TU Clausthal

Clausthal University of Technology

Über die Eignung von Clouds für das Hochleistungsrechnen (HPC)

H. Richter,A. Keidel,R. Ledyayev

IfI Technical Report Series IfI-15-03

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Federico Schlesinger
Contact: federico.schlesinger@tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Inhaltsverzeichnis

Tabellenverzeichnis

Über die Eignung von Clouds für das Hochleistungsrechnen (HPC)

H. Richter,A. Keidel,R. Ledyayev

Technische Informatik und Rechnersysteme
TU Clausthal
hri@tu-Clausthal.de

Zusammenfassung

Cloud Computing hat sich zum allgegenwärtigen Vorbild für Rechen- und Speicherdienste entwickelt. Diese Vorteile machen Cloud Computing auch für Wissenschaftler attraktiv, denn sie müssen dadurch nicht mehr eine eigene IT-Infrastruktur vorhalten und betreiben, sondern können diese an sog. Cloud Service Provider auslagern, die die Wissenschaftler-IT virtualisiert betreiben. Allerdings wird die Situation komplizierter, wenn es um Hochleistungsrechnen geht, wie es beispielsweise bei Simulationen oder bei der Analyse großer Datenmengen der Fall ist. Die Gründe dafür sind, dass HPC-Anwendungen hoch effizient und skalierbar bleiben müssen, auch für den Fall, dass sie von vielen virtuellen Cores, Prozessoren oder Servern parallelisiert ausgeführt werden. Leider ist dies in Standard-Clouds nicht der Fall, wie nachfolgender Artikel zeigt. Darin werden diverse Gründe für Cloud-Ineffizienzen bei der Verwendung von OpenStack als Cloud-Betriebssystem und bei OpenFoam als Beispiel-HPC-Code angeführt. Darüber hinaus werden Vorschläge gemacht, wie man diese Probleme lösen bzw. umgehen kann.

1 Einleitung

Cloud Computing hat sich zum neuen, allgegenwärtigen Vorbild für Rechen- und Speicherdienste entwickelt. Die Gründe dafür sind eine Kostenabrechnung nach tatsächlich verbrauchten Ressourcen und die Anpassbarkeit an Benutzerwünsche. Beispiele für die Anpassbarkeit an Benutzerwünsche sind das Starten und Stoppen von virtuellen Maschinen (VMs), was in [35] als „Elasticity“ bezeichnet wird. Zu dieser Anpassbarkeit gehört auch die freie Konfigurierbarkeit der Zahl der virtuellen Cores pro realer CPU und der Größe des virtuellen Hauptspeichers pro Server. Firmen, Institute und Einzelpersonen können davon profitieren, indem sie ihre Rechen- und Speicherbedarfe von kommerziellen Cloud-Anbietern (CSPs) erfüllen

lassen, denn CSP-Dienste erstrecken sich mittlerweile von einfachen Daten-Backups bis hin zu kompletten virtuellen Rechenzentren. Diese Vorteile machen Cloud Computing auch für Wissenschaftler attraktiv, denn sie müssen dadurch nicht mehr eine eigene IT-Infrastruktur vorhalten und betreiben, sondern können diese an CSPs auslagern, die die Wissenschaftler-IT als „virtualisierte IT“ betreiben. Allerdings wird die Situation komplizierter, wenn es um Hochleistungsrechnen (HPC) geht, wie es beispielsweise bei Simulationen oder bei der Analyse großer Datenmengen der Fall ist. Die Gründe dafür sind, dass HPC- Programme hoch effizient und skalierbar bleiben müssen, auch für den Fall, dass sie von mehreren oder sogar vielen virtuellen Cores, Prozessoren oder Servern parallelisiert ausgeführt werden. Leider ist dies in Standard-Clouds nicht gegeben, wie Messungen gezeigt haben. Beispielsweise stellte das US Department of Energy (DoE), das für das Hochleistungsrechnen in den U.S.A. zuständig ist, die Nützlichkeit von Clouds für HPC im Jahre 2011 in Frage [1]. Andere Autoren, die dieselbe Meinung haben, sind beispielsweise [4], [5] und [6]. Wir glauben, dass weitere Forschungsanstrengungen notwendig sind, um die Ausführungseffizienz von Clouds bei HPC zu verbessern, dass dies aber möglich ist.In diesem Beitrag werden diverse Gründe für Cloud-Ineffizienzen bei der Verwendung von OpenStack als Cloud-Betriebssystem und von OpenFoam als Beispiel-HPC-Code angeführt. Es werden zudem Vorschläge gemacht, wie man diese Probleme lösen bzw. umgehen kann. Die Ergebnisse können auf andere Cloud-Betriebssysteme und andere HPC-Codes übertragen werden. Der Beitrag ist wie folgt aufgebaut: In Kapitel 2 wird der Stand der Technik rezensiert. Kapitel 3 beschreibt unser Projekt und welche Ausrüstung und Werkzeuge wir dafür verwendet haben. In Kapitel 4 werden die durchgeführten Messungen und Ergebnisse präsentiert und diskutiert. Der Artikel endet mit einem Ergebniskapitel, gefolgt von einem Ausblick und einer Literaturliste.

2 Stand der Technik

Es wurden von uns verschiedene wissenschaftliche HPC-Projekte studiert, die sich mit Optimierungen und Scheduling in Clouds beschäftigten. Außerdem wurde eine gründliche Literaturstudie durchgeführt, um herauszufinden, welche Cloud-Eigenschaften verbessert werden sollten, um Clouds HPC-fähig zu machen. Die Fachaufsätze, die in diesem Kapitel rezensiert werden, sind [4] - [11] (in der Reihenfolge ihrer Bedeutung). Auch die Autoren dieses Beitrags haben bereits über die Eignung von Clouds für HPC geforscht [12]. In [4], [5] und in diesem Artikel wird festgestellt, dass der Verwaltungszusatzaufwand (Overhead) aufgrund der Virtualisierung zu einer schlechten Leistungsfähigkeit bei der Interprozesskommunikation führt. Die neuesten Hardware-Beschleuniger für virtualisierte Kommunikation wurden dabei nicht berücksichtigt, da uns diese zum Zeitpunkt der Untersuchungen

nicht zugänglich waren. Gemäß den studierten Artikeln ignorieren existierende Cloud Scheduler die Bedürfnisse von HPC-Anwendungen, ebenso wie die Heterogenität und die gleichzeitige Vielfach-Vermietung (Multi-Tenancy) derselben Cloud-Ressourcen. Es wird gesagt, dass das die Engpässe sind, die effektives HPC in Clouds verhindern. Um die bezeichneten Probleme anzugehen, haben beispielsweise die Autoren von [4] und [5] den „Nova“ Scheduler von OpenStack mit der Information ausgestattet, dass es sich um HPC-Anwendungen handelt,was in einer Leistungssteigerung von 45% resultierte und ein bemerkenswertes Ergebnis darstellt. Im Artikel [6] wird die Idee einer verbesserten Platzierung virtueller Maschinen (VMs) durch den Nova-Scheduler vertieft. Die Autoren haben Nova modifiziert, um ihn über die zu Grunde liegende Cloud Hardware, die Topologie des Verbindungsnetzwerkes, die Gruppierung von Cloud-Ressourcen und über Störungen zwischen Jobs in Form von sog. „lauten Nachbarn“ (Noisy Neighbors) informieren zu können. In [7] wird eine Übersicht über Scheduling-Verfahren gegeben, die auch für Clouds relevant ist. In [8] werden Inter-Cloud Meta-Scheduler diskutiert, die die Systemdynamik, die Interoperabilität und die Heterogenität verschiedener Clouds berücksichtigen. Die Absicht dabei ist, die Eigenschaften eines gegebe- nen HPC-Codes herauszufinden und daraus ein Modell abzuleiten, das die Ressourcenanforderungen dieser Jobklasse in sog. kooperativen E-Science- Infrastrukturen widerspiegelt. Artikel [9] berichtet über ein verteiltes Job- Managementsystem, das Millionen kleiner HPC-Jobs verwalten können soll. Dieses System zielt auf die großen kommerziellen CSPs ab, wie Amazon und Google. Der Fokus liegt dabei auf hohem Durchsatz und guter Auslastung, was genau das ist, was CSPs sich wünschen. In [10] wird die Bezeichnung „HPC-as-a-Service“ als ein neuer CSP-Dienst eingeführt. Das beschriebene Projekt versucht, die Lücke zu schließen zwischen dem, was ein CSP in jedem Moment seinen Kunden als Rechenressourcen zu einem bestimmten Preis anbieten kann, und dem, was die Kunden in diesem Augenblick haben wollen. Es wird erklärt, dass beide Seiten (CSP und Kunden) große Streubreiten und Inhomogenität aufweisen. Es wird weiterhin aufgezeigt, dass deswegen eine multi-kriterielle Optimierung der Cloud-Ressourcen erforderlich sei. Die Autoren verwenden dafür eine Methode, die sie als „gemischte Ganzzahl-Linearprogrammierung“ bezeichnen, sowie ein stochastisches Optimierungsmodell für die effiziente Verwendung gemeinsamer HPC-Ressourcen zur Bereitstellung von Diensten. Ihr Fokus liegt auf dem Kosten/Nutzen-Verhältnis von Cloud-Ressourcen.In [11] wird berichtet, dass Scheduler in Hypervisoren wie z.B. in XEN nicht gleichermaßen gut die Last von solchen Jobs handhaben können, die sich stark im Rechenleistungsbedarf unterscheiden. Der Grund dafür läge darin, dass die Inter-VM-Kommunikation im selben HPC-Job dadurch herabgesetzt würde, dass kommunizierende VMs im Moment der Kommunikation descheduled werden könnten. Die Autoren schlagen vor, nicht alle VMs einer Cloud gemeinsam zu schedulen, sondern das Scheduling nur innerhalb von isolierten Teilmen-

gen von Cloud-Ressourcen zwar parallel aber isoliert voneinander durchzuführen, um die gegenseitige Beeinflussung von Jobs zu begrenzen. Sie verwenden dazu ein Vorhersagemodell samt Software-Implementierung desselben, um eine Prognose darüber abzugeben, welche VM mit welcher anderen in Zukunft kommunizieren wird, um ein Descheduling zum falschen Zeitpunkt zu vermeiden. Außer- dem verschieben sie kommunikationsintensive Gruppen von VMs auf andere Teile der Cloud-Ressourcen, um solche HPC-Codes effizienter in der Cloud zu machen. Das wird durch einen Scheduler bewerkstelligt, der Kenntnis von den IO-Aktivitäten, d.h. von den Kommunikationsbeziehungen der VMs untereinander hat.Die zitierten Artikel haben zu HPC-effizienteren Clouds beigetragen. Allerdings haben nicht wenige Artikel keine realen Clouds, sondern Cloud-Simulatoren benutzt, oder sie haben keine realen HPC-Codes verwendet, sondern synthetische Lastgeneratoren. Aus unserer Sicht ist es deshalb nicht immer klar, wie realistisch die erzielten Ergebnisse sind. Deshalb haben wir die Richtung verfolgt, echte Hardware einzusetzen und ein anerkanntes HPC-Programmpaket zu benützen, um damit Messungen auf einer Standard-Cloud durchzuführen, um dadurch zuverlässigere und realistischere Ergebnisse zu erzielen.

3 Projektbeschreibung

Für unser Projekt haben wir eine eigene Cloud aufgebaut, OpenStack [2] installiert und verwenden OpenFoam [3] als Benchmark. Außerdem wurden Skripte geschrieben, um OpenFoam in diversen Parameter-Konfigurationen automatisiert auszuführen und seine Laufzeit zu messen.

3.1 Einfachere Möglichkeiten

Bevor wir mit dem Bau einer eigenen Cloud begannen, haben wird die nachfolgenden einfacheren Möglichkeiten untersucht: 1.) Installation des Cloud OS auf virtuellen Maschinen anstelle von realer Hardware (sog. Nested Virtualisation), 2.) Verwendung eines Cloud-Simulators und 3.) Durchführung der Messungen auf einer bestehenden Universitäts-Cloud oder einer kommerziellen Cloud. Alle drei Möglichkeiten wurden evaluiert und im Folgenden wird erläutert, warum wir keine davon verwendet haben.

3.1.1 Doppelte Virtualisierung

Wir haben durch Messungen ermittelt, dass bereits eine einfache Virtualisierung, die nicht geschachtelt ist, die Effizienz von HPC-Programmen erheblich schmälert, sofern man nicht die neuesten Hardware-Beschleuniger z.B. von Intel für virtualisiertes Rechnen und virtuelle Kommunikation verwendet. Diese sind z.B. in [19] beschrieben. Dementsprechend ist beim Einsatz

doppelter oder mehrfacher Virtualisierung zusätzlicher Overhead zu erwarten. Die Installation einer Cloud in einer oder mehreren VMs war deshalb keine Option für HPC.

3.1.2 Cloud-Simulatoren

Wir begannen aus diesem Grunde, Cloud-Simulatoren auf Open Source Basis in Erwägung zu ziehen. Dazu zählten CloudSim [14], GreenCloud [15], iCanCloud [16] und eine verbesserte Version des sog. MaGateSim Simulators, die in [18] beschrieben wird. GreenCloud und MaGateSim sind für energiesparendes Cloud Computing gemacht, was nicht in unserem Fokus lag. Sie sind außerdem zu begrenzt für unsere geplanten Leistungsanalysen. iCanCloud ist hilfreich, um den Zielkonflikt zwischen Leistungsvermögen einer Cloud und dazu notwendige Kosten zu bewerten, was für uns ebenfalls nicht relevant war. Unsere Hoffnung lag deshalb bei CloudSim. Ein genauerer Blick zeigte allerdings, dass CloudSim nur über ein sehr eingeschränktes Modell für die Kommunikation verfügt, das nicht in der Lage ist, die komplexe, MPI-basierte Kommunikation von OpenFoam nachzubilden, und auch nicht die virtualisierte Kommunikation von OpenStack. Beispielsweise wird in CloudSim keine Differenzierung zwischen Inter-Core-, Inter-Prozessor- und Inter-Server-Kommunikation vorgenommen. CloudSim war daher zu limitiert für das, was wir benötigten. Darüberhinaus wurde in [17] festgestellt, dass die Ergebnisse von CloudSim nicht realistisch seien. Deswegen haben die Autoren von [17] eine substantielle Erweiterung mit dem Namen „NetworkCloudSim“ entwickelt, die die Cloud-Kommunikation durch ein erweitertes Bandbreite/Latenz-Modell unter- stützt. Daraus schlossen wird, dass von allen zum damaligen Zeitpunkt verfügbaren Simulatoren höchstens NetworkCloudSim für uns relevant sein könnte. Grundsätzlich ist es auch möglich, von NetworkCloudSim zu profitieren, indem man Modelle von Cloud-Anwendungen simulativ exploriert. Diese Modelle werden typischerweise dadurch definiert, dass man Job-Ausführungszeiten und die Kommunikation zwischen den Prozessen eines parallelen Jobs vorab abschätzt. Allerdings konnten wir die Behauptung der Autoren, dass NetworkCloudSim eine präzise Bewertung von Scheduling-Verfahren in wissenschaftlichen und MPI-basierten Anwendungen erlaubt, einschließlich der Kommunikationsinfrastrukturen in Rechenzentren, nicht nachvollziehen. Das Problem war, dass die Autoren keinerlei Zahlen oder Beispiele von echten Messungen geliefert haben, um damit die vielen Parameter von NetworkCloudSim zu eichen. Außerdem kam hinzu, dass Hardware-Beschleuniger, wie z.B. die Single Root IO- Virtualization (SR-IOV) [19], [20], die für eine effiziente virtualisierte Kommunikation unverzichtbar sind, leider nicht in NetworkCloudSim enthalten sind. Sie müssen mit viel Aufwand selbst modelliert werden. Außerdem hat NetworkCloudSim kein Modell für die Inter-VM-Kommunikation über KVM und den vom KVM benutzte OpenVswitch. Ein solches Modell

wäre auch sehr schwierig in NetworkCloudSim zu realisieren, denn Open-Vswitch und die von ihm implementierten virtuellen Netze arbeiten nach dem Prinzip des „Software-Defined Networking"(SDN). Sie sind damit dynamisch variabel, was eine Herausforderung für jedes Modell darstellt. Des Weiteren verfügt NetworkCloudSim über keine fertige Möglichkeit, den Einfluss von Tunnelprotokollen wie GRE [22], sowie von virtuellen lokalen Netzen wie VLAN oder VXLAN [23] zu modellieren. Diese finden in Clouds aber sehr oft Verwendung, weshalb ein Ignorieren nicht möglich ist. Unsere diesbezügliche Literaturstudie zeigte außerdem, dass bislang auch keine sonstige Arbeitsgruppe versucht hat, die unterschiedlichen Konfigurationsparameter eines Hypervisors, die Hardwarebeschleuniger von Clouds, die virtuellen LANs oder die Tunnelprotokolle in NetworkCloudSim zu modellieren. Das bedeutete für uns, dass es nicht möglich war, realistische Simulationsergebnisse ohne erheblichen eigenen Softwareentwicklungs- und Eichaufwand zu erhalten. NetworkCloudSim bot uns keine Möglichkeit, die Kommunikationsstruktur einer HPC-Anwendung einigermaßen verlässlich mit den gegebenen Mitteln nachbilden zu können, so dass eine reale Cloud immer notwendiger wurde. Die einzige Frage war nur noch, ob eine eigene oder eine fremde reale Cloud die einfachere Lösung war.

3.1.3 Existierende kommerzielle oder universitäre Clouds

Wir lernten schnell, dass es bei kommerziellen oder universitären Clouds nicht möglich ist, im laufenden Betrieb die Verbindungsstrukturen zu ändern oder aktuelle Hardware-Beschleunigerkarten hinzuzufügen, weil solche Aktionen die produktiven Abläufe stören. Des weiteren wären Administratorrechte für eine fremde Infrastruktur notwendig gewesen, um Anpassungen vorzunehmen und Messungen durchzuführen, die nicht zu erhalten sind. Weitere Schwierigkeiten bei einer fremden Cloud waren, dass es bei keinem uns bekannten Cloud Service Provider (CSP) möglich ist, die Platzierung von VMs im Rechenzentrum des CSPs und die sonstige Last der Cloud, die parallel existiert, gezielt zu beeinflussen. Damit konnte nicht sichergestellt werden, dass durch das sog. Noisy-Neighbour-Problem Messfehler auftreten, was die Vergleichbarkeit von Messreihen unmöglich macht. Wir entschieden deshalb, dass alle drei diskutierten Optionen nicht gangbar seien und beschlossen, eine eigene Cloud aufzubauen.

3.2 Unsere Projekt-Cloud

Unsere Cloud umfasst 17 gebrauchte Server von Dell und Sun, die zum Zeitpunkt der Messungen alle älter als vier Jahre alt waren (Stand 2014), also nicht die neueste Server-Generation widerspiegeln. Insgesamt standen 76 Cores, 292 GB RAM und 19 TB Plattenspeicher zur Verfügung. Die Server sind

über 17 Infiniband Netzwerkkarten von je 40 Gbit/s und einem 40 Gbit/s Infiniband Switch gekoppelt. Die Netzwerkkarten sind vom Typ Mellanox MHQH19B-XTR und benützen QSFP-Kupferkabel, genau wie der Switch, der vom Typ Mellanox Infiniscale IS5023 ist. Der Switch hat 18 Ports mit einer niedrigen Port-zu-Port-Latenz von nur 100 ns. Parallel zu diesem Hochgeschwindigkeitsnetzwerk wurde ein Standardkommunikationssystem installiert, das 17 Ethernetkarten zu je 1 Gbit/s und einen Ethernet Switch mit 24 Ports zu je 1 Gbit/s umfasst, um Leistungsvergleiche durchführen zu können. Die im weiteren Verlauf des Artikels präsentierten Messungen wurden mit Ubuntu 14.04.01 als HostOS und mit Icehouse als OpenStack-Version gemacht. Als GuestOS wurde Ubuntu 12.04.05 verwendet.

3.3 Integration von Infiniband in unserer Cloud

Für die Integration von Infiniband in unserer Cloud wurden die virtuellen Ethernet-Netzwerkkarten, die die Standard-Benutzerschnittstelle von KVM darstellen, mit Hilfe eines sog. TAP-Gerätetreibers [25] realisiert. TAP simuliert einen Ethernet-Netzwerkanschluss in Software. Anwendun- gen, die TAP benützen, kommunizieren über Read-/Write-Dateizugriffe miteinander. Diese Dateizugriffe werden von TAP in Nutzlasten für virtuelle Ethernet-Datenrahmen übersetzt, die anschließend von OpenVSwitch [24] weitervermittelt werden. Genau wie KVM ist auch OpenVSwitch eine wichtige Komponente von OpenStack und wird vom Cloud-Betriebssystem initialisiert und konfiguriert. OpenVSwitch stellt nach der Konfigurierung für jeden TAP-Gerätetreiber einen virtuellen Switch-Port bereit, an dem die virtuellen Ethernet-Rahmen eingespeist werden. Um Datenschutz zwischen verschiedenen Kunden innerhalb derselben Cloud zu erreichen, wird außerdem von OpenStack und von KVM für die virtuellen Maschinen jedes Kunden ein eigenes virtuelles lokales Netz (VLAN [36]) etabliert, das unabhängig von den VLANs anderer Kunden ist. Datentransfers zwischen den Kunden-VLANs sind nicht möglich. OpenVswitch verarbeitet schließlich jeden virtuellen Ethernet-Rahmen so, dass der Rahmen innerhalb eines VLANs entweder an eine VM im selben Server zugestellt wird, oder dergestalt, dass ein Switch-Ausgang den Rahmen über das GRE-Protokoll [22] tunnelt und so verpackt an das HostOS weiterschickt. Der zweite Weg wird genau dann gewählt, wenn die Empfänger-VM auf einem anderen Server beheimatet ist als die Sender-VM. Da die Übertragung eines IP Packetes in einem Infiniband Frame nicht möglich ist, kommt noch das Zwischenprotokoll IP-over-Infiniband (IPoIB) [33] als Träger der IP-Pakete hinzu. Die skizzierte Lösung war die einzige Möglichkeit, Infiniband in OpenStack zu integrieren, da wir auf den Servern zum Zeitpunkt der Messungen nicht über den Hardwarekommunikationsbeschleuniger SR-IOV [19] verfügten.

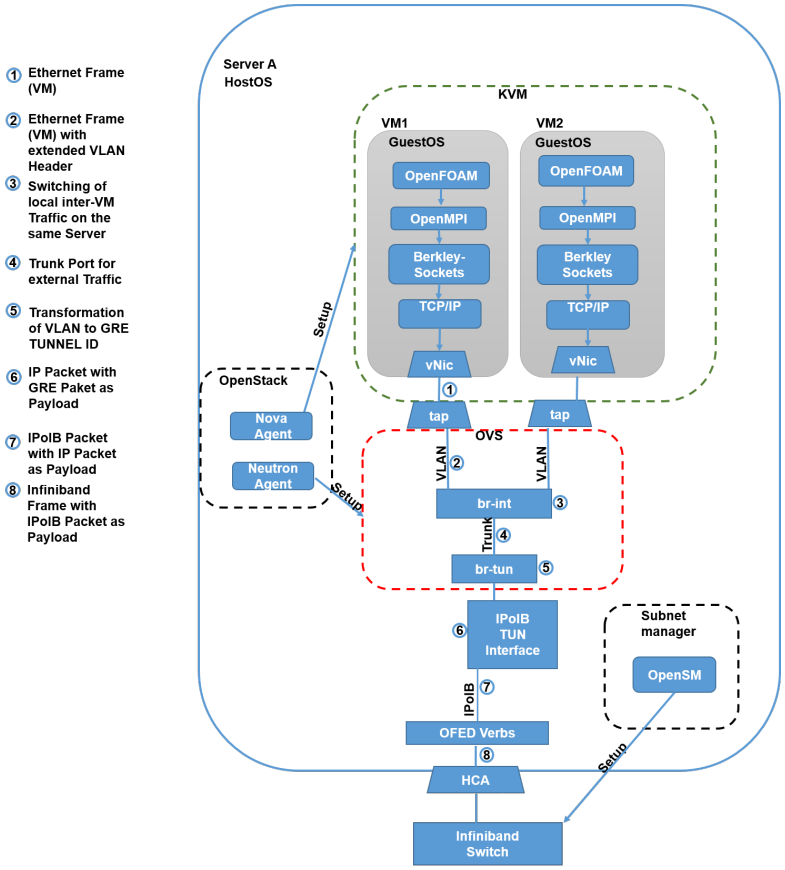

Abbildung 1: Infiniband in unserer Cloud

3.4 Unsere HPC-Anwendung

Als Beispiel für eine HPC-Anwendung in unserer Cloud wurde das weit verbreitete OpenFoam ausgesucht. Dabei handelt es sich um einen parallelen Code für die numerische Lösung von Laplace- und Navier-Stokes- Gleichungen. OpenFoam dient der Berechnung von laminaren und turbu- lenten Strömungen in komprimierbaren und inkompressiblen Fluiden, d.h. in Gasen und Flüssigkeiten. OpenFoam hat zusätzliche Gleichungslöser für allgemeine Teilchenflüsse, Verbrennungsvorgänge, Moleküldynamik, Wärmeleitung, für elektromagnetische Probleme, starre elastische Körper und andere Zwecke, die bei unseren Tests allerdings nicht verwendet wurden. Der Grund dafür ist der folgende: bevor wir mit den OpenFoam Bechmarks begannen, haben wir eine Befragung unter einigen OpenFoam-Nutzern durchgeführt,

um herauszufinden, was sie mit OpenFoam machen und welche Erwartungen sie an dessen Ausführung in einer Cloud hegen. Anhand dieser Befragung haben wir verstanden, dass es bei OpenFoam hauptsächlich um die Lösung der Navier-Stokes-Gleichungen geht, sowie worin die Probleme der Anwender liegen und wo unser Ausgangspunkt ist. Anhand der Messungen, die nachfolgend präsentiert werden, konnten wir auch die Meinung der befragten Benutzer bestätigen, dass OpenFoam und OpenStack in Standard-Clouds, die nicht die neueste Server-Generation beinhalten, keine effiziente Kombination sind.

3.5 Leistungstests

Anfangs konfigurierten wir OpenFoam so, dass das „Dammbruch"-Beispiel ausgeführt wurde, das dem Release 2.2.1 von OpenFoam beigefügt ist, weil dieses Beispiel gut dokumentiert ist. Darin gibt es 7700 Gitterpunkte für geometrische Objekte in zwei Dimensionen. Eine Sekunde in der Realität wird über 1000 Zeitschritte simuliert. Anschließend haben wir die ursprüngliche Konfiguration modifiziert, um fortgeschrittenere Tests durchzuführen. Alle Messungen wurden 50 Mal wiederholt, und der erste Durchlauf wurde gelöscht, um Einschwingvorgänge auszuschließen.

3.5.1 Messergebnisse

Die Programmausführungszeiten aller Testdurchläufe sind in Tabelle 1 gezeigt. Diese Ergebnisse wurden nachbearbeitet, indem die Beschleunigung und die Effizienz berechnet wurden. Beide Maße sind folgendermaßen definiert:
Def. 1: Die Beschleunigung S ist das Verhältnis der Ausführungszeiten eines sequentiellen Programms vor bzw. nach Virtualisierung und Parallelisierung.
Def. 2: Die Effizienz E ist der Ausnutzungsgrad von n Cores bzw. n OpenStack vCPUs und definiert als E=S/n.

3.5.2 Bewertung der Messergebnisse

In Set-Up 1a wurde eine Ausführungszeit von 144 s für eine Problemgröße von 7700 Gitterpunkten ermittelt, die als Referenz für alle nachfolgenden Messungen dient. Dieser geringe Wert für die Ausführungszeit deutet an, dass die Problemgröße gemessen an üblichen HPC-Standards zu klein ist. Die Berechnung eines „Dammbruchs" in nur 7700 Punkten ist aber das Standard-Beispiel in OpenFoam, weshalb auch wir es verwendet haben. Anhand von Set-Up 1b kann man sehen, dass eine Parallelisierung von OpenFoam in unserer Cloud dann gewinnbringend ist, wenn die Programmausführung im selben Server erfolgt. Allerdings zeigt sich, dass die Effizienz bei

Tabelle 1: Messergebnisse für die Set-Ups 1-7.

Set-Up	Wall-Time[s]	Speed Up	Efficiency[%]
1a: 1 core, bare metal	144	1	100
1b: 4 cores, 2 CPUs, 1 server, bare metal	46	3.1	78
2a: 1 vCPU per VM, 1 server, 1 KVM, 1 VM	180	0.8	80
2b: 4 vCPUs per VM, 2 CPUs, 1 server, 1 KVM, 1 VM	62	2.3	58
3: nested virtualization, 1 core, 2 KVMs, 2 VMs	-	-	-
4a: 1 vCPU per VM, 1 KVM, OpenStack, 1 VM	154	0.94	94
4b: 4 vCPUs per VM, 2 CPUs, 1 server, 1 KVM, OpenStack, 1 VM	60	2.4	60
5: 1 vCPU per VM, 8 CPUs, 4 server, 4 Ethernets, 4 KVMs, OpenStack, 4 VMs	320	0.45	11
6: 4 vCPUs per VM, 8 CPUs, 4 server, 4 Ethernets, 4 KVMs, OpenStack, 4 VMs	670	0.21	5
7a: 4 vCPUs per VM, 8 CPUs, 4 SUN server, 4 Infinibands, 1 KVM, OpenStack, 4 VMs	237	0.61	15
7b: 4 vCPUs per VM, 8 CPUs, 4 SUN server, 4 Infinibands, 4 KVMs, OpenStack, 4 VMs	998	0.14	4

nur 4 Cores bereits um 22 Prozentpunkte auf 78% abfällt. Gemäß der Dokumentation sollte OpenFoam auf einem Supercomputer oder Parallelrechner hingegen bis ca. 1000 Cores skalierbar bleiben. Ein Abfall um 22 Punkte bei 4 Cores ist deshalb ein Hinweis darauf, dass der Kommunikationsaufwand im Vergleich zum Rechenaufwand zu hoch ist, und bestätigt, dass die gewählte Problemgröße zu klein für eine gute Cloud-Skalierbarkeit ist. In Messung 2a zeigt sich, dass in unserem Setup die zusätzliche Virtualisierung einen Effizienzverlust von 20 Prozentpunkten zur Folge hat. Dies kann dadurch erklärt werden, dass nur ein rel. alter Hardware-Beschleuniger namens AMD-V in den Server-CPUs zur Verfügung stand, aber nicht neuere Verfahren, die zu geringeren Effizienzverlusten bei der Virtualisierung führen. Eine Übersicht über aktuelle Hardware-Beschleuniger von Intel und AMD ist

in [19] zu finden. Setup 2b zeigt, dass die gleichzeitige Anwendung von Virtualisierung und Parallelisierung die Effizienz um 42 Prozentpunkte schmälert, was durch die Addition der Messergebnisse von 1b und 2a bereits erwartet werden konnte. Setup 3 konnte nicht ausgeführt werden, weil die VM, die von KVM innerhalb einer anderen VM gemäß doppelter Virtualisierung erzeugt wurde, nicht in der Lage war, ihr Gast-Betriebssystem auszuführen. Die Ursache dafür ist unbekannt. Die Setups 4a und 4b zeigen, dass die Standard-Konfigurationsparameter, die OpenStack für VMs unter KVM erzeugt, keinen Effizienznachteil bei der Programmausführung mit sich bringen. Die Effizienz, die mit Hilfe von Open- Stack erzielt werden kann, ist mit der Effizienz, die von KVM alleine er- reicht wird, vergleichbar (Siehe dazu auch die Setups 2a und 2b). Ferner zeigt sich in Setup 4b, dass bei mehreren Cores und parallelem OpenFoam die Effizienz um weitere 18 Prozentpunkte im Vergleich zu der sog. „Bare-Metal-Programmausführung“ von Setup 1b abfällt. Dieser Effekt war nach der Messung von Setup 2a zu erwarten und zeigt, dass bei dem existierenden AMD-V-Beschleuniger die Virtualisierung einen erheblichen Effizienzverlust nach sich zieht, ohne dass dabei noch zusätzliche Netzwerkkommunikation mit im Spiel gewesen wäre. Kommt wie in Setup 5 die Netzwerkkommunikation noch hinzu, entsteht weiterer Cloud Overhead (siehe 3.5.3), und in Folge dessen ein Ansteigen der Ausführungszeit um 500 Prozent im Vergleich zu Setup 4b, bei 1 Gbit/s-Ethernet. Eine Überraschung war, dass der Einsatz von 40 Gbit/s- Infiniband anstelle von 1 Gbit/s-Ethernet nur eine Verbesserung um 26 Prozentpunkte zeigte. Wir hatten angesichts der 40-fachen Übertragungsrate viel mehr erwartet. Die Gründe für dieses Verhalten liegen unserer Meinung nach am Kommunikations-Overhead, der viel CPU-Zeit kostet. Dies wird bei der Betrachtung von Setup 6 und 7 deutlich: so ist die Ausführungszeit von Setup 6, bei dem Ethernet eingesetzt wurde, sogar um 26 Prozent schneller als Setup 7b mit Infiniband. Eine Erklärung dafür ist, dass bei Setup 7b kürzere Nutzlasten als vorher in den Datenrahmen auftreten, da dieselbe Problemgröße von 16 statt von 4 Cores bearbeitet wird. Kürzere Nutzlasten erhöhen aber bei gleichlangem Header den Kommunikations-Overhead. Ein weiterer Grund, der für den Anstieg der Laufzeit in Setup 7b verantwortlich ist, ist die kleinstmögliche Rahmenlänge (Minimum Transport Unit), die bei Ethernet 64 Byte beträgt, bei Infiniband aber 256 Byte, was 4 mal mehr ist. Diese Untergrenzen müssen eingehalten werden, auch wenn die Nutzlast klein ist, was in vielen Füllbytes resultiert. D.h., bei zu kleinen Problemgrößen werden von Infiniband vier Mal mehr Füllbytes verwendet als von Ethernet, weil nicht genügend Nutzdaten zur Verfügung stehen. Außerdem kommen durch die Verwendung von IPoIB [33] vier weitere Header-Bytes hinzu [33], was den Header im Vergleich zur Nutzlast noch größer werden lässt.

3.5.3 Diskussion des Kommunikations-Overheads

Die Kommunikation zwischen zwei VMs, die auf verschiedenen Servern untergebracht sind, wird als Nutzlast in virtuelle Ethernet-Rahmen umgepackt, die wiederum über einen TAP-Gerätetreiber, über OpenVSwitch, einen GRE-Tunnel, einen TUN-Gerätetreiber, sowie mit Hilfe von IP und von IPoIB übertragen werden. Dieser komplexe Vorgang stellt einen erheblichen Overhead dar. Im Folgenden wird dieser Overhead komponentenweise erläutert.

Kommunikations-Overhead im GuestOS Der Overhead im GuestOS entsteht dadurch, dass das VM-Betriebssystem nur über eine virtuelle Ethernet-Netzwerkschnittstelle (Network Interface Card, NIC) und über ein Kunden-VLAN mit anderen VMs desselben Kunden gekoppelt ist. Open MPI kann seine viel schnellere Shared Cache/SharedMemory-Kommunikation nur dann nutzen, wenn sie in derselben Multicore CPU oder im selben Server stattfindet. Im Falle der schnellen Shared Cache/SharedMemory-Kommunikation ist zudem der Kommunikations-Overhead geringer, weil der GRE-Tunnel, die TUN/TUP-Gerätetreiber, TCP/IP und IPoIB entfallen. Es konnte in diesem Zusammenhang leider nicht ermittelt werden, ob der KVM-basierte Speicherschutz zwischen VMs die schnelle Kommunikation von Open MPI nur dann erlaubt, wenn sie zwischen vCPUs derselben VM stattfindet.

Kommunikations-Overhead im HostOS Das HostOS der Sender-VM verwendet IP auf seinen Netzwerkschnittstellen, aber nicht TCP, so dass an der Kommunikation zwischen einer Sender und einer Empfänger-VM auf verschiedenen Servern pro Richtung insgesamt drei Internet-Protokolle beteiligt sind: 1xTCP+ 1xIP im Guest OS und 1xIP im Host OS. Dies erhöht den Kommunikations-Overhead. Immerhin wird das TCP im HostOS für die Kommunikation innerhalb der Cloud nicht benützt, um den Overhead-Anstieg zu begrenzen.

Kommunikations-Overhead in OpenVswitch im HostOS OpenVSwitch ist ein Multiplexer und ein Switch in Software, d.h. er arbeitet im Gegensatz zu echten Multiplexern und Switches virtuell und wird genau deshalb vom „Neutron“-Netzwerkdienst der Cloud verwendet. Das OpenStack Kommunikationsmodul „Neutron“ initialisiert den OpenVSwitch und setzt alle virtuellen Netzwerke in der Cloud auf. Dies ist von der Wirkung her ähnlich wie bei einem Software-Defined Networking (SDN). Während der Laufzeit trägt Neutron erfreulicherweise nicht zum Overhead bei. OpenVSwitch hingegen multiplext zur Laufzeit die Ethernet-Datenrahmen aller vCPUs einer VM auf eine virtuelle Ethernet-Strecke und schaltet alle solche Strecken, die an virtuellen Switch Ports anliegen, von Eingangs- auf Ausgangs-Ports

durch. Dazu liest OpenVSwitch den Guest-OS IP-Header in der Ethernet-Nutzlast des Datenrahmens und trifft anhand dieser Information eine L2 Switching-Entscheidung. D.h. anstelle eines L3 Routings wird ein L2 Switching gemacht. Das Switching basiert auf einer sog. OpenFlow Tabelle [27], die indirekt von Neutron herrührt. Insgesamt besteht OpenVSwitch aus drei Hauptkomponenten: 1.) dem Prozess ovs-vswitchd, 2.) dem Prozess ovsdb-server, die beide im Benutzeradressraum angesiedelt sind, sowie 3.) dem Prozess openvswitch.ko, der im Adressraum des HostOS läuft. Der demon ovs-vswitchd erstellt die OpenFlow Tabelle anhand der Konfiguration, die von Neutron vorgegeben wird. Das Multiplexing und das L2 Switching findet in openvswitch.ko statt. Der ovsdb-server schließlich dient zur Netzwerkadministration durch den Cloud Service Provider. Der Anteil von OpenVSwitch am Kommunikations-Overhead ist als hoch einzuschätzen, dennoch hat sich OpenVSwitch zum de facto Standard bei OpenStack Clouds entwickelt ([26]), da sich dies nur bei HPC-Anwendungen bemerkbar macht.

Kommunikations-Overhead in GRE und in IPoIB GRE ist ein Tunneling-Protokoll im HostOS, das für den Transport virtueller Ethernet-Rahmen über IP des Host OS sorgt. Für den Einsatz von GRE ist OpenVSwitch im Host OS verantwortlich. OpenVSwitch verwendet GRE, um zwischen VMs auf verschiedenen Servern Daten auszutauschen. Bei IP over Infiniband (IPoIB) handelt es sich um ein Protokoll, das Host-OS IP-Pakete über Infiniband transportiert. Der Zugang zu IPoIB wurde in Form eines TUN-Gerätetreibers [25] realisiert.

Kommunikations-Overhead im Hypervisor des HostOS Der Hypervisor ist an der inter-VM-Kommunikation u.a. durch die Aktualisierung seiner Shadow Page Tables beteiligt. Diese Aktivität entfällt, sobald modernere Virtualisierungs-Beschleuniger wie z.B. die Extended Page Tables (EPT) [19] verwendet werden.

Kommunikations-Overhead im Gerätetreiber und auf der Übertragungsstrecke Der Gerätetreiber für Infiniband ist vom Type OFED [30] und bietet zwei Betriebsmodi: Send/Receive und Remote DMA (RDMA). RDMA erlaubt es der Netzwerkkarte, direkt auf Daten der HPC-Anwendung zuzugreifen, ohne diese zuvor in Betriebssystempuffer umkopieren zu müssen, und bietet so eine höhere Bandbreite und geringere Latenz als Send/Receive. RDMA wurde jedoch nicht verwendet, da es ohne SR-IOV nicht von OpenStack unterstützt wird. Die Infiniband-Übertragungsstrecke schließlich stellt im Vergleich zur Shared Cache/Shared Memory-Kommunikation zwischen Cores auf demselben Chip bzw. zwischen CPUs auf demselben Server einen Engpass dar.

3.5.4 Potentielle Vorteile durch SR-IOV

Beim Einsatz von SR-IOV wird nach unserer Meinung der Kommunikations-Overhead signifikant geringer ausfallen, da viele Umwege und Zwischenstufen entfallen (siehe Abb. 2). Der erste Grund dafür ist, dass das vCPU-Datenmultiplexing direkt auf der SR-IOV-Infiniband-Netzwerkkarte (HCA) vorgenommen wird, die dazu in einem SR-IOV-fähigen PCIe Server Motherboard installiert sein muss. Der zweite Grund für den geringeren Overhead ist, dass SR-IOV einer VM den direkten Zugriff auf die HCA ermöglicht, unter Umgehung aller Betriebssystempuffer (=RDMA). Wie in Abb. 2 zu sehen ist,

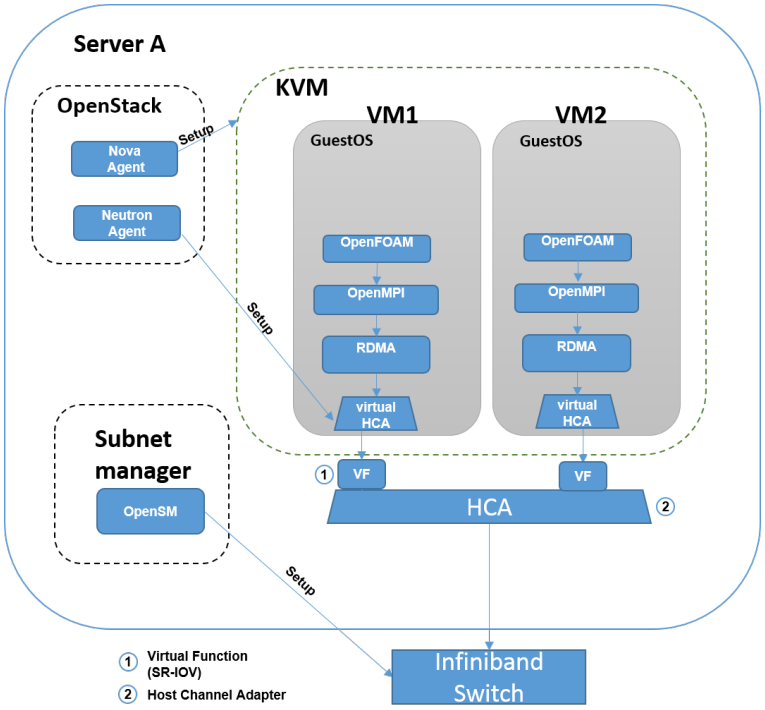

Abbildung 2: Infiniband mit SR-IOV

fällt der Overhead beim Einsatz von SR-IOV signifikant geringer aus. Da u.a. das Switching nicht länger in Software vorgenommen werden muss, sondern direkt auf der HCA / NIC erfolgt. Des Weiteren ermöglicht SR-IOV der VM den direkt Zugriff auf die HCA. Damit entfallen die Protokolle Ethernet, IP, TCP im GuestOS sowie die Zwischenprotokolle wie GRE,IP und IPoIB im HostOS. Ferner ermöglicht SR-IOV die Nutzung von RDMA durch MPI, was viel schneller ist als die Kommunikation über TCP/IP, da RDMA den sog. Kernel-Bypass unterstützt damit ist ein Direkt Zugriff auf einen Speicherbe-

reich des Zielsystems ohne Belastung des Kernels erlaubt. Die Kommunikation ist dadurch weitaus effizienter, da CPU Zeit gespart wird. Leider stand uns SR-IOV zum Zeitpunkt der Messungen nicht zur Verfügung. Außerdem muss der reale Infiniband Switch getrennt von OpenStack mit Hilfe eines sog. Infiniband Subnet Managers konfiguriert werden, weil OpenStack dies nicht kann. Aus Systemadministrationssicht stellt dies einen Nachteil dar.

3.5.5 Ergebnis und Schlussfolgerungen

Die Messungen auf unserer Cloud haben gezeigt, dass bei der Hardware- und Software-Konfiguration, die wir verwendet haben, die beste Beschleunigung für die Fälle erzielt wurde, bei denen Cores innerhalb derselben CPU oder desselben Servers Daten austauschen. Dies wird dadurch erklärt, dass Cores in derselben CPU über einen gemeinsamen Cache kommunizieren, und dass CPUs im selben Server über einen gemeinsamen Hauptspeicher Daten austauschen können. Open MPI nützt diese Möglichkeiten und schaltet automatisch von TCP/IP zwischen Servern auf die schnelleren Kommunikationsformen um, sofern dies möglich ist. Darüber hinaus haben unsere Messungen gezeigt, dass es nicht ausreicht, die Netzwerkinfrastruktur einer Cloud durch eine Hochgeschwindigkeits-Kommunikationstechnologie zu ersetzen, um dadurch Hochleistungsrechnen zu erzielen. Andere Faktoren müssen ebenfalls verbessert werden, wozu der Kommunikations-Overhead gehört. Dieser Overhead machte bei unserer Hardware- und Software-Konfiguration die wesentlich höhere Bandbreite und die geringe Latenz von Infiniband im Vergleich zu Ethernet zunichte, was ein wichtiges Ergebnis ist. Wenn der HPC-Code auf mehrere Server verteilt wurde, dann war es den beteiligten Cores nicht mehr möglich, den Effizienzverlust des Kommunikations-Overheads ausgleichen. Der Grund dafür war, dass kein moderner Hardware-Beschleuniger zur Unterstützung der virtualisierten Kommunikation zur Verfügung stand. Demzufolge wurde das Multiplexen und das Weiterschalten von Datenrahmen von OpenVSwitch vorgenommen. Außerdem war deswegen auch kein Direktzugriff (RDMA) der HPC-Anwendung auf den Infiniband-Gerätetreiber möglich, so wie er z.B. beim Intel VT-d-Beschleuniger [19] oder bei SR-IOV praktiziert wird. Vielmehr mussten zahlreiche Umwege in Form von Protokollen und virtuellen Gerätetreibern gegangen werden. Wir halten deshalb aktuelle Hardware-Beschleuniger zur Unterstützung der virtualisierten Kommunikation beim High-Performance Computing für unverzichtbar. Die erste Schlussfolgerung, die man aus den Messungen ziehen kann ist, dass einfache oder alte Standard-Clouds nicht für HPC geeignet sind, zumindest nicht in der Kombination OpenStack + KVM + OpenVSwitch. Andere Kombinationen wie z.B. OpenStack + VMWare ESXi + NSX haben wir nicht getestet. Die zweite Schlussfolgerung, die man aus unseren Literaturstudien ziehen kann, ist, dass Standard-Clouds deswegen bei HPC problematisch sind, weil

1.) die bestehenden Cloud-Scheduler, HPC-Bedürfnisse ignorieren, weil 2.) die Cloud-Hardware und die Anwender-Software heterogene Kommunikationspartner unterschiedlicher Leistungsfähigkeit zwangsweise zusammenschaltet, weil 3.) das gleichzeitige Vermieten ein- und derselben Cloud- Ressourcen durch den CSP separate VLANs zum Datenschutz notwendig macht, die aber den Kommunikations-Overhead in der Cloud erhöhen, und weil 4.) sich HPC-Kunden aufgrund des Noisy Neighbor-Effekts gegenseitig die Rechen-, Kommunikations- und Speicher-Ressourcen wegnehmen. Unser Resümee ist deshalb, dass ein Cluster bestehend aus Standard-Servern und OpenStack ohne die Anwendung einiger der nachfolgend beschriebenen Verbesserungsvorschläge für HPC nicht geeignet ist. Ebenso ist es nicht zielführend, ein Cloud-Betriebssystem auf einem vorhandenen Parallelrechner zu installieren, um so die Möglichkeiten nutzen zu können, die eine Cloud bietet, da dadurch die Cloud-Effizienzprobleme nicht gelöst werden.

4 Ausblick

Es ist unsere Arbeitshypothese, dass es möglich ist, jede Standard-Cloud in einen Parallelrechner für HPC zu verwandeln, sofern in der Cloud bestimmte Verbesserungen eingeführt werden. Diese Verbesserungen sind:

- Den Ersatz von Standard-Ethernet für die Inter-Server-Kommunikation durch ein schnelles Infinibandnetzwerk von mindestens 10 Gbit/s, besser 40 Gbit/s, das von Achtfach-PCIe-Schnittstellen gespeist wird: Dies ist eine Notwendigkeit, die schon lange bekannt ist.

- Den Einsatz von neuen Hardware-basierten Beschleunigern für virtualisierte Kommunikation. Dies spart sehr viel Overhead. Kommunikationsbeschleuniger übernehmen außerdem Hypervisor-Aufgaben zur Laufzeit der HPC-Anwendung, was ebenfalls schneller ist. Die von uns vorgeschlagenen Beschleuniger VT-d und SR-IOV haben allerdings auch Nachteile in Bezug auf die Systemadministration und den Datenschutz. Beispielsweise erlauben sie im Rahmen der Möglichkeiten von OpenStack keine Separierung in einzelne Kunden-VLANs. Alle Anwendungen der Cloud Nutzer befinden sich bei OpenStack und VT-d oder SR-IOV im selben Netz und können sich gegenseitig abhören, so dass ein Abwägen zwischen Kommunikationsleistung einerseits und IT-Sicherheit andererseits erforderlich ist. Eine Möglichkeit, das Abhör-Problem zu lösen, besteht in der Verwendung von sog. Infiniband-Partitionen. Diese erlauben die Unterteilung des Infiniband-Netzwerks in logische Gruppen und sind mit Ethernet- VLANs vergleichbar. Maximal sind so 32768 logische Infiniband- Netze realsierbar, was allerdings die Skalierbarkeit einer Cloud auf ebenso viele Kunden beschränkt.

Außerdem ist Neutron, d.h. OpenStack nicht in der Lage, Infiniband-Partitionen über den Infiniband-Subnetzmanager einzurichten und zu verwalten, was die Handhabbarkeit dieser Lösung weiter einschränkt.

- Korrektes Konfigurieren und Aktivieren der zuständigen BIOS-, Hypervisor- und Cloud OS-Optionen und Flags, um die zuvor erwähnten Beschleuniger voll ausnutzen zu können: Dies ist zwar eine Selbstverständlichkeit, aber in der Praxis schwierig zu realisieren, weil dazu erhebliches Experten wissen notwendig ist.

- Vermeidung der sog. „Überbuchung“ (over-commiting) von Speicher und Cores durch zu viel Virtualisierung. Die Zahl der gestarteten VMs pro Server sollte begrenzt und überwacht werden, genauso wie die maximale Größe von virtuellem Hauptspeicher pro Core. Auch diese Maßnahme ist schon lange bekannt, aber deshalb keineswegs unwichtig.

- Vermeidung von exzessivem Paging in Guest OS und Host OS durch eine passende Allokation von gestarteten Jobs zu verfügbaren Speicherressourcen in den einzelnen Servern.

- Vermeidung von mehrfacher Virtualisierung (nested virtualization).

- Konsequenter Einsatz von Open MPI bei allen HPC-Anwendungen aufgrund seiner Effizienz, der automatischen Wahl des besten verfügbaren Kommunikationspfades und seiner Zurückhaltung bei der Verwendung von TCP/IP.

- Ersatz von TCP/IP durch „Stubs“ wie z.B. den Mellanox Messaging Accelerator MXM [19] oder das Myrinet Open-MX [19]: Dies ist deswegen möglich, weil OpenStack L3 Routing durch L2 Switching ersetzt. Solange die Cloud nicht über das Internet verteilt ist, ist dies eine sehr gute Verbesserung.

- Ersatz der bisherigen Guest OS und Host OS Scheduler durch Scheduler, die Prioritäten unterstützen, sich der Cloud-Hardware, ihrer Verbindungstopologie und des Problems der „lauten Nachbarn“ bewusst sind, und die verschiedene Job-Klassen unterstütze:. Die Job- Klassen sollten das ganze Spektrum von "Low Performane Computing"bei Web Servern bis High Performance Computing bei numerischen Anwendungen abdecken.

- Hinzufügen von „Gang Scheduling“ [37] in den Cloud Scheduler, um miteinander kommunizierende VMs möglichst auf benachbarte Cores im selben Server oder sogar auf Cores derselben CPU zu schedulen, um die Latenz der Interprozesskommunikation zu minimieren.

- Ein analoges Gang Scheduling sollten auch der Guest OS und der Host OS Scheduler aufweisen, um miteinander kommunizierende VMs gleichzeitig schedulen zu können. Da es sich bei der MPI-Kommunikation oft um ein blockierendes Send und Receive handelt, d.h. um ein Rendezvous zwischen Sender und Empfänger, bei dem beide gleichzeitig über denselben Kanal Daten austauschen müssen, ist das gleichzeitige Scheduling von Sender und Empfänger die Voraussetzung für ein effizientes Rendezvous.

- Vermeidung von Warten auf IO-Ressourcen bei IO-intensiven Jobs: Der Cloud Scheduler sollte solche Jobs möglichst auf Hardware mit schneller Peripherie schedulen. Dies kann mit Hilfe einer Jobablauf-Beschreibungssprache (Job Control Language, JCL) erfolgen, die den Workflow im Job für den Scheduler beschreibt.

- Reduktion des Wartens auf Inter-VM-Kommunikation und auf Ein-/Ausgabe, indem Guest und Host OS Scheduler eine Vorabreservierung von IO-Ressourcen ermöglichen (sog. Advanced Reservation): Dabei kann in der JCL eine frühzeitige Reservierung von IO-Ressourcen für den Job erfolgen, z.B. um Rechenergebnisse ohne Verzögerung auf Festplatte schreiben zu können, sobald sie zur Verfügung stehen.

- Erweiterung des Cloud OS Schedulers durch ein Leistungs-Beschleuniger lieferungsmodell (Performcance Delivery Model) für die Cloud und ein Leistungsverbrauchsmodell seiner Kunden zur vorausschauenden Ressourcenplanung: Diese Angebots-/Nachfragemodelle sollten in der Cloud über die Verbindungstopologie, die Server Hardware und die Anwendungssoftware informiert sein, um so bessere Scheduling- Entscheidungen zu ermöglichen.

- Vermeidung des Effekts der „lauten Nachbarn“ durch reinen Stapelbetrieb (Batch Processing), zumindest in einem Teil der Cloud. Es sollte in dieser Cloud-Partition eine exklusive Zuordnung von Cores zu Jobs geben, in der Cores sich nicht im Zeitscheibenverfahren (Time-Sharing) zwischen Benutzern und Jobs aufteilen müssen. Das Time-Sharing sollte auf eine andere Teilmenge von Cores beschränkt bleiben, da es die Inter-VM-Kommunikation stört

- Erhöhen der Problemgrößen auf mindestens 100 000 Gitterpunkte, um das Verhältnis von Rechnen und Datenaustauschen zu verbessern: Dadurch sinkt anteilig der Overhead durch die Kommunikation, weil mehr gerechnet wird. Auch dies ist eine alte Erkenntnis, die aber wichtig ist.

- Erweiterung von OpenStack, so dass das Cloud-Betriebssystem geeignet ist, Infiniband-basierte Cloud-Kommunikation zu konfigurieren

und effizient zu betreiben: Separate Administrationswerkzeuge außerhalb von OpenStack sollten entfallen.

- Optimierung des Guest und des HostOS durch spezifische Infiniband-Gerätetreiber für MPI-basierte Anwendungen: Dies beinhaltet das Bereitstellen von optimierten Schnittstellen für kurze L2 Datenrahmen, aber auch für lange L3 IP-V6-Jumbo-Pakete. Da dies sich widersprechende Forderungen sind, wäre eine Erweiterung des Cloud Schedulers wünschenswert, die jeder VM eine Netzwerkkarte inkl. Gerätetreiber zuweisen kann, die zum Kommunikationsverhalten der VM passt.

Danksagung: Das Projekt wurde vom Simulationswissenschaftlichen Zentrum Clausthal-Göttingen (SWZ) unter der Kennziffer 11.4.1 gefördert.

Literatur

[1] U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR), The Magellan Report on Cloud Computing for Science, December, 2011

[2] http://www.openstack.org/

[3] http://www.openfoam.com/

[4] A. Gupta, Techniques For Efficient High Performance Computing In The Cloud, Dissertation, University of Illinois at Urbana-Champaign, 2014

[5] A. Gupta, L. V. Kale, Towards Efficient Mapping, Scheduling, and Execution of HPC applications, 27th IEEE International Symposium on Parallel & Distributed Processing Workshop, May 20-24, Boston, USA, 2013

[6] A. Gupta, L. V. Kale, D. Milojicic, P. Faraboschi, S. M. Balle, HPC-Aware VM Placement in Infrastructure Clouds, IEEE International Conference on Cloud Engineering, March 25-28, San Francisco, USA, 2013

[7] Stelios Sotiriadis, Nik Bessis, Fatos Xhafa, Nick Antonopoulos, From meta-computing to interoperable infrastructures: A review of meta-schedulers for HPC, grid and cloud, 26th IEEE International Conference on Advanced Information Networking and Applications, March 26-29, Fukuoka, Japan, 2012

[8] N. Bessis, S. Sotiriadis, V. Cristea, F. Pop, Modelling Requirements for Enabling Meta-Scheduling, Third International Conference on Intelligent Networking and Collaborative Systems, 30 Nov - 02 Dec., Fukuoka Institute of Technology, Fukuoka, Japan, 2011

[9] Iman Sadooghi, Sandeep Palur, Ajay Anthony, Isha Kapur, Karthik Belagodu, Pankaj Purandare, Kiran, Ramamurty, Ke Wang, Ioan Raicu, Achieving Efficient Distributed Scheduling with Message Queues in the Cloud for Many-Task Computing and High-Performance Computing, 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 26-29, Chicago, USA, 2014

[10] Han Zhao, Xiaolin Li, Designing Flexible Resource Rental Models for Implementing, HPC-as-a-Service in Cloud, 26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, DOI 10.1109/IPDPSW.2012.324, 21-25 May, Shanghai, China, 2012

[11] Yanyan Hu, Xiang Long, Jiong Zhang, Enhance Virtualized HPC System Based on I/O Behavior Perception and Asymmetric Scheduling, 14th IEEE International Conference on High Performance Computing and Communications, 25-27 Jun, Liverpool, UK, 2012.

[12] R. Ledyayev, H. Richter, High Performance Computing in a Cloud Using OpenStack, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2014, http://www.iaria.org/conferences2014/CLOUDCOMPUTING14.html, Venice, Italy, 6 pages, May 25 - 29, 2014.

[13] http://www.mcs.anl.gov/research/projects/mpi/

[14] http://www.cloudbus.org/cloudsim/

[15] http://www.opensourceforu.com/2015/01/getting-started-greencloud-simulator/

[16] http://www.arcos.inf.uc3m.es/ icancloud/Home.html

[17] S. K. Garg, R. Buyya, NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations, Proc. Fourth IEEE International Conference on Utility and Cloud Computing, 5-8 Dec., Melbourne, Australia, 2011

[18] C. Thiam, G. Da Costa, J.-M. Pierson, Cooperative Scheduling Antiload balancing Algorithm for Cloud : CSAAC, IEEE International Conference on Cloud Computing Technology and Science, 2-5 Dec., Bristol, UK, 2013

[19] H. Richter, A. Keidel, Hochleistungsrechnen und Echtzeit in virtualisierten Maschinen und Clouds - Die Intel Virtualisierungshilfen, in IfI Technical Report Series ISSN 1860-8477, IfI-14-03, http://www.in.tu-clausthal.de/forschung/technical-reports/, editor: Department of Computer Science, Clausthal University of Technology, Germany, 44 pages, 2014.

[20] https://msdn.microsoft.com/en-us/library/windows/hardware/hh440148%28v=vs.85%29.aspx

[21] http://www.intel.de/content/www/de/de/intelligent-systems/intel-technology/vt-directed-I/O-spec.html, 2013, abgerufen am 24.07.2014.

[22] https://tools.ietf.org/html/rfc2784, abgerufen am 22.06.2015.

[23] https://tools.ietf.org/html/rfc7348, abgerufen am 22.06.2015.

[24] http://openvswitch.org/, abgerufen am 19.06.2015

[25] https://www.kernel.org/doc/Documentation/networking/tuntap.txt abgerufen am 23.06.2015

[26] https://www.openstack.org/assets/presentation-media/openvswitch-in-neutron-1.pdf, abgerufen am 23.06.2015

[27] https://www.opennetworking.org/sdn-resources/openflow abgerufen am 23.06.2015

[28] https://www.kernel.org/doc/Documentation/networking/openvswitch.txt abgerufen am 23.06.2015

[29] https://tools.ietf.org/html/rfc4391

[30] http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v3.1.0.1.pdf abgerufen am 23.06.2015

[31] https://www.open-mpi.org/faq/?category=supported-systems abgerufen am 23.06.2015

[32] https://www.ietf.org/proceedings/50/slides/infinib-1/sld005.htm abgerufen am 23.06.2015

[33] https://www.kernel.org/doc/Documentation/infiniband/ipoib.txt

[34] https://www.openfabrics.org/

[35] http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[36] https://standards.ieee.org/findstds/standard/802.1Q-2003.html

[37] A. Tannenbaum, Moderne Betriebssysteme 2009, Pearson Studium, S.632.