i3sched – Ein OpenNebula Scheduler für die Oracle Grid Engine

Alexander Ditter, Steffen Limmer, Dietmar Fey
Universität Erlangen-Nürnberg, Erlangen, Deutschland, {alexander.ditter, steffen.limmer, dietmar.fey}@cs.fau.de

Kurzfassung

Die effiziente und bedarfsgerechte Nutzung von Rechenressourcen ist einer der entscheidenden Vorteile des Cloud-Computing gebenüber dem herkömmlichen Cluster- und Grid-Computing. Abhängig von den Anforderungen einer Anwendung können Nutzer nur die tatsächlich benötigte Rechenleistung abrufen und diese bei Bedarf auch im laufenden Betrieb erweitern oder verringern. Wir führen diesen Ansatz durch den gleichzeitigen Einsatz von Cluster- und Cloud-Computing auf der selben Hardware-Infrastruktur noch einen Schritt weiter. Ermöglicht wird dies durch den von uns für die Cloud-Middleware OpenNebula, sowie die Oracle Grid Engine (vormals Sun Grid Engine) entwickelten Scheduler, *i3sched*, der den Standard-Scheduler von OpenNebula ersetzt. Er ermöglicht dadurch die Integration und den Betrieb von OpenNebula auf einem bestehenden Cluster, ohne die Batch-Verarbeitung des Clusters zu stören. Wir beschreiben den Aufbau und die Funktionsweise von *i3sched*, sowie die Integration in unsere bestehende Infrastruktur.

1 Einleitung

In diesem Papier stellen wir den für die Cloud-Middleware *OpenNebula* (ON) [1] entwickelten Scheduler, *i3sched*, vor. Wir beschreiben die durch seinen Einsatz mögliche Integration mit der *Oracle Grid Engine* (OGE) [2] und die dadurch ermöglichte parallele Nutzung von Cluster- und Cloud-Computing auf der selben Hardware-Infrastuktur. Die dafür notwendigen Anpassungen, sowie Fehlerbehandlungsszenarien, die durch das Zusammenspiel von OpenNebula und der OGE berücksichtigt werden müssen, werden ebenfalls diskutiert.

Im Bereich des Cloud-Computing gibt es bereits eine Vielzahl von Software und Werkzeuten, die einem beim Einrichten, Betreiben und Warten sowohl von privater als auch öffentlicher Cloud-Infrastruktur unterstützen [1, 3]–[5]. Eine dieser sogenannten Cloud-Middleware-Plattformen ist OpenNebula. Es gehört, zusammen mit OpenStack [4], aktuell zu den populärsten Vertretern und wird von uns für den Betrieb einer Cloud-Infrastruktur verwendet.

1.1 Motivation

In der heutigen Zeit werden, neben den Beschaffungskosten, die Betriebskosten von Großrechenanlagen ein immer größerer Kostenfaktor. Die Betreiber sind daher an einer möglichst hohen Auslastung ihrer Systeme interessiert. Häufig ändern sich jedoch die Anforderungen an die eingesetzten Systeme während der Verwendungsdauer. Der Einsatz von *i3sched* bietet hier den entscheidenden Vorteil, dass eine bestehende Hardware-Infrastruktur zum einen weiterhin als Cluster genutzt werden kann, zum anderen aber auch virtualisierte Ressourcen für OpenNebula zur Verfügung stellen kann.

1.2 Hintergrund

Als Teil einer jeden Cloud-Middleware besteht die Aufgabe eines Schedulers darin, die von einem Nutzer oder dem System angeforderten *virtuellen Maschinen* (VMs) auf vorhandene Rechenressourcen zu verteilen. Dabei ist es erforderlich, dass der Scheduler exklusiv über die entsprechenden Ressourcen verfügen kann bzw., dass diese nicht anderweitig ausgelastet werden. Dieser Ansatz ist jedoch nur dann zweckmäßig, wenn eine Hardware-Infrastruktur ausschließlich für die Cloud-Middleware zur Verfügung steht. Nicht selten kommt es vor, dass bereits bestehende Systeme auch für neue Anwendungen verwendet werden müssen, ohne dabei die bisherige Funktionalität zu beeinträchtigen.

OpenNebula ist, wie bereits erwähnt, neben OpenStack die aktuell am weitest verbreitete und sich am schnellsten weiter entwickelnde Cloud-Plattform. Die Hauptfunktionalität einer solchen Cloud-Middleware besteht darin, die vorhandenen Rechenressourcen zu verwalten, virtuelle Maschinen (VMs) zu Starten, zu überwachen und diese auch wieder zu beenden.

Der typische Lebenszyklus einer VM, wie in vereinfachter Form in Bild 1 dargestellt, besteht bei OpenNebula aus acht Phasen. Zuerst wird für die durch den Nutzer bzw. das System angeforderte VM überprüft ob das entsprechende VM-Image in der Datenbank vorhanden und sich im Zustand READY befindet. Ist das der Fall, so wird vom Scheduler ein Host, entweder nach Vorgabe des Nutzers oder nach Verfügbarkeit ausgewählt auf dem die VM zur Ausführung kommt. Bis zu diesem Punkt befindet sich die VM im Status PENDING. Anschließend wird das Image auf den lokalen Speicher des entsprechenden Hosts kopiert um von dort gestartet zu werden. Diese Phase wird als PROLOG bezeichnet und ist abgeschlossen sobald der Hypervisor auf dem Host mit der Ausführung der VM beginnt. Dies wird als BOOT Phase bezeichnet. Während ihrer Ausführung befindet sich die VM im Zustand

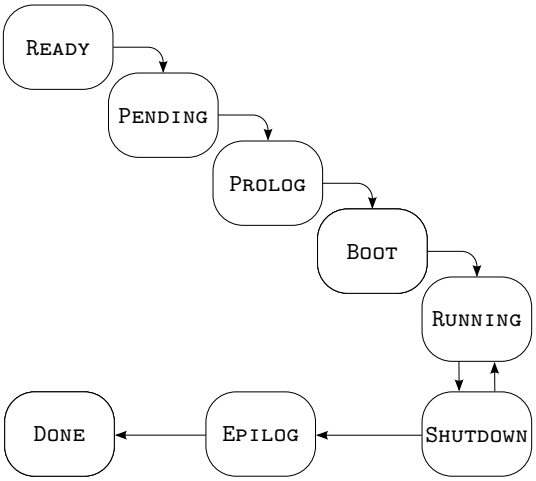

Bild 1. Vereinfachte Darstellung der Zustände einer VM in OpenNebula.

RUNNING; dazu zählen sowohl das interne Hochfahren der Maschine, die Ausführung des Gast-Betriebssystems, sowie das Herunterfahren der VM. Nachdem die VM von OpenNebula mittels ACPI das Signal zum Herunterfahren erhalten hat, ändert sich ihr Status in SHUTDOWN. Dabei ist zu beachten, dass der ON-Scheduler für den Fall, dass die VM ihre Ausführung nicht beendet und sich nach einem gewissen Timeout noch immer im Zustand SHUTDOWN befindet, ihren Status wieder auf RUNNING setzt. War das Herunterfahren hingegen erfolgreich, so wechselt der Status der VM erst in den Zustand EPILOG, der andauert bis alle Daten der VM zurück geschrieben sind und die VM wieder vom Host entfernt ist. Danach wechselt der Zustand der VM zu DONE. In diesem Zustand taucht die VM nicht mehr in der Liste des Schedulers auf, die damit verküpften Daten, wie z. B. die Konfiguration mit der der Hypervisor die VM erzeugt hat oder die Ausführungsdauer, werden z. B. zu Abrechnungszwecken gespeichert und können auch nachträglich noch abgerufen werden.

Die Problematik der gleichzeitigen Nutzung der Rechenressourcen liegt darin begründet, dass auch die OGE „Jobs“ auf die Hosts verteilt. Hierfür verwaltet sie eine Liste der verfügbaren sowie der belegten Ressourcen. Das parallele Betreiben von ON und OGE wäre funktional zwar ohne Probleme möglich, würde jedoch potentiell zur Überauslastung von Hosts führen. In einem solchen Fall würden ON und die OGE vermeintlich einen Host exklusiv nutzen, während tatsächlich nur jeweils 50% der Rechenleistung bzw. Rechenzeit zur Verfügung stehen.

Für die gemeinsame Nutzung, wie in Bild 2 angedeutet, ist es daher notwendig auch eine gemeinsame Liste für die Verwaltung der Ressourcen zu nutzen. In unserem Ansatz erreichen wir dies indem wir der OGE die Verwaltung der gesamten Ressource überlassen und der ON-Scheduler durch den von uns entwickelten *i3sched* ersetzt wird. Dieser gibt seine Ressourcen-Anforderungen direkt an die OGE weiter. Somit ist sichergestellt, dass es nicht zur Überauslastung einzelner Hosts kommen kann und die OGE zu jedem Zeitpunkt den Überblick über die

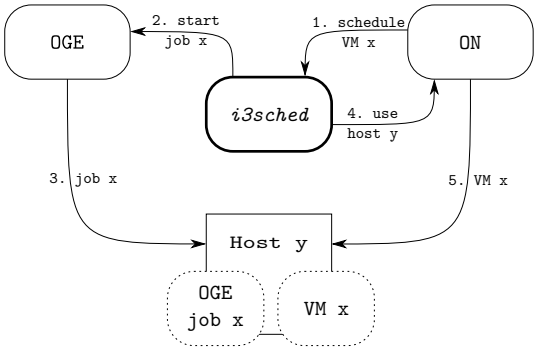

Bild 2. Funktionsweise von *i3sched*.

Auslastung des Gesamtsystems hat.

1.3 Vorarbeiten

Es gibt bereits Lösungen, die einen ähnlichen Ansatz verfolgen. In der Cloud-Plattform Nimbus [5] gibt es z. B. den sogenannten „workspace pilot“, der es ermöglicht Nimbus in ein bestehendes *Portable Batch System* (PBS) Cluster zu integrieren. Hierfür werden sogenannte „pilot jobs“ verwendet, die ebenfalls im Batch-System als Platzhalter für die virtuellen Maschinen fungieren.

Weitere alternative Scheduler für ON sind z. B. *Haizea* [7], der *Green Cloud Scheduler* (GCS) [8] oder das *Energy-aware Multi-start Local Start Metaheuristic for Scheduling VMs in the OpenNebula Cloud* (EMLS-ONC) System [9]. Während Haizea offenbar nicht mehr weiter entwickelt wird, sind von GCS und EMLS-ONC noch Versionen aus dem Jahr 2012 verfügbar. Allerdings ist auch hier, angesichts der rasanten Entwicklung von OpenNebula davon auszugehen, dass keine aktive Weiterentwicklung mehr stattfindet.

2 Architektur

Unser Scheduler ist vollständig in Python (Version 2.7) implementiert. Bild 3 zeigt den Aufbau von *i3sched* anhand eines Klassendiagramms.

Die zentrale Klasse, `Scheduler`, stellt die Hauptfunktionalität bereit – insbesondere die Schleife für den kontinuierlichen Ablauf von *i3sched*. Weiterhin sind vier Hilfsklassen vorhanden: `Log`, `OGE`, `OneRPC` sowie `Config`. Die Klasse `Log` stellt die Logging-Funktionalität zur Verfügung. Die mitprotokollierten Ereignisse dienen hauptsächlich dem Debugging um z. B. feststellen zu können auf welchem Host eine VM ausgeführt werden soll. Die Klasse `OGE` stellt die Schnittstelle zur Oracle Grid Engine bereit. Die Anbindung erfolgt durch die Kommandozeilen Tools *`qsub`*, *`qstat`*, *`qdel`* und *`qhost`* und ist mit der OGE Version 6.2u5 kompatibel. Die Klasse `OneRPC` im `client`-Modul enthält die Schnittstelle zu OpenNebula; die Kommunikation erfolgt über die XML-RPC API und ist aktuell kompatibel bis zur OpenNebula Version 4.0. Die Klasse `Config` stellt Methoden zur Verwendung von Konfigurationsdateien zur Verfügung. Dafür wird das Modul `configobj` genutzt, welches von Michael Foord und Nicola Larosa [10] entwickelt wurde und als Open

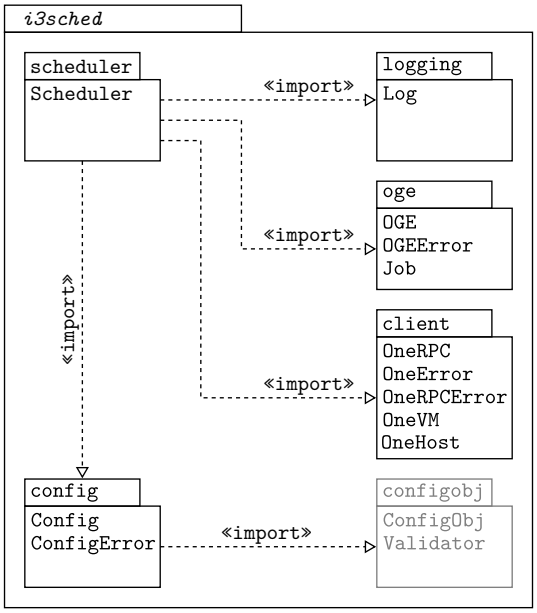

Bild 3. *i3sched* Klassendiagramm.

Source unter der Berkeley Software Distributed (BSD) Lizenz verfügbar ist.

3 Funktionsweise

Beim Start von *i3sched* wird als erstes ein Objekt der Klasse `Scheduler` initialisiert, welches wiederum je ein Objekt der Klassen `Log`, `OGE`, `OneRPC` sowie `Config` erzeugt. Anschließend werden die Informationen der Konfiguration gelesen und die Hauptschleife des Schedulers gestartet. Bild 4 zeigt die Aktivitäten eines Durchlaufs der Hauptschleife.

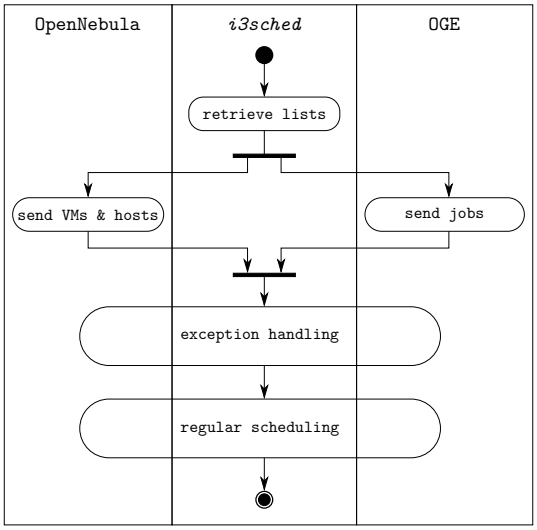

Bild 4. Ablaufdiagramm von *i3sched*.

Am Anfang jeder Iteration aktualisiert *i3sched* seine Informationen, d. h. er ruft die Liste aller VMs und Hosts von OpenNebula, sowie die aller Jobs von der OGE ab. Daran anschließend wird die Fehlerbehandlungsphase durchlaufen, in dieser wird nach Inkonsistenzen gesucht und diese, falls vorhanden, aufgelöst. Danach folgt die Scheduling Phase in der zum starten bereite VMs auf den verfügbaren Hosts, die über ausreichend freie Kapazitäten verfügen, eingeplant und gestartet werden.

Bild 5 zeigt den Ablauf des Einlastens einer VM durch *i3sched*. Als erstes wird für jede VM die sich im Zustand PENDING befindet überprüft, ob ein entsprechender Job in der OGE vorhanden ist und falls ja, auf welchem Host dieser platziert wurde. Anschließend wird OpenNebula angewiesen die entsprechende VM auf diesem Host zu erstellen.

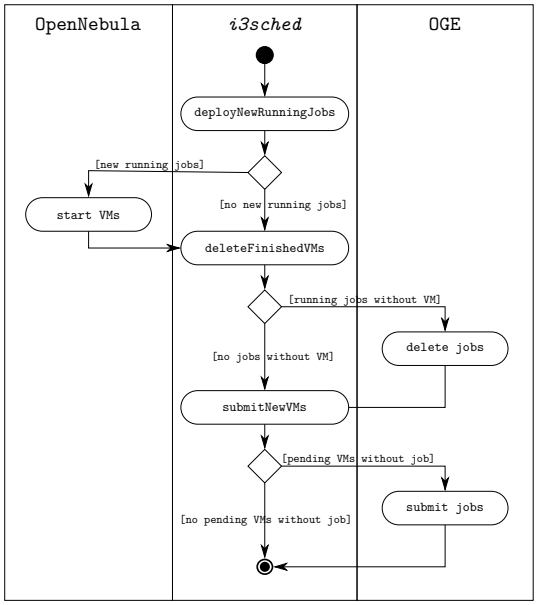

Bild 5. Einlasten einer VM durch *i3sched*.

Danach wird für jede beendete VM überprüft, welchem OGE-Job sie zugeordnet ist. Dies ist vorallem dann relevant, wenn die entsprechende VM seit dem letzten Durchlauf durch OpenNebula beendet wurde. In diesem Fall wird der entsprechende OGE-Job entfernt, damit die entsprechenden Ressourcen auf dem Host wieder freigegeben werden und für neue Jobs zur Verfügung stehen.

Als letztes werden alle sich im Zustand PENDING befindlichen VMs identifiziert, für die noch kein OGE-Job vorhanden ist. Dieser wird dann entsprechend der Anforderungen der VM durch die OGE erstellt. Diese OGE-Jobs sind „Dummy-Jobs“, d. h. in ihnen wird außer einem „Sleep“, nichts getan. Dies kann dazu führen, dass eine VM mit fehlerhafter Konfiguration (z. B. zu viele CPUs) nie gestartet wird, da von der OGE kein entsprechender Job eingelastet und an *i3sched* weitergegeben werden kann. Weiterhin werden von der OGE nur Hosts für ein Einlastung von „Dummy-Jobs“ zugelassen die auch von OpenNebula überwacht werden.

Bild 6 zeigt den Ablauf der Ausnahmebehandlung. Zuerst werden in dieser alle laufenden VMs die der OGE nicht bekannt sind gesucht. Dafür ist es notwendig die gesamte Liste der OpenNebula bekannten VMs zu durchlaufen und diese mit den laufenden Jobs der OGE abzugleichen. Ob ein Job zu einer laufenden VM gehört

oder nicht, kann anhand der bei OpenNebula vermerkten ID sowie dem Namen des OGE-Jobs festgestellt werden, da der Name des OGE-Jobs die entsprechende ID enthält. Falls die Funktion `shutdownRunningZombies` tatsächlich eine VM ohne entsprechenden Eintrag in der OGE finden sollte, so wird diese durch den entsprechenden XML-RPC Aufruf durch OpenNebula beendet. Dieses Vorgehen ist von essentieller Bedeutung für die reibungslose Kooperation von OpenNebula und der OGE.

Im zweiten Schritt der Fehlerbehandlung wird überprüft, dass sich keine VM länger als eine bestimmte Zeit im Zustand SHUTDOWN befindet. Dieses Vorgehen ist notwendig, da VMs vereinzelt nicht vollständig von ON beendet werden können und in diesem Fall als „Leichen“ sowohl in OpenNebula als auch in der OGE verbleiben würden. Dies könnte mit steigender Betriebsdauer dazu führen, dass Hosts mit VMs belegt erscheinen die eigentlich schon längst beendet sind und damit das gesamte System zu Stillstand bringen. Wenn *i3sched* eine VM findet, die sich bereits länger als das vorgegebene Zeitlimit im entsprechenden Status befindet, so wird diese direkt auf dem Host durch den Hypervisor (z. B. bei XEN mit `xm destroy`) beendet. Nach dem erfolgreichen Beenden werden OpenNebula und die OGE über den neuen Zustand informiert.

Im letzten Schritt der Fehlerbehandlung werden alle sich im Zustand UNKNOWN befindlichen VMs beendet, da diese nicht mehr von OpenNebula angesprochen werden und daher auch nicht mehr durch OpenNebula beendet werden können. Auch diese VMs werden direkt auf dem entsprechenden Host durch den Hypervisor (via SSH) beendet.

Nach Durchlauf der Fehlerbehandlung sind alle Inkonsistenzen aufgelöst und das normale Scheduling wird durchlaufen. Wie man ebenfalls in Bild 6 sieht, ist die OGE an der Fehlerbehandlung nicht beteiligt, da sich diese nur auf die VMs bezieht.

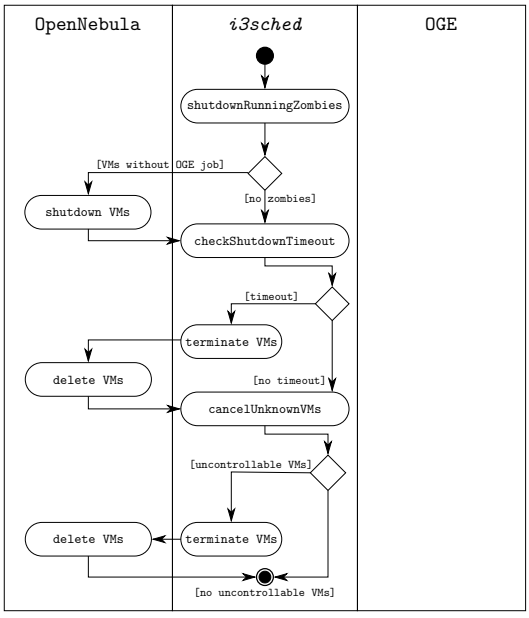

Bild 6. Fehlerbehandlung in *i3sched*.

4 Zusammenfassung und Ausblick

Mit *i3sched* haben wir eine einfache, aber wichtige Erweiterung zum Standard-Scheduler von OpenNebula entwickelt. Der Einsatz ist aktuell nur im Zusammenspiel mit der OGE möglich, allerdings ist eine Erweiterung für andere Batch-Scheduling-Systeme, wie z. B. Slurm [11], geplant, de diese in Zukunft auch in unserem Cluster zum Einsatz kommen sollen. Eine Erweiterung für weitere Cloud-Middleware, wie z. B. OpenStack, ist aktuell nicht geplant. Zum einen, weil OpenNebula mittelfristig bei uns weiter im Einsatz bleiben wird und zum anderen die Machbarkeit einer Erweiterung für andere Cloud-Middleware maßgeblich von deren Architektur abhängig ist.

5 Danksagung

Diese Arbeit entstand im Rahmen des vom Bundesministeriums für Wirtschaft und Technologie (BMWi), durch Beschluss des deutschen Bundestags, geförderten Projekts Cloud4E.

Literatur

[1] OpenNebula Webseite: *http://opennebula.org/* (Abgerufen am 20. September 2013).

[2] Oracle Grid Engine Webseite: *http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html* (Abgerufen am 20.09.2013).

[3] Eucalyptus Webseite: *http://www.eucalyptus.com/* (Abgerufen am 20. September 2013).

[4] OpenStack Webseite: *http://www.openstack.org/* (Abgerufen am 20. September 2013).

[5] Nimbus Webseite: *http://www.nimbusproject.org/*, (Abgerufen am 20. September 2013).

[6] T. Freeman and K. Keahey, *Flying Low: Simple Leases with Workspace Pilot*, Euro-Par 2008 – Parallel Processing, LNCS, vol. 5168, pp. 499–509, Springer Berlin Heidelberg, 2008.

[7] B. Sotomayor, K. Keahey, and I. Foster, *Combining batch execution and leasing using virtual machines*, Proceedings of the 17th international symposium on High performance distributed computing, pp. 87–96, ACM, New York, NY, USA, 2008.

[8] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and A. Kipp, *Energy Aware Dynamic Resource Consolidation Algorithm for Virtualized Service Centers Based on Reinforcement Learning*, 10th International Symposium on Parallel and Distributed Computing (ISPDC), 2011, pp.163–169, 2011.

[9] Y. Kessaci, N. Melab, and E. Talbi, *An energy-aware multi-start local search heuristic for scheduling VMs on the OpenNebula cloud distribution*, International Conference on High Performance Computing and Simulation (HPCS), 2012, pp. 112–118, 2012.

[10] M. Foord and N. Larosa, *Reading and Writing Config Files – ConfigObj 4 Introduction and Reference*, verfügbar unter: *http://www.voidspace.org.uk/python/configobj.html*, (Abgerufen am 20. September 2013).

[11] SLURM Webseite: *http://slurm.schedmd.com/slurm.html*, (Abgerufen am 20. September 2013).