Aus diese Informationen lässt sich ermitteln, welche virtuelle Maschinen verschoben werden müssen.

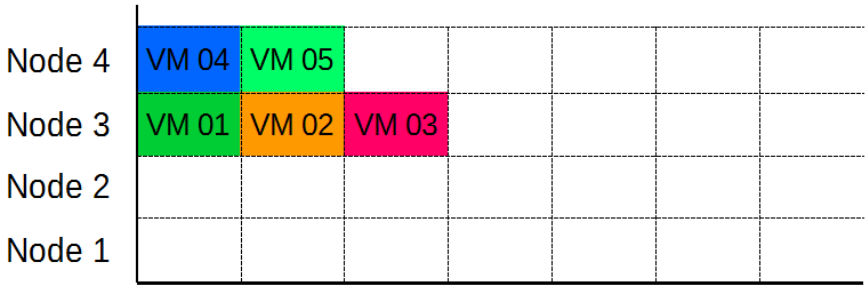

Abbildung 13: Beispiel der laufenden virtuellen Maschinen.

Ein Beispiel dazu ist in Abbildung 13 dargestellt. Wenn eine Anwendung nun Knoten (Node) 3 benötigt, müssen die virtuelle Maschinen *VM 01*, *VM 02* und *VM 03* auf einen freien Knoten verschoben werden, beispielsweise Knoten 1 oder 2.
Sollten alle Knoten für HPC-Anwendungen verwendet werden, dann werden die VMs auf die reservierten Knoten verschoben.
Es besteht noch die Möglichkeit, welche in dem Prototyp nicht umgesetzt wurde, dass die virtuelle Maschinen, beim *FREE_HPC_NODES*-Timer, wieder auf die freigewordenen Knoten verteilt werden. Da dies für diesen Test der Machbarkeit der Kombination beider Cluster Nutzungen nicht relevant ist, wird es nicht berücksichtigt. Dies sollte jedoch getan werden, wenn die reservierten Knoten zu überlastet sind und somit die VMs nicht ihre volle Leistung erbringen können, als wenn sie wieder jeweils auf einzelnen Knoten

laufen, wie es vorher war. Für diesen Vorgang müsste eine Routine entwickelt und der Prototyp erweitert werden. Dazu sind noch mehr Informationen, wie etwa die Auslastung des Hauptspeichers oder der Prozessoren der einzelnen Knoten notwendig. Es wird jedoch erst einmal nicht berücksichtigt, da es für die Machbarkeit der Kombination der Cluster nicht relevant ist, sondern eher in den Bereich des Ausblicks, in Abschnitt 6.2, fällt.

4.3 Aufbau von HIVICMA

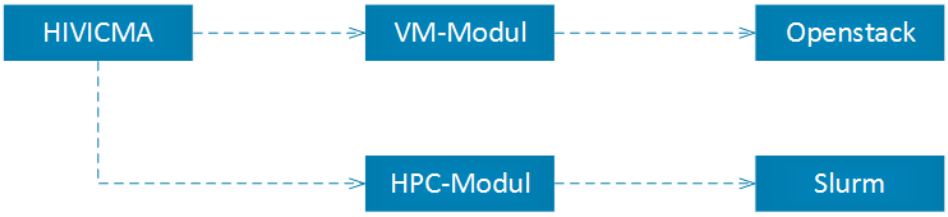

Abbildung 14: Aufbau des Lösungsansatz

Der grobe Aufbau von HIVICMA mit seinen Modulen ist in Abbildung 14 dargestellt. Theoretisch könnte die Funktionen des VM-Moduls sowie des HPC-Moduls in *HIVICMA* integriert sein. Es wurde sich jedoch für diese Variante entschieden, da die Module individuelle Befehle für Openstack beziehungsweise Slurm beinhalten und somit die Module austauschbar sind. Somit wäre es möglich, andere Module für andere Verwaltungssysteme über die selbe API anzusprechen.
Das *HIVICMA*-Paket spricht dabei das VM-Modul sowie das HPC-Modul an, um die entsprechenden Informationen zu erhalten. Diese wiederum sprechen OpenStack beziehungsweise Slurm an, um die jeweiligen Information anzufordern oder Manipulationen durchzuführen. Die notwendigen Information werden dabei nur in *HIVICMA* selbst gespeichert. Mit den gespeicherten Informationen werden dann die Timer erstellt und je nach Timer, siehe Abschnitt 4.1.1, die entsprechenden Aufgaben abgearbeitet. In der restlichen Zeit legt sich das Programm schlafen, um keine Last auf dem Server zu erzeugen, bis die Zeit des aktuellen Timers abgelaufen ist.

4.3.1 HPC-Modul

Zur Verwaltung der HPC-Anwendungen wird das HPC-Modul benötigt. Dies stellt zum einen Funktionen bereit, um die verwendeten Knoten zu

ermitteln, sowie deren Manipulation durchzuführen, also dem Entfernen der Knoten, damit diese nicht für den Einsatz von HPC-Anwendungen genutzt werden können. Damit ist es möglich, wie in den gestellten Anforderungen aus Abschnitt 3.4, reservierte Knoten für virtuelle Maschinen zu erstellen. Außerdem wird eine Funktion für das Abfragen der Warteschlangen, um die Jobinformation zu bekommen, erstellt. Sollte jedoch ein Handler an die HPC-Verwaltung übergeben werden können, dann wird solch eine Funktion für diese Übergabe benötigt, da diese wie vorher beschreiben derzeit nicht umgesetzt wurde. Dieser wird ausgeführt, wenn ein neuer Job hinzugefügt wird.

4.3.2 VM-Modul

Wie für die HPC-Verwaltung wird auch für die VM-Verwaltung ein Modul benötigt, um mit der VM-Verwaltung zu kommunizieren. Dieses benötigt, wie bei der HPC-Verwaltung, Funktionen zum Manipulieren von Knoten, welche von der VM-Verwaltung genutzt werden. Darunter existiert eine Funktion, welche alle Knoten ermittelt, auf denen virtuelle Maschinen gestartet werden können. Außerdem gibt es die Möglichkeit, Knoten aus der VM-Verwaltung, für den Betrieb der VMs, zu entfernen. Um die Knoten für HPC-Anwendungen frei zu bekommen, werden Funktionen benötigt, die virtuellen Maschinen zu manipulieren. Dazu gehört die Möglichkeit, virtuelle Maschinen auf andere Knoten zu migrieren.

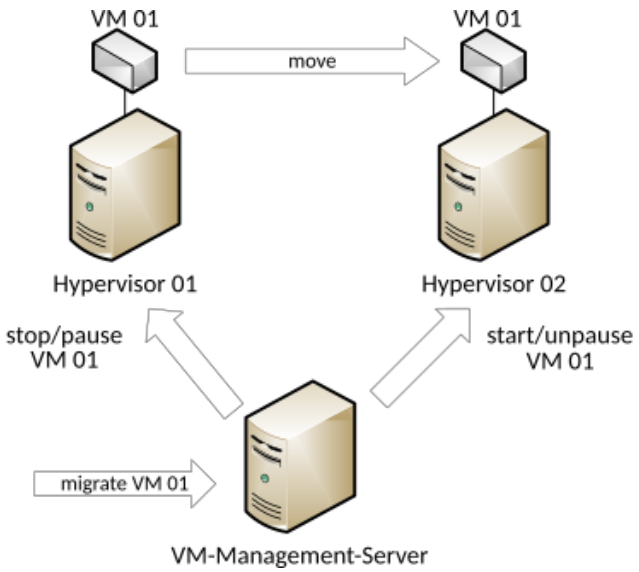

Abbildung 15: Klassische Migration von virtuellen Maschinen

VM-Migration Zwei Varianten zur Migration von virtuellen Maschinen stehen zur Verfügung. Zum einen ist dies die klassische Variante. Diese ist in Abbildung 15 dargestellt. Dabei wird die zu migrierende virtuelle Maschine ausgeschaltet oder sie pausiert. Nachdem diese ausgeschaltet ist oder pausiert hat, wird diese VM auf einen anderen Knoten verschoben. Danach wird die VM wieder gestartet beziehungsweise von der Pause aufgeweckt.

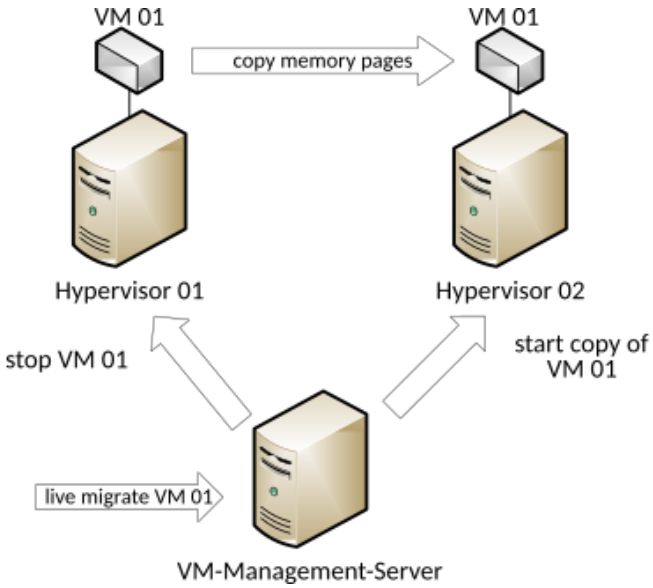

Abbildung 16: Live-Migration von virtuellen Maschinen

Die zweite Variante ist die Live-Migration. Diese ist in Abbildung 16 gezeigt. Dabei wird die virtuelle Maschine im laufenden Betrieb auf einen anderen Knoten verschoben. Die Verwaltung für die virtuelle Maschinen erhält den Befehlt, die VM 01 von dem Hypervisor 01 auf den Hypervisor 02 live zu migrieren. Dazu wird als erstes eine Kopie der VM auf dem Hypervisor 02 gestartet. Danach wird der Speicherinhalt von VM 01 des Hypervisors 01 auf den von Hypervisor 02 kopiert, damit die VM ohne Unterbrechung arbeiten kann. Im Anschluss wird die Netzwerkverbindung für die VM 01 auf den Hypervisor 02 umgelenkt. Nachdem dies geschehen ist, kann die VM auf Hypervisor 01 abgeschaltet werden. Die Benutzer dieser VM werden dadurch nicht, wie in der ersten Variante, beeinträchtigt.
Genaueres zur Verwendung der Migrationen bei HIVICMA, wird in der Beschreibung der Tests in Abschnitt 5.3 beschrieben.

4.4 Programmabläufe

Im Programm gibt es vier verschiedene Abläufe. Darunter ist als erstes der Start des Programms. Dieser wird in Listing 2 dargestellt.

```
1. Programm Start
2. hole Knoten von der HPC-Verwaltungssoftware
3. hole Knoten von der VM-Verwaltungssoftware
4. ermittele Knoten, welche beide verwenden und
   reserviere Knoten für die VMs
5. hole mögliche HPC Job Informationen
  • baue Job zu Knoten Struktur
  • analysiere Jobs und baue freie Zeiten Struktur
6. hole mögliche VM und baue Struktur zu VM-Verteilung
7. setze Timer anhand der ermittelten freien Zeiten,
   wenn VMs verschoben oder Knoten der VM-Verwaltung
   entfernt oder hinzugefügt werden müssen
8. setze Timer für nächsten Check in der HPC
   Verwaltungssoftware
9. schlafen legen
```

Listing 2: Ablauf des Start vom Programm

Daneben gibt unterschiedliche Abläufe, welche anhand der Timer ausgeführt werden. Als erstes wäre dort der *FREE_VM_NODES*-Timer, welcher kurz vor dem Starten eines Jobs ausgeführt wird. Dort müssen zuerst die Knoten von der VM-Verwaltung entfernt werden und virtuelle Maschinen von den

verwendeten Knoten verschoben werden. Listing 3 beschreibt die Schritte, welche durch bei diesem Timer durchgeführt werden.

```
1. Knoten aus der Timer Information holen, welche
   genutzt werden
2. Knoten aus der VM-Verwaltung entfernen
3. alle Information zu VMs von der VM-Verwaltung holen
4. VMs für Knoten sondieren, welche frei werden müssen
5. VMs auf freie Knoten verschieben
6. nächsten Timer holen
7. anhand des Timers schlafen legen
```

Listing 3: Ablauf vor dem Start eines Jobs

Des weiteren gibt es den Ablauf, welcher beim *FREE_HPC_NODES*-Timer, durchgeführt wird. Nachdem ein Job bearbeitet wurde und der oder die Knoten nicht mehr von einem anderen Job genutzt werden, beziehungsweise die Zeit zwischen zwei Jobs lange genug ist, können diese Knoten wieder für den VM Betrieb zur Verfügung gestellt werden. Dieser Ablauf wird in Listing 4 dargestellt.

```
1. verwendete Knoten aus dem Timer holen
2. Knoten in der VM-Verwaltung hinzufügen, damit diese
   die Knoten wieder nutzen kann
3. nächste Timer holen
4. anhand des Timers schlafen legen
```

Listing 4: Ablauf nach Beenden eines Jobs und Knoten frei wird

Zusätzlich zu diesen beiden Timern gibt es noch einen allgemeinen Timer. Dieser dient dazu, regelmäßig nachschauen, ob neue Jobs in der Warteschlange der HPC-Verwaltung vorhanden sind, um die bisherigen Timer zu aktualisieren. Dieser Timer kann eventuell durch einen Handler, welcher der HPC-Verwaltung übergeben werden kann, ersetzt werden. Dieser Handler könnte beim Eintreffen eines neuen Jobs gestartet werden und das Programm damit die HPC-Informationen nur abrufen, wenn es notwendig ist und somit wird das unnötige Aufwecken verhindert. Der Behandlung des HPC-Check-Timers ist in Listing 5 veranschaulicht

```
1. hole mögliche HPC Job Informationen
2. teste ob sich die Warteschlange verändert hat
3a. wenn Ja
  • baue Job zu Knoten Struktur
  • analysiere Jobs und baue freie Zeiten Struktur
  • erstelle neue Timer und ersetze alte
3b wenn nein setze neuen HPC-CHECK-Timer
4. nächsten Timer holen
5. anhand des Timers schlafen legen
```

Listing 5: Ablauf des allgemeinen Timers

5 Tests von HIVICMA

In diesem Kapitel geht es um die Tests, welche mit HIVICMA durchgeführt wurden, um die Anforderungen der Machbarkeit der Kombination der Beiden Clusternutzungsarten, welche in Abschnitt 3.4, zu belegen. Dabei wird zuerst in Abschnitt 5.1 auf den, von der Freien Universität Berlin zur Verfügung gestellten, Cluster eingegangen. Danach werden die Einstellungen und Voraussetzungen in Abschnitt 5.2 dargestellt. Im Anschluss werden in Teil 5.3 die durchgeführten Tests beschrieben. Die Testläufe werden danach in Abschnitt 5.4 erläutert. Am Schluss gibt es in Teil 5.5 ein kurze Auswertung der Testläufe.

5.1 Aufbau der Testumgebung

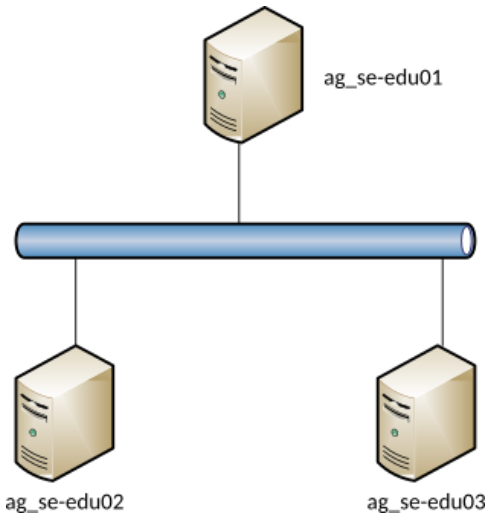

Abbildung 17: Aufbau des Clusters für den Test

Für die Tests wurde ein kleines Cluster von der Freien Universität Berlin zur Verfügung gestellt. Dieses Cluster ist in Abbildung 17 dargestellt. Dabei ist auf den Knoten ein Debian Linux mit der Version *3.16.36* installiert.
Der Knoten *ag_se-edu01* gilt dabei als Management-Server. Auf diesem Knoten sind alle Programme zum Verwalten des Clusters installiert. Darunter ist zum einen OpenStack, in der Version *Icehouse - 2014.1.3*, zum Verwalten der virtuellen Maschinen. Zum Verwalten der HPC Anwendungen ist die Slurm Anwendung in der Version *14.03.9* installiert. Die Knoten *ag_se-edu02* und *ag_edu-03* sind als Worker-Knoten definiert. Das bedeutet, auf ihnen laufen

entweder die HPC-Anwendungen und/oder virtuelle Maschinen, welche vom Management-Server verwaltet werden. Auf ihnen läuft dazu für Slurm der Service *slurmd* und für OpenStack der *nova-compute* Service. Für die Entwicklung von HIVICMA wurden außerdem noch folgende Bibliotheken auf dem Management Server installiert.

- **libcurl3-gnutl[13]**

  Dies wird verwendet, um die REST-Abfragen an OpenStack zu schicken. Diese kann auch durch anderen *libcurl* Bibliotheken ersetzt werden.

- **json-c[25]**

  Da für die REST-Anfragen an OpenStack JSON verwendet wird, ist zur Einfachheit halber eine Bibliothek ausgewählt worden, statt dies selbst zu implementieren. Aus den verschiedenen Bibliotheken wurde sich für json-c entschieden worden, weil der Einstieg intuitiv war und es nicht Bestandteil der Arbeit ist, sich weiter damit auseinander zu setzen.

- **libslurm[14]**

  Diese Bibliothek wird benötigt, um die API von Slurm zu verwenden. Sie wird mit <*slurm\slurm.h*> angesprochen.

5.2 Einstellungen und Voraussetzungen

Damit die Tests durchgeführt werden können, gibt es einige Einstellungen und Voraussetzungen, welche in Slurm und OpenStack durchgeführt und angenommen werden. Dazu gehört, dass nicht darauf geachtet wird, ob ein für VM reservierter Knoten eventuell überlastet ist und laufende VMs eventuell nicht effektiv arbeiten können. Außerdem muss die Anwendung, wie die Auswertung der Tests für das HPC-Moduls in Abschnitt 5.5.1 zeigt, mit Administrationsrechten gestartet werden.

5.2.1 Slurm

Zur Einfachheit halber wurde in Slurm nur eine Partition mit der *debug*-Partition eingestellt, welche die beiden Knoten *ag_se-edu02* und *ag_se-edu03* enthält. Damit wird nur diese Partition bei den Tests manipuliert.

```
#!/bin/bash -l
#
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -J test_job
#SBATCH -o output.txt
#SBATCH -e errors.log
#SBATCH -t 01:00:00
#

echo "Hello Slurm"
```

Listing 6: Verwendeter Job für Tests mit Slurm.

Der Job, welcher verwendet wurde, um die Slurm Funktionalitäten zu testen, ist in Listing 6 dargestellt. Die wichtigsten Einstellungen dabei sind *-t 01:00:00* und *-p debug*. Dies bedeutet, dass der Job eine Stunde dauert und auf der *debug*-Partition gestartet wird.

```
sbatch --begin=YYYY-MM-DD[THH:MM[:SS]] --nodelist=ag_se-edu02 test-job.sh
```

Listing 7: Befehl zum Starten eines Slurm-Jobs.

Der Befehl, um die Test-Jobs für Slurm zu starten, ist in Listing 7 dargestellt. Dabei wird mit Hilfe des Parameters *−−begin* angegeben, wann dieser Job gestartet werden soll. Der Parameter *−−nodelist* gibt an, dass der Job nur auf dem Knoten *ag_se-edu02* laufen soll. Diese Einschränkung wird durch die Größe des Test-Clusters festgelegt, da nur auf diesem die Jobs starten dürfen. Im Abschnitt 5.3.1 wird beschrieben, wo dieser Knoten ausgelesen wird.

5.2.2 OpenStack

Die Informationen für die virtuelle Maschinen werden nur für ein Projekt getestet, auch wenn es möglich und eines der Kern-Features von OpenStack ist, dass jeder Anwender ein eigenes Projekt besitzt. Für die Tests wird angenommen, dass ein globales Projekt für alle virtuellen Maschinen existiert. Außerdem ist es notwendig, dass der Nutzer, welcher sich über die API bei Openstack autorisiert, administrative Rechte besitzt, da das Migrieren von virtuellen Maschinen sowie das Hinzufügen oder Entfernen von Knoten, diese voraussetzen.

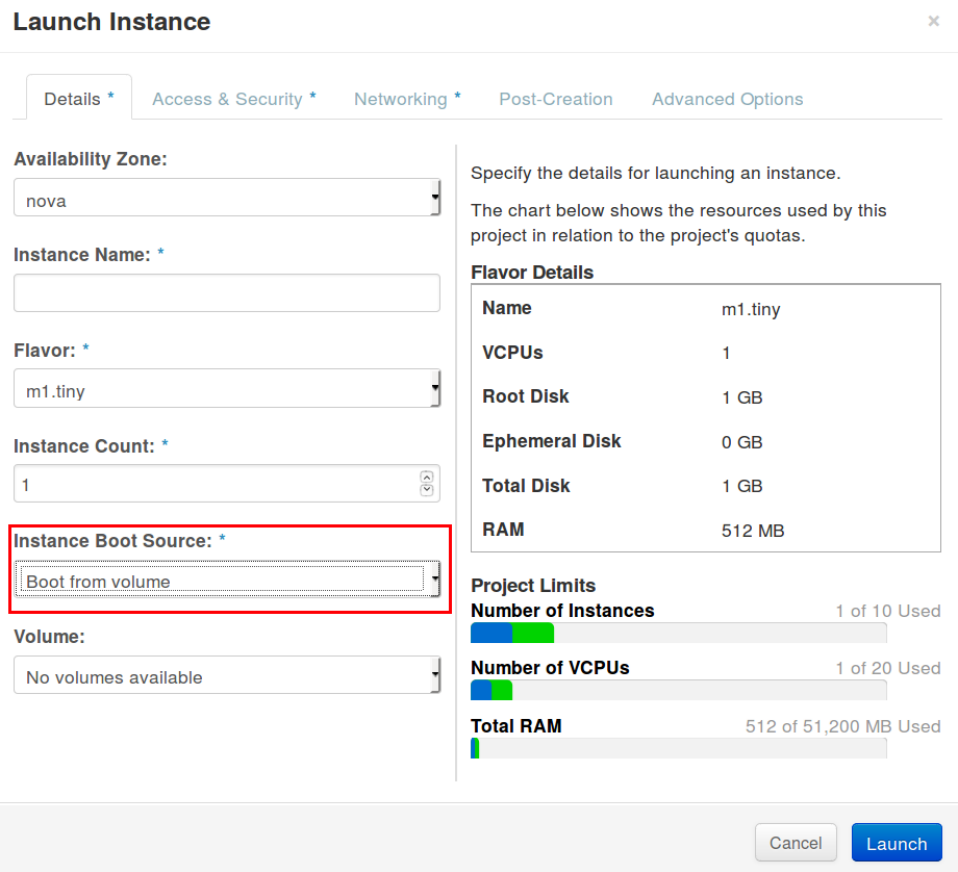

Abbildung 18: Einstellung beim Erstellen einer VM unter OpenStack.

Die Einstellung für das Erstellen von einer VM bei OpenStack ist in Abbildung 18 dargestellt. Es ist notwendig, in der Abbildung rot umrandet, dass beim Erstellen von virtuellen Maschinen mit einem *Volume* als Boot Quelle (Source) gearbeitet wird. Das ist wichtig, da ansonsten nicht die geplante Live-Migration durchgeführt werden kann. Für die Tests wurde, mit den angegebenen Einstellungen, eine virtuelle Maschine erstellt.

5.3 Beschreibung der Tests

In den nun folgenden Abschnitten werden die Tests beschrieben, welche mit dem Prototypen durchgeführt werden, um ein Cluster für HPC-Anwendungen und ein Cluster für VM zu kombinieren. Zuerst kommen die Tests der Module für das Ansprechen der Verwaltung der HPC-Anwendungen und der Verwaltung der virtuellen Maschinen. Danach werden die Tests beschrieben,

welche beide Module verbinden, um das Cluster zu verwalten.

5.3.1 HPC-Informationen ermitteln

Bei diesem Test wird ermittelt, ob alle benötigten Informationen über eine HPC-Anwendung erhalten werden können. Dieser Test dient zur Überprüfung der Anforderung *"HPC Job Informationen beschaffen"*. Dazu gehören die Informationen über die verwendeten Knoten sowie die Zeiten für den Start und das Ende des Jobs, wie in Abschnitt 4.1 beschrieben. Außerdem wird die Zeit der letzten Aktualisierung der Warteschlange von Slurm, ermittelt. Dazu wird, mittels der Slurm API, die Struktur aus dem Anhang 11 abgerufen. Diese enthält alle Informationen zu den aktuellen Jobs, sowie die Zeit der letzten Aktualisierung *time_t last_update*. Aus dieser wird die Struktur aus Anhang 10 zu den einzelnen Jobs geladen. In dieser Struktur sind die verwendetet Knoten in *char *req_nodes* gespeichert, welche im Abschnitt 5.2, beim Erstellen des Jobs, gesetzt wurden, sowie die Zeit für den Start in *time_t start_time*. Der Zeitpunkt des Endes wird mit Hilfe des *uint32_t time_limit* Eintrages, welcher in Minuten angegeben ist, berechnet. Dabei wird für den Endzeitpunkt der Startzeitpunkt plus dem Zeitlimit genommen.
Für die Tests werden Jobs erzeugt, welche von Slurm angenommen werden. Danach wird geschaut, ob alle benötigten Daten von den Jobs ermittelt werden können.

5.3.2 Knoten der HPC-Verwaltung ermitteln und manipulieren

Um die Knoten der HPC-Verwaltung zu ermitteln und gegebenenfalls zu manipulieren, dient dieser Test. Er wird verwendet, um zu zeigen, dass die Knoten, wie in der Beschreibung des Lösungsansatzes für HPC-Informationen aus Abschnitt 4.1 wurden, ermittelt werden und um die Anforderung *"Cluster verwalten"* zu überprüfen. Dabei ist es notwendig, dass alle Knoten bekannt sind, welche die HPC-Verwaltung verwendet, und dass Knoten eventuell entfernt werden können, um sie als reservierte Knoten für virtuelle Maschinen zu markieren.
Zur Umsetzung dieser Voraussetzungen werden, die in Abschnitt 2.2.1 beschriebenen Definition von Slurm, verwendeten Partitionen ermittelt und die Knoten extrahiert. Diese sind in der Struktur (siehe Anhang 12) gespeichert.

Die Struktur enthält für jede der Partition von Slurm eine Struktur (siehe Anhang 13). Aus dieser werden die Knoten, aus dem Eintrag *char *nodes*, ermittelt. Die selbe Struktur wird auch zum Manipulieren der Partitionen verwendet. Dabei wird der Eintrag der Knoten verändert, wenn Knoten, für den reservierten Bereich, entfernt werden. Der Aufbau des Tests ist in Listing 8 als Pseudocode dargestellt.

```
Funktion hpc_knoten_manipulations_test
  hole alle HPC Knoten
  entferne einen Knoten
  hole alle HPC Knoten
ende Funktion
```

Listing 8: Aufbau des Testablaufes für die Ermittlung und Manipulation der HPC-Knoten

Zuerst werden alle Knoten ermittelt, welche die HPC-Verwaltung verwendet. Danach wird ein Knoten entfernt und dann wieder alle Knoten ermittelt. Der entfernte Knoten sollte nicht mehr in der Liste der Knoten enthalten sein.

5.3.3 VM-Informationen ermitteln und manipulieren

Dieser Test soll feststellen, ob alle Informationen über eine virtuelle Maschine ermittelt werden können und ob die VMs manipuliert werden können. Er dient damit der Überprüfung der Anforderungen *"VM Informationen beschaffen"* und *"Verschieben von virtuellen Maschinen"*. Unter der Manipulation fällt das Stoppen, Starten, Pausieren, Aufwecken oder Migrieren einer VM auf einen anderen Knoten. Zuerst wird unter OpenStack eine virtuelle Maschine für die Tests erstellt. Danach werden die Informationen über diese ermittelt. Im Anschluss wird diese VM manipuliert. Es werden die verschiedenen Methoden jeweils über das Webinterface von OpenStack, sowie die von HIVICMA ermittelten Information geprüft, ob die jeweilige Methode erfolgreich war. Für den Test der Migration wird der jeweilige andere Knoten aus der Beschreibung des Testaufbaus in Abschnitt 5.1 genommen. Wenn die virtuelle Maschine auf *ag_se-edu02* gerade gestartet ist, wird diese auf *ag_se-edu03* migriert und umgekehrt. Dabei wurde der Einfachheit halber für HIVICMA nur die Live-Migration implementiert.

5.3.4 Knoten der VM-Verwaltung ermitteln und manipulieren

Da je nach Situation jeweils Knoten aus der VM-Verwaltung entfernt oder hinzugefügt werden müssen, dient dieser Test dazu, dies zu erproben. Damit werden die Anforderungen *"Vorbereiten von Knoten im Cluster für HPC-Jobs"* und *"Knoten wieder für virtuelle Maschinen zur Verfügung stellen"* überprüft. Im Test werden also zuerst geprüft, ob alle Knoten von OpenStack ermittelt werden können. Danach wird die Funktion abgeschaltet, damit der Knoten nicht mehr für OpenStack zur Verfügung steht. Im OpenStack-Webinterface wird danach geprüft, ob der Knoten in den Zonen nicht mehr für die Verwendung von VMs zur Verfügung steht. Nach diesem Test wird der Knoten der Zone wieder hinzufügt und geprüft, ob der Knoten von den Zonen wieder genutzt werden kann.

5.3.5 Kombination des HPC- und VM-Management-Systems

Die Tests, um beide Systeme zu kombinieren, bauen aufeinander auf, da jeweils verschiedene Schritte notwendig sind, um das ganze System zu testen. Zuerst wird getestet, ob die jeweiligen Knoten beider Systeme ermittelt werden können. Wenn alle Knoten ermittelt wurden, wird getestet, ob einer von der HPC-Verwaltung entfernt werden kann, um den reservierten Bereich für die virtuelle Maschinen zu erstellen. Danach wird geprüft, ob Knoten von der VM-Verwaltung entfernt und hinzugefügt werden können. Im Anschluss wird getestet, ob die HPC-Informationen ermittelt werden können. Dazu wird ein Test-Job gestartet. Aus diesem Schritt wird versucht, die im Lösungsansatz für HPC-Informationen vorgestellten (Abschnitt 4.1), Strukturen zu bauen. Dabei wird für die Zeit, welche zwischen zwei HPC-Anwendungen frei sein kann, um einen Knoten für VMs zu verwenden auf mindestens vier Stunden festgelegt. Es wird angenommen, dass bei weniger Zeit es unrentabel ist, diesen Knoten für VMs freizugeben.
Um danach die Timer zu erstellen und den Programmablauf zu testen, werden Dummy-HPC-Events erstellt. Dabei werden folgende Zeiten genommen.

- **CHECK_HPC_TIMER**

  Wenn nicht alle Knoten für ein HPC-Job verwendet werden, dann wird in einem Abstand von eine Stunde nachgeschaut, ob neue HPC-Anwendungen in die Warteschlangen von Slurm hinzugefügt wurden.

Es wurde eine Stunde gewählt, da angenommen wird, dass der Zyklus in denen neue HPC-Anwendungen hinzugefügt werden, hoch genug ist. Sollten jedoch alle Knoten verwendet werden, dann wird dieser Timer auf zehn Minuten vor dem Zeitpunkt für das angegebene Ende gesetzt. Es ist nicht nötig, wenn der Job länger als eine Stunde dauert, dass vorher nachgeschaut wird, ob neue Anwendungen hinzugefügt wurden, da diese unabhängig vom laufenden Job sind. Notwendig ist es jedoch vor dem Ende der Anwendung, da eventuell neue Jobs hinzugefügt wurden, welche die gleichen oder nur einen Anteil der Knoten benötigen und daher den *FREE_HPC_NODES*-Timer der gerade laufenden Anwendung überschreiben.

- **FREE_HPC_NODES**

  Nach einer HPC-Anwendung muss eventuell noch etwas aufgeräumt werden, also der verwendete Speicher oder die gesperrte Bereiche befreit werden. Daher wird dieser Timer erst zehn Minuten nach der angegebenen Endzeit der HPC-Anwendung eingestellt.

- **FREE_VM_NODES**

  Da eine Migration je nach Größe einer VM, wie im Artikel "Performance Analysis during Live Migration" [42] ermittelt, etwas über eine Minute dauert und außerdem auf einem Knoten mehrere VMs laufen können, wird die Zeit dafür auf etwa dreißig Minuten bevor eine HPC-Anwendung auf einen Knoten gestartet wird, gelegt. Dazu darf auch nicht die Auslastung im Netzwerk vernachlässigt werden, da diese den Vorgang verlangsamen kann.

Aus den drei Timer-Möglichkeiten ergeben sich folgende Tests. Zuerst einmal wird geprüft, ob der *CHECK_HPC_TIMER* in vorher regelmäßigen Abständen testet, ob neue Jobs in der Warteschlange stehen. Außerdem wird analysiert, ob dies geändert wurde, wenn alle Knoten verwendet werden. Dann wird geprüft, ob der *FREE_HPC_NODES*-Timer 10 Minuten nach dem Ende einer HPC-Anwendung die verwendeten Knoten wieder der VM-Verwaltung hinzufügt. Im Anschluss wird getestet, ob beim *FREE_VM_NODES*-Timer die von der HPC-Anwendung verwendeten Knoten von der VM-Verwaltung entfernt wurden und die darauf gestarteten Knoten verschoben wurden. Wie der Testaufbau in Abschnitt 5.1 beschreibt, stehen dafür nur zwei Knoten

zur Verfügung, deshalb werden nur auf einem HPC-Anwendungen gestartet und der andere steht als Knoten zum Ausweichen zur Verfügung.

5.4 Testläufe

In den folgenden Abschnitten werden die Durchführungen der Tests beschrieben, welche vorher in Abschnitt 5.3 erläutert wurden. Dabei werden zuerst die Testläufe für das HPC-Modul in Teil 5.4.1 erläutert. Danach sind die Testläufe für das VM-Modul in Abschnitt 5.4.2 an der Reihe. Zum Schluss wird der Testlauf mit die Kombination beider Module dargestellt.

5.4.1 HPC-Modul Tests

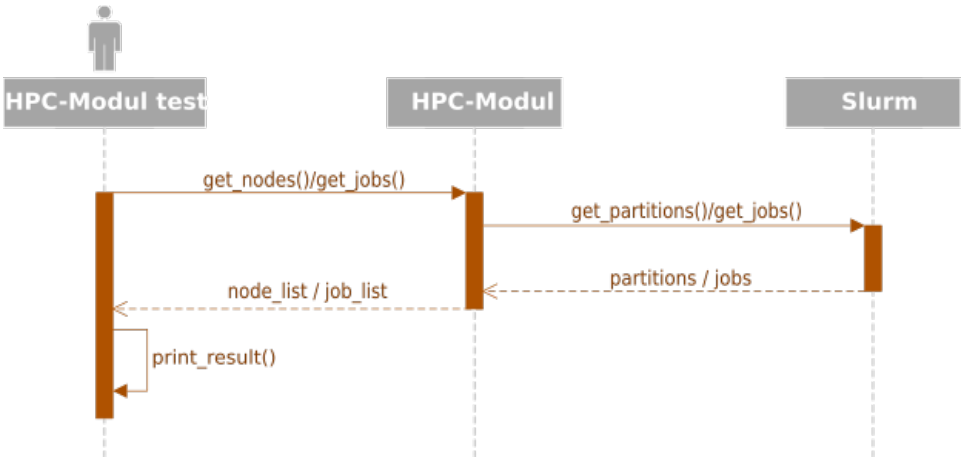

Abbildung 19: Aufbau der Tests für den Erhalt von HPC-Informationen

In der Abbildung 19 ist der Ablauf für die Tests dargestellt, um Informationen von der HPC-Verwaltung zu erhalten. Dabei wird das HPC-Modul entweder nach allen Knoten (Node) gefragt, welches von diesem verwaltet wird oder nach den Informationen der Jobs, welche sich in der Warteschlange befinden. Um diese Antworten zu erhalten, wird die API von Slurm abgefragt. Nachdem Slurm die Informationen geliefert hat, werden diese im HPC-Modul analysiert und die benötigten Informationen an die Testumgebung übergeben. Diese sind entweder die verwendeten Knoten oder die Information zu den Jobs, welche in Abschnitt 5.3.1 erläutert sind. Im Anschluss werden diese Ausgaben analysiert, um zu überprüfen, ob sie übereinstimmen.

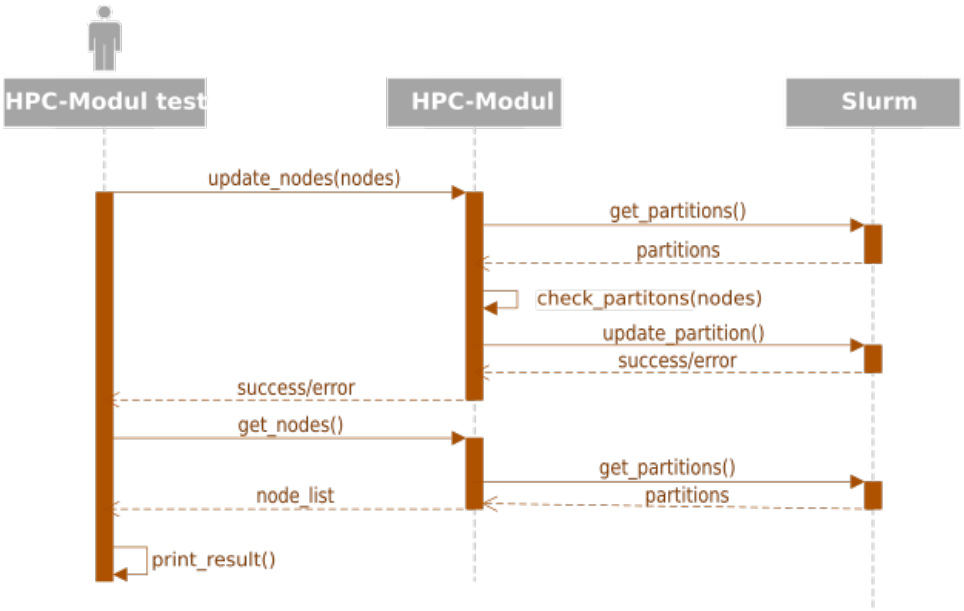

Abbildung 20: Aufbau der Tests zum Entfernen von Knoten aus Slurm

Der Ablauf des Tests, Knoten aus der HPC-Verwaltung zu entfernen, ist in Abbildung 20 veranschaulicht. Dabei wird zuerst eine Anfrage, mit den zu entfernenden Knoten an das HPC-Modul gesendet. Dieser befragt die Slurm API nach den Partitionen. Aus diesen werden die Partitionen ermittelt, welche die Knoten enthalten, die entfernt werden sollen. Die Partitionen, welche die Knoten enthalten, werden danach über die Slurm-API aktualisiert. Eventuelle Fehler werden zurückgegeben und ausgegeben. Danach wird das HPC-Modul noch einmal abgefragt, um nun die Informationen über die Knoten zu erhalten, welche Slurm nun verwaltet. Diese werden danach ausgegeben, um abzugleichen, ob die Änderung umgesetzt wurde.

5.4.2 VM-Modul Tests

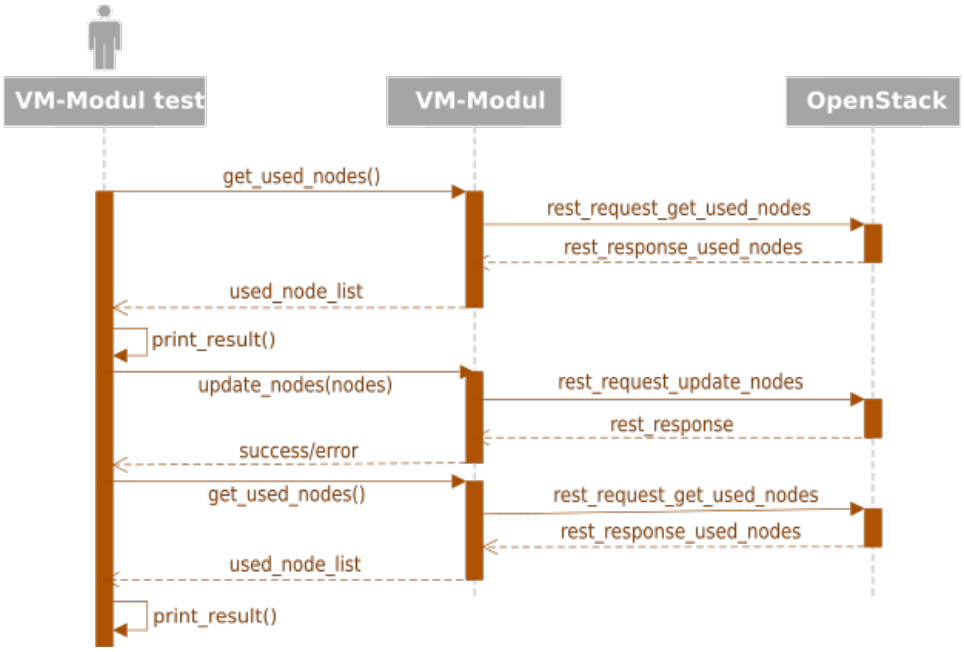

Abbildung 21: Aufbau der Tests zum Erhalten und Manipulieren von VM Hypervisoren

Der Testablauf zum Erhalten und Manipulieren von Hypervisoren, welche von OpenStack verwaltet werden, wird in Abbildung 21 dargestellt. Dabei wird zuerst das VM-Modul abgefragt, welche Knoten von OpenStack verwaltet werden. Dieses sendet einen REST-Anfrage an die REST-API von OpenStack. Die Antwort wird analysiert und die Informationen über die verwendeten Knoten zurückgegeben und im Testprogramm ausgegeben, um abgeglichen zu werden. Danach wird entweder ein Knoten als Hypervisor entfernt oder hinzugefügt. Diese Anfrage wird an das VM-Modul gesendet, welche die REST-Anfrage an OpenStack schickt. Nach der Antwort wird noch einmal OpenStack nach dem verwendeten Hypervisor abgefragt und ausgegeben. Diese Ausgabe wird jeweils über das Webinterface von OpenStack, in Abbildung 22 dargestellt, abgeglichen. Der hinzugefügte oder entfernte Hypervisor ist in der Zone *nova*, entweder vorhanden oder nicht mehr.

Availability Zones

Availability Zone Name	Hosts
internal	• ag_se-edu03 (Services Up) • ag_se-edu02 (Services Up) • ag_se-edu01 (Services Up)
nova	• ag_se-edu03 (Services Up) • ag_se-edu02 (Services Up)

Abbildung 22: Anzeige der Hypervisoren in OpenStack-Webinterface

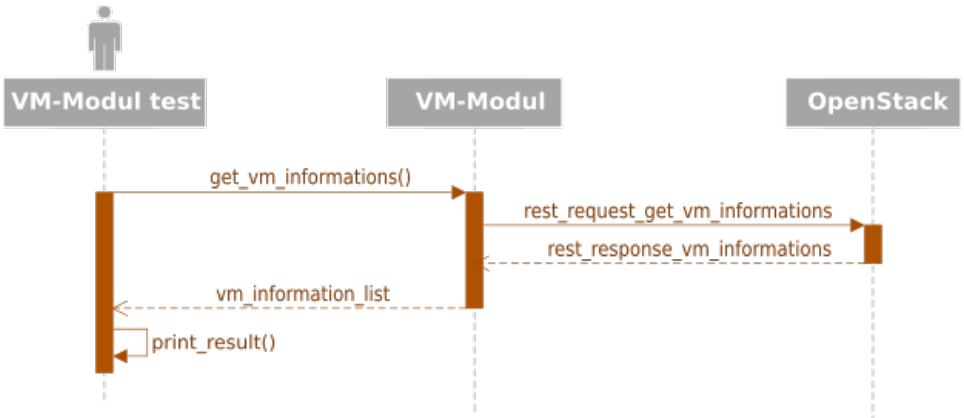

Abbildung 23: Aufbau des Tests zum Erhalten von Information zu den virtuellen Maschinen

Die Abbildung 23 zeigt den Testablauf, um zu überprüfen, ob alle notwendigen Information von OpenStack für die virtuellen Maschinen erhalten werden. Dazu wird das VM-Modul abgefragt, welches wiederum eine REST-Anfrage an OpenStack sendet. Aus deren Antwort werden dann die Information der VMs ermittelt. Diese werden am Ende ausgegeben und danach mit dem OpenStack-Webinterface abgeglichen.

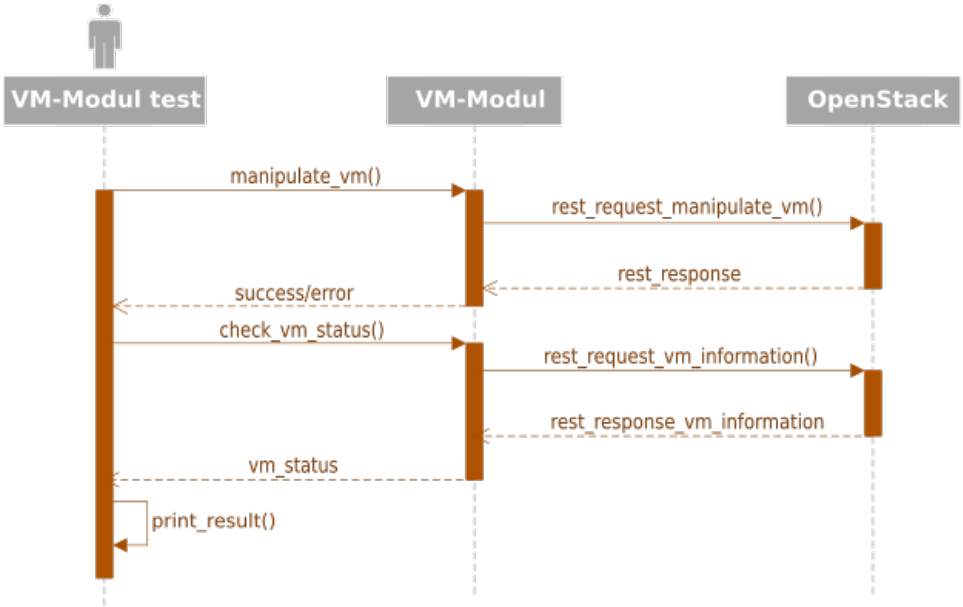

Abbildung 24: Aufbau der Tests zum Manipulieren von virtuellen Maschinen

Der Ablauf der Tests, um die virtuelle Maschinen zu manipulieren, ist in der Abbildung 24 dargestellt. Dabei wird das VM-Modul aufgerufen, um eine VM zu stoppen, zu starten, pausieren zu lassen, aufzuwecken oder auf einen anderen Hypervisor zu migrieren. Das Modul schickt dann die entsprechende REST-Anfrage an OpenStack. Die Antwort wird ausgewertet und an das Testprogramm geschickt. Danach wird der Status der VM abgefragt, um zu überprüfen, ob die Manipulation erfolgreich war. Die Ausgabe wird dann mit dem OpenStack-Webinterface verglichen. Das OpenStack-Webinterface, welche die Informationen der virtuellen Maschinen anzeigt, wird in Abbildung 25 gezeigt. Daran ist zu erkennen, welchen Status die virtuellen Maschinen besitzen und auf welchen Knoten sie laufen. Dies zeigen die Felder *Status* und *Power State*. Sie stehen auf *Aktive* beziehungsweise *Running*, wenn die VM läuft, *Shutoff* und *Shutdown*, sollte die VM ausgeschaltet sein oder *Paused*, wenn sie pausiert wurde.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State
test-volume	-	10.0.110.9	m1.tiny | 512MB RAM | 1 VCPU | 1.0GB Disk	-	Active	nova	None	Running

Abbildung 25: Ansicht des OpenStack-Webinterface für die Informationen der virtuellen Maschinen

5.4.3 HPC-VM-Kombination Tests

Die Kombination beider Module findet nun im Prototyp HIVICMA statt. Dabei wird ein realer Betrieb getestet und die Schritte werden entlang des Ablaufs aufgezeichnet und gegebenenfalls im OpenStack-Webinterface überprüft. Der Ablauf ist in Listing 9 dargestellt.

```
1. hole alle Knoten von Slurm
2. hole alle Knoten von OpenStack
3. entferne ag_se-edu03 von Slurm
4. hole alle Information zu verhanden Jobs aus Slurm.
5a. - Wenn Jobs vorhanden sind erstelle Strukturen und
      errechne freie Zeiten für ag_se-edu02
    - erstelle Timer für FREE VM NODES, FREE HPC NODES
      und CHECK HPC TIMER mithilfe der Informationen.
 b. wenn keine Jobs vorhanden sind, stelle Timer für
    CHECK HPC
6. nehme den ersten Timer aus der liste
7. lege dich entsprechend des ersten Timer schlafen
8. aufwecken nach entsprechender Zeit des Timers
9a. wenn FREE VM NODES
  - entferne ag_se-edu02 aus OpenStack
  - teste ob VMs auf ag_se-edu02 laufen
    - wenn ja verschiebe sie nach ag_se-edu03
 b. wenn FREE HPC NODES
  - füge Knoten ag_se-edu02 zu OpenStack hinzu
 c. wenn CHECK HPC TIMER
  - hole Job Information au Slurm
  - teste ob sich etwas geändert hat
  - dann mit 5. weiter
10. wiederhole ab 6.
```

Listing 9: Ablauf des Test mit HIVICMA

5.5 Auswertung der Testläufe

Dieser Abschnitt beinhaltet eine kurze Auswertung der Testläufe, ob dieser erfolgreich waren oder worauf geachtet werden musste. Zuerst werden dabei die Testläufe für das HPC-Modul, in Abschnitt 5.5.1, und für das VM-Modul, in Abschnitt 5.5.2, ausgewertet. Danach die Testläufe für die Kombination beider Module in HIVICMA, in Abschnitt 5.5.3 betrachtet.

5.5.1 HPC-Modul Testläufe

Die beiden verschiedenen Testabläufe für das HPC-Modul sind einmal die Ermittlung aller notwendiger Jobinformationen sowie das Ermitteln der verwendeten Knoten von Slurm und deren Manipulation. Dabei werden jedoch, wie in den Anforderungen der Aufgabe, in Abschnitt 3.4 beschrieben, nur Knoten entfernt, um einen Bereich für die virtuellen Maschinen zu erstellen, damit diese für VMs bereit steht, falls alle anderen Knoten von HPC-Anwendungen benutzt werden.

```
goerickm@ag_se-edu01:~$ sbatch --begin=2016-12-24T12:00:00 --nodelist=ag_se-edu02 test-slurm-job.sh
Submitted batch job 122
goerickm@ag_se-edu01:~$ ./hpc-test.run 1
--------------------------
job id : 122
partition: debug
used nodes: ag_se-edu02
start time: Sat Dec 24 12:00:00 2016
end time: Sat Dec 24 12:05:00 2016
goerickm@ag_se-edu01:~$
```

Abbildung 26: Beispiel Ausgabe eines Jobs, welcher mit dem Test ermittelt wurde, sowie des vorherigen Starts des Jobs.

Ein Beispiel für die Ermittlung aller Informationen zu einen HPC-Job ist in Abbildung 26 dargestellt. Wie zu erkennen, sind die Zeiten für den Start und das Ende des Jobs, sowie die verwendeten Knoten ermittelt worden. Der dazu erstellte Test-Job ist in Abschnitt 5.2 unter Slurm beschrieben. Somit ist gezeigt, dass alle notwendigen Informationen ermittelt werden können.

```
goerickm@ag_se-edu01:~$ ./hpc-test.run 2
node name: ag_se-edu02
node name: ag_se-edu03
goerickm@ag_se-edu01:~$ sudo ./hpc-test.run 3
partition changed
goerickm@ag_se-edu01:~$ ./hpc-test.run 2
node name: ag_se-edu02
goerickm@ag_se-edu01:~$
```

Abbildung 27: Ausgabe der Manipulation der Knoten, welche Slurm verwaltet.

Die Abbildung 27 zeigt die Ausgabe des Tests zum Ermitteln und Manipulieren der Knoten von Slurm, welcher in Abschnitt 5.3.2 beschrieben ist. Dabei werden zuerst alle Knoten von Slurm ermittelt. Danach wird der Knoten *ag_se-edu03* aus der *default*-Partition entfernt. Die Ausgabe *partition changed* bestätigt, dass keine Fehler beim Ändern der Partition entstanden sind. Ansonsten würde die Ausgabe mit dem entstandenen Fehler von Slurm angezeigt werden. Zum Schluss werden nochmals alle Knoten von Slurm ermittelt, um zu zeigen, das der entsprechende Knoten entfernt wurde. Der Einfachheit halber sind es drei Aufrufe der Tests. Wie zu erkennen, werden die Knoten ermittelt und mögliche Knoten von Slurm entfernt, um somit einen Bereich nur für VMs zu erstellen. Für das Manipulieren der Knoten werden Administrationsrechte benötigt und somit müssen diese Tests sowie die Tests mit HIVICMA mittels *sudo* gestartet werden. Auf diese Einschränkung sollte beim ausführen geachtet werden, da hiermit Probleme verhindert werden.
Die Tests für da HPC-Modul können als erfolgreich gewertet werden, da zum einen die Informationen über die Jobs ermittelt werden können und zum anderen die Knoten von Slurm ausgelesen sowie manipuliert werden können. Damit ist es möglich, das HPC-Modul für HIVICMA zu verwenden.

5.5.2 VM-Modul Testläufe

Bei den Testläufen für das VM-Modul ging es darum, die Anforderungen für Aufgabe, in Abschnitt 3.4 beschrieben, zu ermitteln. Da wäre sowohl die Beschaffung aller von OpenStack verwendeten Knoten und das Entfernen und Hinzufügen von Knoten als auch das Ermitteln der benötigten Information aller VMs und das Manipulieren dieser VMs und im wesentlichen das Migrieren auf einen anderen Knoten.

```
goerickm@ag_se-edu01:~$ ./vm-test.run [redacted] [redacted] 8

Test to get all hosts
------------------
hosts:
host name: ag_se-edu02
host name: ag_se-edu03
goerickm@ag_se-edu01:~$
```

Abbildung 28: Ausgabe des Tests zum Ermitteln der Knoten von OpenStack.

Die Abbildung 28 zeigt die Ausgabe der Knoten, welche von OpenStack

verwaltet wird. Die gezeigten Knoten werden dann mit der in Abbildung 22 gezeigten Knoten der Zone *nova* verglichen. Wie zu erkennen ist, werden alle Knoten, welche von OpenStack verwendet werden, auch mithilfe des VM-Moduls ermittelt.

```
goerickm@ag_se-edu01:~$ ./vm-test.run [redacted] [redacted] 7

Test to get all vms
---------------------------
vm name: cddd90be-2a4f-4dd2-b3a5-0d56cd70d81a
vm host: ag_se-edu03
vm state: active
goerickm@ag_se-edu01:~$
```

Abbildung 29: Ausgabe des Tests zum Ermitteln der Informationen aller VMs.

In der Abbildung 29 wird die Ausgabe des Tests zur Beschaffung der Information für die VMs dargestellt. Diese werden mit der in Abbildung 25 gezeigten Darstellung im OpenStack-Webinterface abgeglichen, um zu erkennen, ob die Informationen übereinstimmen. In den Tests wurden immer die korrekten Informationen der VMs ermittelt.

```
goerickm@ag_se-edu01:~$ ./vm-test.run [redacted] [redacted] 5
Test to migrate vm.

Test to get all vms
---------------------------
vm name: cddd90be-2a4f-4dd2-b3a5-0d56cd70d81a
vm host: ag_se-edu03
vm state: active

Migrate vm cddd90be-2a4f-4dd2-b3a5-0d56cd70d81a from node ag_se-edu03 to node ag_se-edu02
10 second sleep.

Test to get all vms
---------------------------
vm name: cddd90be-2a4f-4dd2-b3a5-0d56cd70d81a
vm host: ag_se-edu02
vm state: active
goerickm@ag_se-edu01:~$
```

Abbildung 30: Ausgabe des Live-Migration Tests.

Die Ausgabe des Tests der Migration von virtuellen Maschinen wird in der Darstellung 30 gezeigt. Dabei wird zuerst die Informationen der VM angezeigt. Danach wird die VM von *ag_se-edu02* auf *ag_se-edu03* migriert. Nach kurzer Zeit wird nochmal die Information der VM abgefragt. Dies kann, je nach Größe der VM ein paar Minuten dauern. Es ist zu erkennen, dass die VM nach der Migration den Hypervisor (host) gewechselt hat.

```
goerickm@ag_se-edu01:~$ ./vm-test.run [redacted] [redacted] 9
remove ag_se-edu02 for vm usage
ag_se-edu02 removed.
goerickm@ag_se-edu01:~$ ./vm-test.run [redacted] [redacted] 10
add node for vm usage
ag_se-edu02 added
goerickm@ag_se-edu01:~$
```

Abbildung 31: Ausgabe des Tests zum Entfernen oder Hinzufügen der Knoten für OpenStack.

Die Darstellung 31 zeigt die Ausgabe der Tests, um einen Knoten zu entfernen oder hinzuzufügen. OpenStack kann keine VMs auf den entfernten Knoten starten beziehungsweise wenn dieser Knoten wieder hinzugefügt wurde, kann es dort wieder VMs starten. Dies kann im Webinterface, in Abbildung 22 dargestellt, in der Zone *nova* geprüft werden. Dabei ist der Knoten *ag_se-edu02* entweder nicht vorhanden, wenn er entfernt wird, beziehungsweise vorhanden, wenn er hinzugefügt wird.
Wie in diesen Tests zum VM-Modul gezeigt wurde, werden die vorher beschrieben Anforderungen erfüllt. Damit kann dieses nun für HIVICMA verwendet werden.

5.5.3 HPC-VM-Kombination Testläufe

Die Testläufe von HIVICMA zeigen, dass das VM-Modul sowie das HPC-Modul angesprochen werden kann. Dazu werden die, in Abschnitt 4.1.1 beschriebenen, Timer anhand von Testjobs für Slurm erstellt und HIVICMA anhand dieser schlafen gelegt und aufgeweckt. Des weiteren ist zu sehen, dass die Knoten beim *FREE_VM_NODES*-Timer für den VM Betrieb entfernt werden und beim *FREE_HPC_NODES*-Timer hinzugefügt werden. Außerdem ist zu sehen, dass die VMs verschoben werden, bevor auf *ag_se-edu02* eine HPC-Anwendung gestartet wird. Damit ist gezeigt, dass mit den in Abschnitt 5.2 beschriebenen Einstellungen und Voraussetzungen die Kombination von einer HPC-Verwaltung und einer VM-Verwaltung auf dem gegebenen Test-Cluster möglich ist.

6 Fazit und Aussicht

In diesem letzten Abschnitt geht es darum, die Tests des von mir entwickelten Programms HIVICMA in Hinblick auf die Kombination von HPC- und VM-Cluster zu bewerten und deren Machbarkeit zu diskutieren. Dies geschieht in Absatz 6.1. Danach wird in Teil 6.2 noch eine Aussicht über weitergehende Entwicklungen in dieser Richtung gegeben, sowie eventuelle Anpassungen von HIVICMA.

6.1 Fazit

Mit den Tests aus Kapitel 5 konnte gezeigt werden, dass die Möglichkeit besteht, dass Cluster für HPC-Anwendungen und Cluster für VMs zusammengeführt werden können. Die Voraussetzung, dass HPC-Anwendungen priorisiert werden, wurde umgesetzt. Außerdem wurde, die in der Aufgabenstellung aufgestellten Anforderungen vollständig erfüllt. Es ist möglich, dass die Informationen von der HPC-Verwaltung sowie der VM-Verwaltung ermittelt werden. Des weiteren können die VMs verschoben werden. Außerdem können die Knoten dieser Verwaltungen manipuliert werden. Damit besteht die Möglichkeit, einen reservierten Bereich für VMs zu erstellen und Hypervisoren hinzuzufügen sowie zu entfernen.
Jedoch sind auch einige Einschränkungen bei der Kombination zu beachten. Diese Einschränkungen gehen vor allem aus den Voraussetzungen für die Tests, in Abschnitt 5.2, hervor. Dazu zählt die Erstellung von virtuellen Maschinen. Die erstellten VMs müssen die Möglichkeit besitzen, mittels Live-Migration verschoben zu werden. Ansonsten muss die Migration entsprechend implementiert werden und damit die Zeiten der Timer angepasst werden, weil diese Migration länger dauert.
Des weiteren ist das Testsystem sehr klein, wodurch für die HPC-Verwaltung nur ein Knoten zur Verfügung stand und die Zeiten für die Verwaltung des Clusters relativ kurz waren, welche bei größeren Clustern nicht vernachlässigt werden dürfen. Außerdem wurde die Implementierung wurde nur für Slurm als HPC-Verwaltung und OpenStack als VM-Verwaltung umgesetzt. Andere Umsetzungen für diese Verwaltungen bieten vielleicht keine oder solch umfangreiche API, um die notwendigen Information zu erhalten oder die Manipulationen der VMs beziehungsweise Knoten durchzuführen.
Es ist außerdem nicht darauf eingegangen worden, wie aus der Beschreibung

der unterschiedlichen Cluster-Architekturen aus Abschnitt 2.4.6, hervorgeht, dass das kombinierte Cluster ein spezielles Kommunikationsnetzwerk für die HPC-Anwendungen besitzen müsste. Für den Aufbau eines Clusters, welche beide Funktionalitäten bereitstellt, bräuchte der Bereich, welcher nur von VMs genutzt wird, keinen Anschluss an solch ein Kommunikationsnetzwerk, womit Kosten eingespart werden können.
Die Nutzer einer Software, wie HIVICMA, würden nur beim Start dieser Einstellungen wie etwa die Knoten, welche nur für die Nutzung von VMs festgelegt wurden, oder die Zeiten der Timer anpassen. Ansonsten würde die Software im Hintergrund autonom laufen und die Verwaltung der virtuellen Maschinen weiterhin über OpenStack oder ähnliche Software erlauben, wenn dafür Module entwickelt wurden beziehungsweise für das Anlegen von HPC-Anwendungen weiterhin Slurm oder ähnliche Verwaltungen genutzt werden, sollten dafür Module entwickelt worden sein.

6.2 Ausblick

In Bezug auf weitere Arbeiten in dem Bereich der Kombination beider Cluster-Nutzungsarten wäre das Durchführen von Tests mit einem größeren Cluster, als dem zur Verfügung gestellten. Außerdem sollten diese Tests mit mehreren virtuellen Maschinen sowie realen HPC-Anwendungen, durchgeführt werden. Des weiteren wird im Prototyp, bedingt durch die Größe des Test-Clusters, nur mit einfachen Listen gearbeitet. Davon müssten einige möglicherweise durch effektivere Datenstrukturen, wie Bäume ersetzt werden. Dazu müssten je nach Größe der Cluster Tests durchgeführt werden, um zu ermitteln, welche Datenstruktur effizient ist.
Des weiteren könnten die Zeiten der Timer mittels Experimente angepasst werden. Je nach Situation ist eine dynamische Anpassung der Timer möglich. Unter den Experimenten wären Tests nötig, wie häufig und zu welchen Zeiten neue HPC-Anwendungen hinzugefügt werden und eventuell, welche Knoten oft benutzt werden. Daraus ergebe sich, wann die HPC-Verwaltung nach Information abgefragt werden würde und welche Knoten eher für VMs genutzt werden können.
Des weiteren wären Experimente zur Dauer der Migrationen sinnvoll, um abzuschätzen, wann ein Knoten für eine HPC-Anwendung freigeräumt werden muss. Dies könnte durchaus auch im Programm zur Kombination beider Systeme durchgeführt werden und damit die Zeiten dynamisch angepasst

werden. Das ist jedoch abhängig von der VM-Verwaltung, wenn diese die Daten der Dauer von bisherigen Migrationen zur Verfügung stellen kann. Damit könnten die Knoten effektiver ausgenutzt werden, bevor die VMs in den reservierten Bereich verschoben werden müssten.
Um Einstellungen, wie der Zeiten für Timer und Nutzung der Knoten im Cluster, anzupassen sowie die Benutzbarkeit zu vereinfachen wäre es sinnvoll ein Benutzeroberfläche zu entwickeln.

7 Anhang

```
typedef struct job_info {
  char *account;                /* charge to specified account */
  char *admin_comment;          /* administrator's arbitrary comment */
  char    *alloc_node;          /* local node making resource alloc */
  uint32_t alloc_sid;           /* local sid making resource alloc */
  void *array_bitmap;           /* NOTE: set on unpack */
  uint32_t array_job_id;        /* job_id of a job array or 0 if N/A */
  uint32_t array_task_id;       /* task_id of a job array */
  uint32_t array_max_tasks;     /* Maximum number of running tasks */
  char *array_task_str;         /* string expression of task IDs in this
                                 * record */
  uint32_t assoc_id;            /* association id for job */
  uint16_t batch_flag;          /* 1 if batch: queued job with script */
  char *batch_host;             /* name of host running batch script */
  char *batch_script;           /* contents of batch script */
  uint32_t bitflags;            /* Various job flags */
  uint16_t boards_per_node;     /* boards per node required by job   */
  char *burst_buffer;           /* burst buffer specifications */
  char *burst_buffer_state;     /* burst buffer state info */
  char *command;                /* command to be executed, built from
                                 * submitted job's argv and NULL for salloc
                                 * command */
  char *comment;                /* arbitrary comment (used by Moab
                                 * scheduler)*/
  uint16_t contiguous;          /* 1 if job requires contiguous nodes */
  uint16_t core_spec;           /* specialized core count */
  uint16_t cores_per_socket;    /* cores per socket required by job  */
  double billable_tres;         /* billable TRES cache. updated upon resize */
  uint16_t cpus_per_task;       /* number of processors required for
                                 * each task */
  uint32_t cpu_freq_min;        /* Minimum cpu frequency  */
  uint32_t cpu_freq_max;        /* Maximum cpu frequency  */
  uint32_t cpu_freq_gov;        /* cpu frequency governor */
  time_t deadline;              /* deadline */
  uint32_t delay_boot;          /* delay boot for desired node state */
  char *dependency;             /* synchronize job execution with other jobs*/
  uint32_t derived_ec;          /* highest exit code of all job steps */
  time_t eligible_time;         /* time job is eligible for running */
  time_t end_time;              /* time of termination, actual or expected */
  char *exc_nodes;              /* comma separated list of excluded nodes */
  int32_t *exc_node_inx;        /* excluded list index pairs into node_table:
                                 * start_range_1, end_range_1,
                                 * start_range_2, .., -1  */
  uint32_t exit_code;           /* exit code for job (status from wait call)*/
  char *features;               /* comma separated list of required features*/
  char *fed_origin_str;         /* Origin cluster's name */
  uint64_t fed_siblings;        /* bitmap of sibling cluster ids */
  char *fed_siblings_str;       /* string of sibling cluster names */
  char *gres;                   /* comma separated list of generic resources*/
  uint32_t group_id;            /* group job submitted as */
  uint32_t job_id;              /* job ID */
  job_resources_t *job_resrcs;  /* opaque data type, job resources */
  uint32_t job_state;           /* state of the job, see enum job_states */
  char *licenses;               /* licenses required by the job */
  uint32_t max_cpus;            /* maximum number of cpus usable by job */
  uint32_t max_nodes;           /* maximum number of nodes usable by job */
  char *mcs_label;              /* mcs_label if mcs plugin in use */
  char *name;                   /* name of the job */
  char *network;                /* network specification */
  char *nodes;                  /* list of nodes allocated to job */
  uint32_t nice;                /* requested priority change */
  int32_t *node_inx;            /* list index pairs into node_table for
                                 * *nodes:
                                 * start_range_1, end_range_1,
                                 * start_range_2, .., -1  */
  uint16_t ntasks_per_core;     /* number of tasks to invoke on each core */
```

```
  uint16_t ntasks_per_node;     /* number of tasks to invoke on each node */
  uint16_t ntasks_per_socket;   /* number of tasks to invoke on each socket*/
  uint16_t ntasks_per_board;    /* number of tasks to invoke on each board */
  uint32_t num_cpus;            /* minimum number of cpus required by job */
  uint32_t num_nodes;           /* minimum number of nodes required by job */
  uint32_t num_tasks;           /* requested task count */
  char *partition;              /* name of assigned partition */
  uint64_t pn_min_memory;       /* minimum real memory per node, default=0 */
  uint16_t pn_min_cpus;         /* minimum # CPUs per node, default=0 */
  uint32_t pn_min_tmp_disk;     /* minimum tmp disk per node, default=0 */
  uint8_t power_flags;          /* power management flags,
                                 * see SLURM_POWER_FLAGS_ */
  time_t preempt_time;          /* preemption signal time */
  time_t pre_sus_time;          /* time job ran prior to last suspend */
  uint32_t priority;            /* relative priority of the job,
                                 * 0=held, 1=required nodes DOWN/DRAINED */
  uint32_t profile;             /* Level of acct_gather_profile {all | none}*/
  char *qos;                    /* Quality of Service */
  uint8_t reboot;               /* node reboot requested before start */
  char *req_nodes;              /* comma separated list of required nodes */
  int32_t *req_node_inx;        /* required list index pairs into node_table:
                                 * start_range_1, end_range_1,
                                 * start_range_2, .., -1  */
  uint32_t req_switch;          /* Minimum number of switches */
  uint16_t requeue;             /* enable or disable job requeue option */
  time_t resize_time;           /* time of latest size change */
  uint16_t restart_cnt;         /* count of job restarts */
  char *resv_name;              /* reservation name */
  char *sched_nodes;            /* list of nodes scheduled to be used for
                                 * job */
  dynamic_plugin_data_t *select_jobinfo; /* opaque data type,
                                          * process using
                                          * slurm_get_select_jobinfo() */
  uint16_t shared;              /* 1 if job can share nodes with other jobs */
  uint16_t show_flags;          /* conveys level of details requested */
  uint16_t sockets_per_board;   /* sockets per board required by job */
  uint16_t sockets_per_node;    /* sockets per node required by job  */
  time_t start_time;            /* time execution begins, actual or expected*/
  uint16_t start_protocol_ver;  /* Slurm version step was started with
                                 * either srun or the lowest slurmd version
                                 * it is talking to */
  char *state_desc;             /* optional details for state_reason */
  uint16_t state_reason;        /* reason job still pending or failed, see
                                 * slurm.h:enum job_state_reason */
  char *std_err;                /* pathname of job's stderr file */
  char *std_in;                 /* pathname of job's stdin file */
  char *std_out;                /* pathname of job's stdout file */
  time_t submit_time;           /* time of job submission */
  time_t suspend_time;          /* time job last suspended or resumed */
  uint32_t time_limit;          /* maximum run time in minutes or INFINITE */
  uint32_t time_min;            /* minimum run time in minutes or INFINITE */
  uint16_t threads_per_core;    /* threads per core required by job  */
  char *tres_req_str;           /* tres reqeusted in the job */
  char *tres_alloc_str;         /* tres used in the job */
  uint32_t user_id;             /* user the job runs as */
  uint32_t wait4switch;         /* Maximum time to wait for minimum switches*/
  char *wckey;                  /* wckey for job */
  char *work_dir;               /* pathname of working directory */
} slurm_job_info_t;
```

Listing 10: Slurm Struktur für Informationen eines einzelnen Jobs [19]

```
typedef struct job_info_msg {
  time_t last_update;           /* time of latest info */
  uint32_t record_count;        /* number of records */
  slurm_job_info_t *job_array;  /* the job records */
} job_info_msg_t;
```

Listing 11: Slurm Struktur für alle Job Informationen [19]

```
typedef struct partition_info_msg {
  time_t last_update;                /* time of latest info */
  uint32_t record_count;             /* number of records */
  partition_info_t *partition_array; /* the partition records */
} partition_info_msg_t;
```

Listing 12: Slurm Struktur für aller Informationen der Partitionen [19]

```
typedef struct partition_info {
  char *allow_alloc_nodes;      /* list names of allowed allocating
                                 * nodes */
  char *allow_accounts;         /* comma delimited list of accounts,
                                 * null indicates all */
  char *allow_groups;           /* comma delimited list of groups,
                                 * null indicates all */
  char *allow_qos;              /* comma delimited list of qos,
                                 * null indicates all */
  char *alternate;              /* name of alternate partition */
  char *billing_weights_str;    /* per TRES billing weights string */
  uint16_t cr_type;             /* see CR_* values */
  uint64_t def_mem_per_cpu;     /* default MB memory per allocated CPU */
  uint32_t default_time;        /* minutes, NO_VAL or INFINITE */
  char *deny_accounts;          /* comma delimited list of denied accounts */
  char *deny_qos;               /* comma delimited list of denied qos */
  uint16_t flags;               /* see PART_FLAG_* above */
  uint32_t grace_time;          /* preemption grace time in seconds */
  uint32_t max_cpus_per_node;   /* maximum allocated CPUs per node */
  uint64_t max_mem_per_cpu;     /* maximum MB memory per allocated CPU */
  uint32_t max_nodes;           /* per job or INFINITE */
  uint16_t max_share;           /* number of jobs to gang schedule */
  uint32_t max_time;            /* minutes or INFINITE */
  uint32_t min_nodes;           /* per job */
  char *name;                   /* name of the partition */
  int32_t *node_inx;            /* list index pairs into node_table:
                                 * start_range_1, end_range_1,
                                 * start_range_2, .., -1  */
  char *nodes;                  /* list names of nodes in partition */
  uint16_t over_time_limit;     /* job's time limit can be exceeded by this
                                 * number of minutes before cancellation */
  uint16_t preempt_mode;        /* See PREEMPT_MODE_* in slurm/slurm.h */
  uint16_t priority_job_factor; /* job priority weight factor */
  uint16_t priority_tier;       /* tier for scheduling and preemption */
  char *qos_char;               /* The partition QOS name */
  uint16_t state_up;            /* see PARTITION_ states above */
  uint32_t total_cpus;          /* total number of cpus in the partition */
  uint32_t total_nodes;         /* total number of nodes in the partition */
  char    *tres_fmt_str;        /* str of configured TRES in partition */
} partition_info_t;
```

Listing 13: Slurm Struktur für Informationen einer einzelnen Partition [19]

Abbildungsverzeichnis

1 Übersicht über Type I und Type II Hypervisoren [4] 4
2 Aufbau von OpenStack [15] 5
3 Slurm Architektur [18] 9
4 High Availability Cluster Beispiel 12
5 Load-Balance Cluster Beispiel Aufbau 13
6 Aufbau des Clusters für virtuelle Maschinen 14
7 HPC Cluster Beispiel 15
8 Die ersten drei der Top 500 Supercomputer von November 2016 [6] 16
9 Unterschiede zwischen Object-Storage und File-Storage [32] . 17
10 Vergleich CPU und GPU [36] 20
11 Beispiel eines Aufbaus der Events zu Knotenstruktur 24
12 Aufbau der freien Zeiten zur Knotenstruktur 25
13 Beispiel der laufenden virtuellen Maschinen. 28
14 Aufbau des Lösungsansatz 29
15 Klassische Migration von virtuellen Maschinen 31
16 Live-Migration von virtuellen Maschinen 31
17 Aufbau des Clusters für den Test 35
18 Einstellung beim Erstellen einer VM unter OpenStack. 38
19 Aufbau der Tests für den Erhalt von HPC-Informationen . . 43
20 Aufbau der Tests zum Entfernen von Knoten aus Slurm . . . 44
21 Aufbau der Tests zum Erhalten und Manipulieren von VM Hypervisoren 45
22 Anzeige der Hypervisoren in OpenStack-Webinterface 46
23 Aufbau des Tests zum Erhalten von Information zu den virtuellen Maschinen 46
24 Aufbau der Tests zum Manipulieren von virtuellen Maschinen 47
25 Ansicht des OpenStack-Webinterface für die Informationen der virtuellen Maschinen 48
26 Beispiel Ausgabe eines Jobs, welcher mit dem Test ermittelt wurde, sowie des vorherigen Starts des Jobs. 49
27 Ausgabe der Manipulation der Knoten, welche Slurm verwaltet. 49
28 Ausgabe des Tests zum Ermitteln der Knoten von OpenStack. 50
29 Ausgabe des Tests zum Ermitteln der Informationen aller VMs. 51

30 Ausgabe des Live-Migration Tests. 51
31 Ausgabe des Tests zum Entfernen oder Hinzufügen der Knoten für OpenStack. 52

Listings

1 Aufbau der Timer 26
2 Ablauf des Start vom Programm 32
3 Ablauf vor dem Start eines Jobs 33
4 Ablauf nach Beenden eines Jobs und Knoten frei wird 33
5 Ablauf des allgemeinen Timers 34
6 Verwendeter Job für Tests mit Slurm. 37
7 Befehl zum Starten eines Slurm-Jobs. 37
8 Aufbau des Testablaufes für die Ermittlung und Manipulation der HPC-Knoten 40
9 Ablauf des Test mit HIVICMA 48
10 Slurm Struktur für Informationen eines einzelnen Jobs [19] . 56
11 Slurm Struktur für alle Job Informationen [19] 58
12 Slurm Struktur für aller Informationen der Partitionen [19] . 58
13 Slurm Struktur für Informationen einer einzelnen Partition [19] 58

Literatur

[1] Goddard Space Flight Center. https://www.nasa.gov/goddard.

[2] Ethernet in High Performance Computing Clusters. Technical report, Force10 Networks, Inc., 2006.

[3] Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing. Technical report, HPC Advisory Council, 2009.

[4] Much ado about type-2. https://microkerneldude.wordpress.com/2010/10/14/much-ado-about-type-2/, October 2010 (besucht Dezember 2016).

[5] Virtualizing hpc and technical computing with vmware vsphere. Technical report, Physics Labortory, John Hopkins University, 2016.

[6] Top 500. https://www.top500.org/, (besucht Dezember 2015).

[7] Beowulf Project. http://www.beowulf.org, (besucht Dezember 2016).

[8] Citrix systems, inc. https://www.citrix.com/, (besucht Dezember 2016).

[9] DDN. http://www.ddn.com/, (besucht Dezember 2016).

[10] Detusches Klimarechenzentrum. https://www.dkrz.de/, (besucht Dezember 2016).

[11] Fachbereich Mathematik und Informatik der Freie Universität Berlin, allegro. http://www.allegro.imp.fu-berlin.de/, (besucht Dezember 2016).

[12] Fibre Channel. http://fibrechannel.org/, (besucht Dezember 2016).

[13] libcurl. https://curl.haxx.se, (besucht Dezember 2016).

[14] libslurm-dev. https://packages.debian.org/jessie/libslurm-dev, (besucht Dezember 2016).

[15] Openstack. http://www.openstack.org/, (besucht Dezember 2016).

[16] Openstack ironic. https://wiki.openstack.org/wiki/Ironic, (besucht Dezember 2016).

[17] Projekte am Fachbereich Mathematik und Informatik der Freien Universität Berlin. http://www.mi.fu-berlin.de/en/inf/projects/index.html, (besucht Dezember 2016).

[18] Slurm. http://slurm.schedmd.com/, (besucht Dezember 2016).

[19] Slurm github. https://github.com/SchedMD/slurm, (besucht Dezember 2016).

[20] Virtuelle Server Service des Fachbereich Mathematik und Informatik der Freie Universität Berlin. http://www.mi.fu-berlin.de/w/IT/ItServicesVirtualServer, (besucht Dezember 2016).

[21] Vmware. http://www.vmware.com/, (besucht Dezember 2016).

[22] Xen. https://www.xenproject.org/, (besucht Dezember 2016).

[23] Zedat High-Performance Computing. https://www.zedat.fu-berlin.de/HPC/Home, (besucht Dezember 2016).

[24] Blaise Barney. Message Passing Interface (mpi). Technical report, Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/mpi/, 2016.

[25] Michael Clark, Jehiah Czebotar, Eric Haszlakiewicz, and C. Watford. libjson-c. https://github.com/json-c/json-c, (besucht Dezember 2016).

[26] IEEE 802 LAN/MAN Standards Committee. IEEE 802.3 ETHERNET WORKING GROUP. http://www.ieee802.org/3/, November 2016 (besucht Dezember 2016).

[27] Michael Feldman. Ethernet Will Get Spotlight at SC16. https://www.top500.org/news/ethernet-will-get-spotlight-at-sc16/, November 2016 (besucht Dezember 2016).

[28] Pravin Ganore. Cluster Computing: Definition and Architecture of a Cluster. https://www.esds.co.in/blog/cluster-computing-definition-and-architecture-of-a-cluster/, März 2014 (besucht Dezember 2016).

[29] Robert P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, Cambridge, MA, 1972.

[30] Ken Hess. What's next in aaS? Workspace-as-a-Service. http://www.zdnet.com/article/whats-next-in-aas-workspace-as-a-service/, Oktober 2015 (besucht Dezember 2016).

[31] Achim Karpf. Vor und Nachteile von Virtualisierungsarchitekturen. http://www.datacenter-insider.de/vor-und-nachteile-von-virtualisierungsarchitekturen-a-137958/, Juli 2008 (besucht Dezember 2016).

[32] Michael Lee. Object Storage vs. File Storage: What's the Difference? https://cloudian.com/blog/object-storage-vs-file-storage/, September 2016 (besucht Dezember 2016).

[33] Isaac Lopez. IDC Talks Convergence in High Performance Data Analysis. https://www.datanami.com/2013/06/19/idc_talks_convergence_in_high_performance_data_analysis/, June 2013 (besucht Dezember 2016).

[34] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Confidentiality issues on a GPU in a virtualized environment. In *FC 2014, 18th International Conference on Financial Cryptography and Data Security, 3-7 March 2014, Barbados*, Barbados, BARBADOS, 03 2014.

[35] Microsoft. Hyper-v. https://www.microsoft.com/de-de/server-cloud/solutions/virtualization.aspx, (besucht Dezember 2016).

[36] Nvidia. Was ist gpu computing? http://www.nvidia.de/object/gpu-computing-de.html, (besucht Dezember 2016).

[37] Oracle. Virtualbox. https://www.virtualbox.org/, (besucht Dezember 2016).

[38] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[39] M.E. Porter. *Competitive Advantage of Nations.* Free Press, 1990.

[40] P. Rad, A. T. Chronopoulos, P. Lama, P. Madduri, and C. Loader. Benchmarking Bare Metal Cloud Servers for HPC Applications. In

2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pages 153–159, Nov 2015.

[41] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. Internet Small Computer Systems Interface (iSCSI). RFC 3720 (Proposed Standard), April 2004. Obsoleted by RFC 7143, updated by RFCs 3980, 4850, 5048, 7146.

[42] R Shreya, R Shruthi, MR Anala, and G Shobha. Performance analysis during live migration. In *International Journal of Engineering Research and Technology*, volume 2. ESRSA Publications, 2013.

[43] T.J. Socolofsky and C.J. Kale. TCP/IP tutorial. RFC 1180 (Informational), January 1991.

[44] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14. CRC Press, 1995.