Service zum Start von Grid-Jobs in virtuellen Machinen

Steffen Limmer, Dietmar Fey
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Informatik 3, Erlangen, Deutschland
{steffen.limmer,dietmar.fey}@informatik.uni-erlangen.de

Kurzfassung

In diesem Papier stellen wir einen Service für den transparenten Start von Grid-Jobs in virtuellen Maschinen vor. Damit wird die Bereitstellung von Laufzeitumgebungen und Applikationen für Grid-Jobs auf fremden Ressourcen ermöglicht. Der dafür nötige Zusatzaufwand für den Nutzer soll so gering wie möglich gehalten werden. Das Paper stellt die Funktionsweise und Benutzung vor und geht auf Aspekte der Performanz ein.

1 Einleitung

Ein wesentliches Merkmal von Grid-Systemen ist ihre Heterogenität. Diese bezieht sich nicht nur auf Hardwaremerkmale der Ressourcen eines Grids, sondern auch auf die dort installierte bzw. verfügbare Software und Laufzeitumgebung. Wenn eine Anwendungssoftware, die in einem Grid-Job benötigt wird, spezieller Anforderungen wie z.B. einem bestimmten Betriebssystem oder bestimmter Bibliotheken bedarf, so kann diese und somit der Grid-Job unter Umständen nicht auf einer oder mehreren der verfügbaren Ressourcen ausgeführt werden. Weiterhin ist es möglich, dass die benötigte Anwendung auf einer Ressource nicht verfügbar ist und nur mit Root-Privilegien (mit vertretbarem Aufwand) installiert werden kann. Unter Umständen erfordert eine Anwendung auch die Ausführung mit Root-Rechten. In all diesen Fällen können die betreffenden Ressourcen nicht für die Ausführung des Jobs genutzt werden.

Ein möglicher Ausweg hierfür besteht im Einsatz von Virtualisierungstechnologien um auf diese Weise einem Nutzer zu erlauben, die benötigte Laufzeitumgebung und/oder Anwendung zur Jobausführung in Form von virtuellen Maschinen (VMs) bzw. VM-Images mitzuliefern. Da die VMs auf vom Nutzer bereitgestellten Images basieren sollen, können diese erst vor Jobstart gestartet werden und sollten nach Beendigung des Jobs wieder heruntergefahren werden um nicht unnötig Ressourcen zu konsumieren. Das steht im Gegensatz zur Nutzung von VMs die wie physische Hosts statisch in ein Grid eingebunden werden und genau wie diese über die Grid-Middleware erreichbar sind.

Wir stellen in diesem Papier einen Service (im Folgenden als VM-Service bezeichnet) vor, der eine möglichst komfortable Ausführung von Gridjobs in VMs erlaubt. Der Service wurde im Kontext des vom BMBF geförderten D-Grid-Projekts OptiNum-Grid [1] entwickelt. Er basiert auf der quelloffenen Software Nimbus [2].

Der Rest des Papiers ist wie folgt aufgebaut: zunächst wird im folgenden Abschnitt auf verwandte Arbeiten eingegangen bevor im Abschnitt 3 kurz die Architektur von Nimbus erläutert wird. In Abschnitt 4 wird dann auf die Funktionsweise des VM-Services eingegangen. Die Nutzung des Services wird anhand eines Anwendungsbeispiels in Abschnitt 5 erklärt. Abschnitt 6 erläutert Aspekte zur Performanz bzw. dem zu erwartenden Overhead in Verbindung mit dem Service bevor das Papier mit einer kurzen Zusammenfassung abschließt.

2 Verwandte Arbeiten

Eine Software mit ähnlicher Zielstellung ist XenBee (Xen-Based Execution Environment) [3]. XenBee bietet einen Daemon, der über einen Message Queue Server von einem entfernten Rechner über einen entsprechenden Client kontaktiert werden kann. Der Daemon ist in der Lage auf dem lokalen Rechner eine Xen-basierte VM zu starten. Es ist notwendig, dass in der VM eine Komponente von XenBee, namens *xbeinstd*, nach dem Bootvorgang automatisch gestartet wird. Diese Komponente kann von dem XenBee-Daemon kontaktiert werden und ist für den eigentlichen Start von Jobs in der VM zuständig.

XenBee hat die Einschränkung, dass alle Input-Dateien über einen HTTP- oder FTP-Server verfügbar gemacht werden müssen, von dem die Dateien vor Jobstart bezogen werden können. Genauso müssen auch zu nutzende Images und Kernel bereitgestellt werden und Speicherstellen für Output-Dateien auf einem FTP- oder HTTP-Server angegeben werden. Weiterhin ist XenBee nur schlecht für den Betrieb in Clustern eines Grids geeignet. Da der Daemon von überall Verbindungen entgegen nimmt, kann die Grid-Middleware und das Batch-System des Clusters umgangen werden, außerdem kann er nur lokal VMs starten.

Neben XenBee ist den Autoren kein Projekt zum Start von Jobs in VMs bekannt. Es existieren lediglich etliche Cloud-Infrastrukturen wie zum Beispiel Eucalyptus, OpenNebula oder Nimbus. Diese ermöglichen Nutzern zwar den Start von VMs auf fremden Ressourcen, um alles weitere was die Jobsausführung in der VM betrifft, wie z.B. den Transfer von Inputdateien in die VM, muss sich der Nutzer allerdings selbst kümmern.

Mit dem Start von VMs mit Netzwerkanbindung und

dem Herunterfahren dieser im Auftrag von Nutzern beherrschen die angesprochenen Cloud-Infrastrukturen jedoch schon einen wichtigen Bestandteil der Jobausführung in VMs. Daher nutzen wir mit Nimbus einen solchen Service als Basis um darauf aufbauend alle noch fehlenden Funktionalitäten zur Jobausführung in VMs hinzuzufügen. Der folgende Abschnitt beschreibt kurz den Aufbau und die Funktionsweise von Nimbus.

3 Nimbus

Bei Nimbus handelt es sich um einen Cloud-Service, der im Globusumfeld entwickelt wurde und ursprünglich zum Start von VMs in Grids gedacht war (bevor die Cloud-Idee aufkam).

Abbildung 1 zeigt einen Überblick über die Architektur von Nimbus. Ein sog. Service Node fungiert als Frontend. Auf diesem wird ein Nimbus-Service ausgeführt, der Client-Anfragen sowohl über eine Webservice-Schnittstelle als auch über eine Amazon Elastic Compute Cloud (EC2) kompatible Schnittstelle entgegen nehmen kann. Auf mehreren Maschinen ist je ein sog. Virtual Machine Monitor (VMM) verfügbar. Dabei handelt es sich um ein Kommandozeilenprogramm, implementiert in Python, das in der Lage ist mit Hilfe von libvirt [4] Xen- oder KVM-basierte VMs [5, 6] zu starten. Die Netzwerkkonfiguration der VMs erfolgt über DHCP.

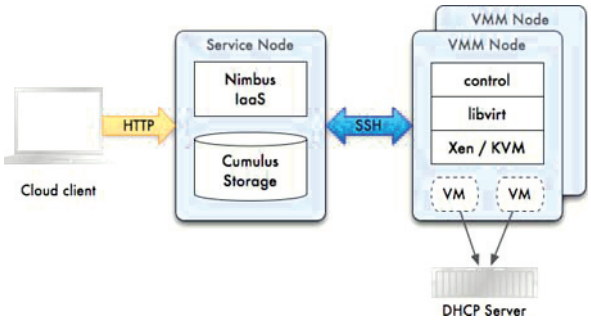

Bild 1. Nimbus-Architektur

Fragt ein Client beim Nimbus-Service eine virtuelle Maschine an, so wird über SSH einer der verfügbaren VMMs gestartet und auf der entsprechenden Maschine eine VM gestartet und die Kontaktinformationen für die VM an den Client zurück gegeben.

Nimbus ist somit für den transparenten Start von Grid-Jobs in VMs nur bedingt geeignet. Zum einen würde der Start von VMs an der Grid-Middleware und dem lokalen Batchsystem der Site vorbei laufen und zum anderen ist die Benutzung unkomfortabel. Der Nutzer müsste vor Jobausführung explizit eine VM starten, die Job-Inputs in sie transferieren, den Job z.B. per SSH selbst starten, die Outputs rücktransferieren und die VM letztendlich wieder runterfahren. All dies sollte möglichst automatisiert erfolgen. Wir haben aufbauend auf dem VMM einen Service implementiert, der diese Funktionalität bietet. Die Funktionsweise dieses Services soll im nächsten Abschnitt erläutert werden.

4 Funktionsweise

4.1 Überblick

Zunächst haben wir den VMM zu einem Server umgeschrieben, der als Daemon auf Ressourcen eines Grids ausgeführt werden kann. Dies bietet den Vorteil, dass er stets über alle von ihm auf einer Ressource gestarteten VMs im Bilde ist. In der Originalarchitektur von Nimbus hat der Service Node den Überblick über alle laufenden VMs. Dieser entfällt nun, da kein zentraler Dienst für unsere Zwecke nötig ist. Der von uns umgeschriebene Server nimmt nur Verbindungen von *localhost* entgegen und kann über einen entsprechenden Client bedient bzw. kontaktiert werden. Das hat den Grund, dass das lokale Batchsystem für den Start von VMs möglichst nicht umgangen werden soll. Der Service läuft unter einem privilegierten Linux-Account, der z.B. das Recht hat virtuelle Maschinen zu starten, was normalerweise nur dem root-User vorbehalten ist.

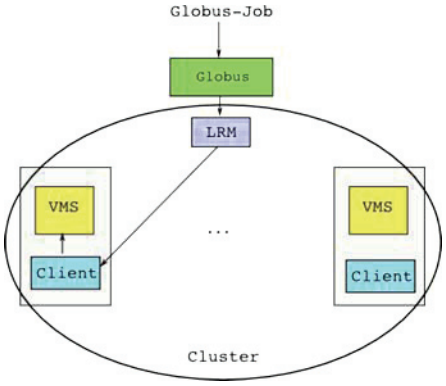

Bild 2. Überblick Architektur VM-Service

Die Abbildung 2 zeigt wie der Service in einem Grid eingesetzt werden kann. Auf den Knoten eines Clusters (oder einem Teil davon) läuft der VM-Service (VMS) in Form des Servers. Zudem ist auf allen Knoten der Client, z.B. im Home-Verzeichnis des Nutzers, verfügbar. Möchte der Nutzer nun einen Grid-Job in einer VM ausführen (hier über die Middleware Globus), so gibt er als Executable nicht die eigentliche Executable des Jobs an, sondern den Client, der den Namen der Executable als Argument erhält. Der Grid-Job wird nun vom lokalen Ressourcenmanager (LRM) - also dem Batchsystem - auf einem Knoten des Clusters gestartet. Der Client kontaktiert dort den Server und veranlasst den Start des Jobs in einer VM.

4.2 Jobausführung

Die Ausführung eines Jobs über den Service geht wie folgt vonstatten:

Um eine VM zu starten wird ein Image benötigt. Dieses wird vom Nutzer bereit gestellt bzw. der Nutzer muss auf dem Host, wo die VM gestartet werden soll, Zugriff auf ein entsprechendes Image haben. Der Nutzer startet nun den Client, dem er den Pfad zum Image zusammen mit weiteren Argumenten (wie z.B. die zu startende Executable) übergibt. Der Client kontaktiert

den Server auf dem selben Host und dieser prüft, ob noch VMs gestartet werden können. Der Administrator gibt nämlich für jeden Host eine Reihe von VMs an, die dort gestartet werden dürfen. Sind diese für einen Host bereits alle in Benutzung so kann der Server dort vorerst keine neuen VMs starten. Wenn aber noch weitere VMs gestartet werden können, so gibt der Server dem Client einen Pfad zurück, in den der Client das Image kopiert. Dass bei jedem Jobstart eine Kopie des Images angefertigt wird hat mehrere Gründe: dadurch, dass der Client das Image kopiert, wird sichergestellt, dass der Nutzer Leserechte auf das Image hat und sich nicht unberechtigten Zugriff darauf verschafft. Außerdem wird so sichergestellt, dass das Image nach Jobausführung im Originalzustand verbleibt. Der wichtigste Grund ist, dass es andernfalls nicht möglich wäre mehrere VMs von ein und dem selben Image gleichzeitig zu starten.

Wenn das Image kopiert wurde, hängt der Server das Image (bzw. die Kopie) in ein temporäres Verzeichnis ein und gibt dem Client den Pfad zum eingehängten Image zurück damit dieser alle benötigten Inputdateien in ein Verzeichnis des eingehängten Images kopieren kann (genauer gesagt, werden alle Dateien unter */root* im Image abgelegt). Um später den Job per SSH starten zu können, wird vom Server nun ein SSH-Schlüsselpaar erzeugt und der öffentliche Schlüssel in die */root/.ssh/authorized_keys* im eingehängten Image geschrieben. Danach wird das Image vom Server wieder ausgehangen, die VM gestartet und gewartet bis die VM auf Port 22 erreichbar ist. Sobald die VM erreichbar ist, wir der Job per SSH gestartet. Die Ausführung erfolgt als root-User und das Arbeitsverzeichnis ist */root* in der VM. Nach Jobausführung wird der Standard-Output und -Error an den Client zurückgegeben und die VM wieder heruntergefahren. Abschließend wird das Image nochmal eingehangen und der Pfad an den Client gegeben damit dieser die gewünschten Output-Dateien herauskopieren kann.

Da Images u.U. vergleichsweise groß sind, können sie dem Client auch in gepackter Form übergeben werden. Dieser entpackt sie dann vor der Benutzung. Da oftmals ein großer Teil des Images aus ungenutztem Speicherplatz der VM besteht kann so in vielen Fällen eine große Menge an Speicherplatz eingespart werden.

Der Nutzer hat zudem die Möglichkeit, die Inputs nicht als einzelne Dateien, sondern in einem Extraimage zu übergeben, welches beim Start der VM in diese eingebunden wird. Analog können dann auch die Outputs in diesem Image rücktransferiert werden.

Wenn das VM-Image keinen eigenen Kernel enthält, so wird ein Standard-Kernel für die VM benutzt, der vom Administrator angegeben wird. Das kann z.B. der Kernel des physischen Hosts sein.

5 Anwendungsbeispiel

Der VM-Service könnte zum Beispiel genutzt werden um VHDL-Simulationen mittels GHDL auf Grid-Ressourcen durchzuführen, auf denen GHDL nicht vorinstalliert ist. Die Installation von GHDL über Paketquellen oder über vorkompilierte Binärdateien erfordert nämlich Root-Rechte. Eine Installation über die Quelldateien ist aufwändig da dafür wiederum ein Ada-Compiler installiert sein muss mit dessen Hilfe der C-Compiler *gcc* mit speziellen Optionen compiliert werden muss. Sollte der Administrator der Grid-Ressourcen nicht gewillt oder rechtzeitig in der Lage sein, GHDL mit Root-Rechten zu installieren, kann man es trotzdem auf den Ressourcen nutzen, wenn dort ein VM-Service zur Verfügung steht.

Angenommen es soll eine VHDL-Testbench *tb* per GHDL auf einer Grid-Ressource simuliert werden. Der Nutzer könnte ein Image *img* mit z.B. einer Standardinstallation von Debian erstellen oder beziehen. In dieses Image installiert er GHDL (mit Hilfe der Binärdateien wäre dafür nur ein Einhängen des Images und das Kopieren der Dateien nach */usr/local* nötig) und transferiert das Image per GridFTP auf die zu nutzende(n) Ressource(n). Dort könnte er die Simulation in dem bereitgestellten Image über den in Abbildung 3 dargestellten Aufruf des VM-Service-Clients starten.

```
workspace-client.sh -a execute -i ~/img -c "ghdl -r tb --vcd=tb.vcd" \
--inputs="~/tb,tb" --outputs="tb.vcd,~/tb.vcd"
```

Bild 3. Aufruf des Clients des VM-Services zur Ausführung einer GHDL-Simulation. Über "-a execute" wird angegeben, dass ein Job ausgeführt werden soll, über "-i" das zu nutzende Image und "-c" das auszuführende Kommando. Der Transfer von In- und Output-Dateien wird über die Argumente "–inputs" und "–outputs" angegeben.

Damit würde eine VM vom Image *img* gestartet, die Datei *tb* in die VM transferiert und dort per GHDL simuliert (durch den Aufruf *ghdl -r tb –vcd=tb.vcd*). Die daraus resultierende Datei *tb.vcd* mit den Signalverläufen würde letztendlich wieder aus der VM auf den physischen Host kopiert, von wo sie vom Nutzer bezogen werden kann. Den in Abbildung 3 gezeigten Aufruf könnte er in ein Bash-Skript *skript.sh* schreiben, welches sich über ein Globus-Skript wie in Abbildung 4 auf einer Grid-Ressource starten ließe.

```
<job>
 <executable>${GLOBUS_USER_HOME}/skript.sh</executable>
  <fileStageIn>
   <transfer>
    <sourceUrl>gsiftp://userhost.de:2811/home/tb</sourceUrl>
    <destinationUrl>file:///${GLOBUS_USER_HOME}/tb</destinationUrl>
   </transfer>
  </fileStageIn>
  <fileStageOut>
   <transfer>
    <sourceUrl>file:///${GLOBUS_USER_HOME}/tb.vcd</sourceUrl>
    <destinationUrl>gsiftp://userhost:2811/home/tb.vcd</destinationUrl>
   </transfer>
  </fileStageOut>
</job>
```

Bild 4. Globus-Skript für das beschriebene Beispiel einer GHDL-Simulation.

6 Performanz

In diesem Abschnitt soll untersucht werden, in wie fern sich die Nutzung des VM-Services auf die Performanz

auswirkt. Zunächst haben wir Messungen durchgeführt, um den Overhead zum Kopieren des Images, Starten der VM etc. zu bestimmen. Diese wurden auf einem Knoten eines Clusters ausgeführt, der mit zwei Opteron 2435 Hexcores (2,6 GHz Taktfrequenz) ausgestattet ist und über 32 GB Arbeitsspeicher verfügt. Das Cluster hat ein verteiltes Raid5-Dateisystem. Wir haben zehn mal den Befehl */bin/true* über den VM-Service ausgeführt und die benötigten Ausführungszeiten bestimmt. Zumal */bin/true* nichts weiter macht als sofort mit dem Rückgabewert 0 zurückzukehren, entsprechen die gemessenen Ausführungszeiten praktisch den Zeiten, die vom Service benötigt wurden, um die VM zu starten und in dieser den Job zu starten. Für die VM haben wir ein 4 GB großes Image einer Debian-Installation benutzt. Die durchschnittliche gemessene Ausführungszeit beträgt 41,9 s, wobei ein Großteil für das Kopieren des Images benötigt wurde. Solch ein Overhead würde sich natürlich stark auf die Ausführungszeit sehr kurzer Jobs niederschlagen. Allerdings kann der durch die Grid-Middleware und das lokale Batchsystem entstehende Overhead unter Umständen noch größer sein, wodurch sich kurze Jobs im Allgemeinen sowieso nicht für die Ausführung in Grids eignen.

Wie sich die Virtualisierung auf die Ausführungszeit von Anwendungen auswirkt, ist nur schwer zu sagen. Che et al. haben 2010 mit Hilfe gängiger Benchmarks die Performanz von Xen-basierten VMs, OpenVZ-basierten VMs, KVM-basierten VMs und einem nativen System verglichen [7]. Laut den sich daraus ergebenen Messwerten wirkt sich die Virtualisierung durch Xen praktisch nicht auf die reine Rechenleistung aus. Speicherzugriffe können allerdings bis zu etwa 10 % und Festplattenzugriffe um bis zu 5 % durch Virtualisierung verlangsamt werden (die Werte sind Abschätzungen auf Grundlage der in der Arbeit enthaltenen Graphen, genaue Werte sind dort nicht angegeben). Die Werte können aber auf anderen Systemen wie dem Test-System abweichen, da sie von vielen Faktoren wie der eingesetzten Linux-Distribution, dem Kernel, der Xen-Version etc. abhängen. Allgemein kann man wohl sagen, dass für IO-intensive Anwendungen mit Geschwindigkeitseinbußen zu rechnen ist, für CPU-intensive Anwendungen hingegen weniger.

Das Kopieren von Input-Dateien in eine VM und das Kopieren von Output-Dateien aus einer heraus verursacht einen weiteren Overhead. Jedoch findet das Kopieren ja nur lokal auf dem Host statt, so dass dies erst ab Dateigrößen von mehreren GB merklich ins Gewicht fallen sollte. Auf unserem Testsystem ergibt sich bspw. ein Durchsatz von ca. 320 MB/s für das Kopieren einer 200 MB großen Datei auf dem verteilten Dateisystem.

Der Transfer des Images auf die Grid-Ressource kostet zusätzliche Zeit. Für den Transfer einer 200 MB großen Datei zwischen unserem Cluster in Erlangen und dem der GWDG in Göttingen mittels GridFTP unter Nutzung von 16 parallelen Datenströmen haben wir eine Transferrate von ca. 30 MB/s gemessen. Ein 3 GB großes Image würde demnach ca. 102 s Transferzeit kosten und damit einen relativ großen Overhead verursachen. Wenn ein Image allerdings für eine Vielzahl von Jobs genutzt werden kann und somit nur einmal transferiert werden muss, fiele diese Zeit nicht mehr so gravierend ins Gewicht.

Zusammenfassend lässt sich zur Performanz sagen, dass das Nutzen des VM-Services immer mit einem gewissen Overhead verbunden ist. Wie groß dieser Overhead in absoluten Zeiten und relativ zur Gesamtausführungszeit ist, hängt vom konkreten Job ab. Bei kurzen IO-intensiven Jobs, die den zuvorigen Transfer eines Images bedürfen, kann sich der Overhead sehr stark bemerkbar machen während er bei längeren CPU-intensiven Jobs, die ein bereits auf der Ressource vorhandenes Image nutzen, nicht stark ins Gewicht fallen sollte.

7 Zusammenfassung

Basierend auf einer Komponente von Nimbus haben wir einen Service implementiert, der den transparenten Start von Jobs in virtuellen Maschinen ermöglicht. In- und Output-Dateien werden automatisch in die virtuelle Maschine transferiert, der Job automatisch in ihr gestartet und die virtuelle Maschine wird nach Fertigstellung des Jobs automatisch wieder beendet. So hat ein Grid-Nutzer die Möglichkeit, auf fremden Grid-Ressourcen seine eigene Laufzeitumgebung für seine Jobs bereitzustellen. Für einen Großteil der in der Praxis vorkommenden Anwendungen sollte der durch den Service verursachte Overhead in einem erträglichen Rahmen bleiben. Als zukünftige Erweiterung käme die Ermöglichung der Job-Ausführung in KVM-basierten VMs in Frage, da KVM bereits grundlegend durch die genutzte Nimbus-Komponente VMM unterstützt wird. Außerdem wäre es sinnvoll dem Nutzer die Möglichkeit zu geben, die benötigte Anzahl an Rechenkernen oder Menge an Arbeitsspeicher für die zur Jobausführung genutzte VM anzugeben. Solche Ressourcenanforderungen müssten vom VM-Service verwaltet werden um sicherzustellen, dass auf einem Host nicht insgesamt von allen laufenden VMs mehr Ressourcen angefordert werden als vorhanden sind bzw. eine gewisse Obergrenze nicht überschritten wird.

Literatur

[1] Webseite OptiNum-Grid: http://www.optinum-grid.de/

[2] Nimbus-Dokumentation: http://www.nimbusproject.org/docs/2.7/

[3] Alexander Petry: “Design and Implementation of a Xen-Bases Execution Environment”. Diplomarbeit 2007

[4] Libvirt-Dokumentation: http://libvirt.org/docs.html

[5] Xen-Dokumentation: http://www.xen.org/support/documentation.html

[6] KVM-Dokumentation: http://www.linux-kvm.org/page/Documents

[7] Jianhua Che, Congcong Shi, Yong Yu, Weimin Lin: “A Synthetical Performance Evaluation of OpenVZ, Xen and KVM”. In: Services Computing Conference (APSCC), IEEE Asia-Pacific 2010