Abstract

P2PSim – Ein Framework zur Simulation der Heterogenität und Volatilität von Ressourcen in Peer-to-Peer Desktop Grids

Das folgende Dokument beschreibt die Entwicklung eines Frameworks, mit dessen Hilfe die Heterogenität und die Volatilität von Ressourcen in Peer-to-Peer Desktop Grids simuliert werden kann. Da diese Simulation auf einem Rechencluster stattfinden soll, auf dem die Ressourcen (die Rechenknoten) weder Heterogenität noch Volatilität aufweisen, wird eine Möglichkeit entwickelt, diese künstlich herzustellen.

Die auf dem Cluster installierte Software „Virtual Box“ ermöglicht es, jedem Rechenknoten eine virtuelle Gastmaschine zuzuordnen. Diese virtuellen Maschinen sind individuell konfigurierbar hinsichtlich der Heterogenität (Geschwindigkeit des Prozessors und Größe des Arbeitsspeichers). Darüber hinaus lassen sich die virtuellen Maschinen einzeln jederzeit starten und wieder stoppen, wodurch sich eine beliebige Volatilität herstellen lässt.

Um für die Simulation eine realistische Verteilung von verschiedenen Prozessorgeschwindigkeiten und Arbeitsspeichergrößen zu erreichen, wird für die Generierung dieser Werte auf eine umfassende Sammlung von Systemaufzeichnungen aus parallelen und verteilten Systemen zurückgegriffen. Hierbei handelt es sich um das Failure Trace Archive, aus dem sich nach Analyse aller zugrunde liegenden Daten die Datenbank des SETI@home - Projektes (Verteiltes Rechnen) als verwendbar erwiesen hat. Für die Konfiguration der virtuellen Maschinen liegen aus der genannten Datenbank mehrere 10.000 Datensätze vor.

In dieser Arbeit wird die vollständige Implementierung der Lösung beschrieben um sie für Anwender und Interessierte nachvollziehbar zu machen. Neben der Analyse und Aufbereitung der SETI-Datenbank wird die gesamte Systemarchitektur, Funktionsweise und das Zusammenspiel der unterschiedlichen Komponenten beschrieben. Darüber hinaus werden im Detail einige wichtige Vorgehensweisen beleuchtet, wie Programmaktionen mit Hilfe der Linux-Shell (bash) ausgeführt werden.

Anschließend wird dem Anwender eine Installations- und Konfigurationsanleitung für eine virtuelle Maschine auf einem Rechencluster gegeben, die im Folgenden als Vorlage für alle weiteren automatisch generierten Instanzen von virtuellen Maschinen dient, die an der Simulation teilnehmen.

Zum Schluss wird die Verwendung von P2PSim im Detail beschrieben.

Inhaltsverzeichnis

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DFPOP	Floating-Point-Operations per Second with Double Precision
FTA	Failure Trace Archive
GPL	Gnu Public License
GUI	Graphical User Interface (Grafische Benutzeroberfläche)
IPv4	Internet Protocol Version 4 (definiert in RFC 791, 1981)
IPv6	Internet Protocol Version 6 (Nachfolgeprotokoll zu IPV4, welches dieses in den nächsten Jahren ablösen soll, bzw. muss)
MMOG	Massive Multiplayer Online Game, ein Online-Spiel mit einer sehr großen Anzahl an Teilnehmern
P2P	Peer to Peer (Verbindungsart in Rechnernetzen, wobei Rechner direct mit anderen Rechnern kommunizieren)
SCP	Secure Copy, ein Protokoll und ein Programm zur verschlüsselten Dateiübertragung
SETI	Search for extraterrestrial intelligence, ein Verteiltes-Rechnen-Projekt derUniversität Berkeley
TCP	Transmission Control Protocol (verbindungsorientiertes Netzwerkprotokoll)
TRM	Torque Resource Manager
UDP	User Datagram Protocol (verbindungsloses Netzwerkprotokoll)
UI	User Interface (Benutzerschnittstelle)
UUID	Universally Unique Identifier (eindeutige Kennzeichnung von Ressourcen)
XML	Extensible Markup Language, eine Auszeichnungssprache zur Darstellung von hierarchischen Datenstrukturen in Textdateien

1. Einleitung

1.1 Aufgabenstellung

Peer-to-Peer Desktop Grids vereinen die ungenutzten Ressourcen verteilter Rechner (Desktop oder Cluster) durch einen losen Zusammenschluss dieser zu einem Parallelrechner von enormer Leistungsfähigkeit.

Aufgrund der Vielzahl von administrativen Domänen, denen die teilnehmenden Rechner unterliegen, ergeben sich gegenüber dem Supercomputing mit Clusterarchitektur zusätzliche Anforderungen. So haben die Knoten im Allgemeinen keine einheitliche Architektur und die zur Verfügung stehenden Ressourcen können spontan vom Besitzer entzogen werden. Aus diesem Grund müssen Systeme und Anwendungen tolerant gegenüber Heterogenität und Volatilität der Ressourcen sein.

Die Entwicklung und Evaluation von Anwendungen auf realen Peer-to-Peer Desktop Grids ist jedoch meist nicht möglich, da hier keine reproduzierbaren Bedingungen herrschen und Experimente mit verschiedenen Parametern somit nicht vergleichbar sind. Deswegen werden Testläufen meist auf Rechenclustern durchgeführt, welche weder Heterogenität noch Volatilität der Ressourcen zeigen. Um dennoch gezielt die Robustheit der Anwendungen im Hinblick auf diese speziellen Eigenschaften von Peer-to-Peer Desktop Grids zu untersuchen, müssen diese simuliert werden.

In dieser Studien- oder Bachelorarbeit soll ein Framework entstehen, welches den Entwicklungsprozess sowie die Evaluation solcher Simulationsabläufe unterstützt. Es soll die Möglichkeit bieten, Simulationsszenarien mit definierter Heterogenität und Volatilität zu erstellen, die Simulation zu kontrollieren und die Ergebnisse gesammelt zur Verfügung zu stellen.

Hierzu muss zunächst die Möglichkeit geschaffen werden, die Konfiguration der Rechenknoten, Anwendungen sowie die Zuordnung von Anwendungen zu Rechenknoten zu definieren. Die Knotenkonfigurationen sollen sowohl vom Anwender vorgegeben als auch automatisch generiert werden können, wobei im letzteren Fall das Failure Trace Archive (FTA, http://www.fta.inria.fr) als Datenbasis dienen soll. Diese Sammlung speichert Konfigurationen und Verfügbarkeitsverläufe von Rechnern aus realen Peer-to-Peer Desktop Grids in einem maschinenlesbaren Format.

Aus den erstellten Konfigurationen sollen dann Steuerinformationen für den Torque Resource Manager und dessen Queuing-System auf der vorhandenen Cluster-Infrastruktur erzeugt werden. Die Integration von virtuellen Maschinen (Virtual Box) bietet hierbei die Möglichkeit, die Leistung der CPU, die Größe des Arbeitsspeichers sowie die Verfügbarkeit der Gastsysteme zu bestimmen und damit Heterogenität und Volatilität zu simulieren. Die erarbeitete Lösung soll in Java implementiert und insbesondere auf die Integration der Cohesion DesktopGrid Computing Platform (http://www.cohesion.de) abgestimmt werden.

1.2 Stand der Technik

Nachfolgend werden einige der derzeit aktuellen Lösungen für die Simulation von Peer-to-Peer Systemen vorgestellt. Jede dieser Lösungen setzt andere Prioritäten und es sind individuelle Stärken und Schwächen vorhanden. Allen diesen Lösungen gemeinsam ist, dass es sich hierbei um äußerst umfangreiche Projekte handelt, die für das in dieser Arbeit angestrebte Ziel deutlich überdimensioniert sind. So kommt ein Benutzer bei diesen Projekten um eine aufwendige Einarbeitung im Allgemeinen nicht herum.

1.2.1 OverSim

Die Forschungsgruppe von Prof. Zitterbart am Karlsruher Institut für Technologier (KIT) hat ein Open Source Framework für die OMNet++ Simulation Environment (http://www.omnetpp.org/) entwickelt, welches die Simulation von Overlay-Netzwerken und Peer-to-Peer-Netzwerken ermöglicht. Verschiedene Netzwerkprotokolle werden unterstützt, so z.B. Chord, Pastry, Bamboo, Koorde, Broose, Kademlia, GIA, NICE, Ntree, Quon, Vast und Publish-Subscribe für MMOGs.

Das Hauptaugenmerk dieses Frameworks liegt auf der detaillgetreuen Simulation der Netzwerkprotokolle. Es werden sowohl IPV4 als auch IPv6 unterstützt, sowie TCP und UDP. Die Topologie der Netzwerke kann individuell konfiguriert werden, d.h. dass z.B. Bandbreiten, Latenzzeiten und Paketverluste angepasst werden können.

Projekthomepage: http://www.oversim.org/

1.2.2 PeerSim

Mark Jelasity, Alberto Montresor, Gian Paolo Jesi und Spyros Voulgaris haben mit der unter der GPL Open Source Lizenz stehenden Software PeerSim eine äußerst umfangreiche Java-Anwendung entwickelt, mit dessen Hilfe insbesondere die Volatilität in extrem großen Peer-to-Peer-Systemen simuliert werden kann. Skalierbarkeit und Performanz der Anwendung stehen bei diesem Projekt im Vordergrund, so dass mit Netzwerken gearbeitet werden kann, die mehrere Millionen Knoten beinhalten. Dies impliziert jedoch, dass die Realitätstreue vernachlässigt wird, da verschiedene Vereinfachungen bei der Modellierung getroffen werden.

Der Benutzer der Anwendung kann entscheiden, ob die Simulation cycle-based oder event-based ausgeführt werden soll. Cycle-based bedeutet, dass die Knoten nicht nebenläufig simuliert werden, sondern dass diese in einer vorgegebenen Reihenfolge Rechenzeit zugewiesen bekommen. Dies muss bei der Analyse der Ergebnisse berücksichtigt werden. Mitunter sind manche Problemstellungen auf diesem Wege nicht zu simulieren. Details des Transportprotokolls können bei dieser Art der Simulation ebenfalls nicht berücksichtigt werden. Event-based bedeutet, dass die simulierten Knoten nicht periodisch Rechenzeit zugewiesen bekommen, sondern nur auf explizites Anrufen eines Knotens durch einen anderen. Hierbei wird der Nachrichtenaustausch detaillierter modelliert als bei der cycle-based Simulation. Dies eröffnet die Möglichkeit durch die Simulation Probleme zu erkennen, die mit dem dem cycle-based Ansatz nicht sichtbar sind. Ein Nachteil jedoch ist, dass die Skalierbarkeit abnimmt, da der Rechenaufwand für die event-based Simulation deutlich größer ist.

Projekthomepage: http://peersim.sourceforge.net/

1.2.3 PeerfactSim.KOM

Das Projekt PeerfactSim.KOM ist ein Projekt des Multimedia Communications Lab (KOM) der Technischen Universität Darmstadt. Hierbei handelt es sich um eine Java-Anwendung, mit welcher sehr großen Peer-to-Peer-Systeme simuliert werden können.

In Anlehnung an das ISO/OSI-Modell besteht die Anwendung im Prinzip aus mehreren Schichten, die simuliert werden und untereinander verbunden werden können (pluggable layers). Solch ein modularer Aufbau der Software erleichtert die Implementierung neuer Komponenten.

Eine zentrale Rolle in der Architektur spielt der Simulationskern, welcher die Möglichkeit bietet unterschiedliche Aktionen für das Simulationsszenario zu definieren und zu konfigurieren. Darüber hinaus beinhaltet der Kern die benötigten Algorithmen für das Scheduling der Events.

Bereits von Haus aus bringt PeerfactSim.KOM die Unterstützung einer Vielzahl von P2P-Protokollen mit, so z.B. Gnutella, CAN, Chord und Kademlia.

Abgeschlossene Simulationen können auf unterschiedliche Art und Weise visualisiert werden, z.B. mit Hilfe von gnuplot.

Projekthomepage: http://peerfact.kom.e-technik.tu-darmstadt.de/

2. Theoretischer Lösungsansatz

2.1 Ziel der Studienarbeit

Mit dieser Studienarbeit soll ein Framework geschaffen werden, welches eine realitätsgetreue Simulation von Heterogenität und Volatilität verteilter Systeme ermöglicht und zugleich leichtgewichtig und einfach zu verwenden ist.

Skalierbarkeit und Performanz sind von sekundärem Interesse, da die Anzahl der zu simulierenden Knoten maximal im niedrigen dreistelligen Bereich liegen soll.

Die Implementierung soll in Java stattfinden, wodurch die Anwendung portabel ist und auf unterschiedlichen Clientplattformen lauffähig ist.

2.2 Verwendete Ressourcen

Zur Verfügung steht ein Rechencluster (IPVS, Uni Stuttgart) mit AMD Opteron 254 – Rechenknoten (1 GHz Takfrequenz, 1 GB Cache). Bei den Prozessoren handelt es sich um Dual-Core-CPUs, welche folgende Eigenschaften aufweisen:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 37
model name      : AMD Opteron(tm) Processor 254
stepping        : 1
cpu MHz         : 1000.000
cache size      : 1024 KB
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
```

flags	: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr_opt lm 3dnowext 3dnow pni lahf_lm
bogomips	: 1989.13
TLB size	: 1024 4K pages
clflush size	: 64
cache_alignment	: 64
address sizes	: 40 bits physical, 48 bits virtual
power management	: ts fid vid ttp

Gesteuert werden die Knoten über ein Queueing-System, dem Torque Resource Manager (weitere Informationen hierzu unter http://www.adaptivecomputing.com/products/open-source/torque/).

Auf dem Rechencluster läuft ein Linux als Betriebssystem. Sowohl das Frontend als auch die einzelnen Rechenknoten sind per SSH zu erreichen.

Die Rechenknoten sind vollständig homogen, d.h. jeder Knoten hat exakt die gleiche Taktfrequenz und die gleiche Prozessorarchitektur.

Ebenso weisen die Rechenknoten im Allgemeinen keine Volatilität auf. Es ist zwar durchaus möglich, dass Rechenknoten irgendwann einmal ausfallen, jedoch ist die Wahrscheinlichkeit hierfür sehr gering. Für den Zeitraum einer laufenden Simulation ist daher nicht damit zu rechnen, dass Rechenknoten währenddessen ausfallen.

2.3 Beschreibung der vorgesehenen Lösung

Bei P2PSim handelt es sich um eine Java-Anwendung, die lokal auf einem Linux-Client ausgeführt wird.

Der Benutzer erhält eine einfach zu bedienende grafische Schnittstelle, über die verschiedene Einstellungen der geplanten Simulation vorgenommen werden können. Folgende Einstellungen können gemacht werden:

- Anzahl der zu simulierenden Maschinen
- Minimalwert und Maximalwert der Prozessorgeschwindigkeit
- Minimalwert und Maximalwert der Arbeitsspeichergröße
- Ursprung der Daten für Prozessor und Speicher (per Zufall oder aus dem FTA)
- Grad der Volatilität

Der Benutzer hat somit die Möglichkeit Einstellungen zur Heterogenität und zu Volatilität einer anstehenden Simulation vorzunehmen. Diese Einstellungen umfassen für die Homogenität die Größe des Arbeitsspeichers und die Prozessorgeschwindigkeit von teilnehmenden Peers. Der Grad der Volatilität ist die prozentuale Wahrscheinlichkeit für den Ausfall eines jeden Peers innerhalb von 24 Stunden.

Als Alternative zur manuellen Konfiguration der Heterogenität hat der Benutzer die Möglichkeit anzugeben, dass die zugrunde liegenden Werte aus einer Datenbank aus dem Failure Trace Archive (FTA) ermittelt werden sollen.

Darüber hinaus gibt der Benutzer noch die Anzahl der teilnehmenden Peers ein.

Ist dies erfolgt, sind alle Vorbereitungen für die Simulation getroffen. Die Einstellungen können jederzeit persistent gemacht werden, wodurch ein Laden und Speichern der Programmeinstellungen ermöglicht wird.

P2PSim generiert im weiteren Verlauf verschiedene Skripte und Konfigurationsdateien, welche auf das Frontend des Clusters hochgeladen werden.

Auf den verwendeten Rechenknoten des Clusters ist die Software „VirtualBox“ installiert. Mit dessen Hilfe werden Instanzen von virtuellen Maschinen erstellt, die individuell konfiguriert werden. So entsteht ein Satz von Maschinen, die die an der Simulation teilnehmenden Peers darstellen. Es werden die Taktfrequenz des Prozessors sowie die Größe des Arbeitsspeichers der Gastsysteme angepasst. Auf diese Weise können heterogene virtuelle Maschinen auf homogenen physikalischen Rechenknoten erstellt werden. Über das Queuing System werden die virtuellen Maschinen gestartet und diese beginnen nach dem Hochfahren automatisch mit ihrer Arbeit (der Bearbeitung einer Aufgabe).

Die zu bearbeitende Aufgabe (ein Bash-Skript, welches selbst die zu bearbeitende Aufgabe ist oder weitere Programme startet) legt der Benutzer fest. Sie wird ebenfalls vor Beginn der Simulation auf das Frontend des Clusters hochgeladen.

Entsprechend des zuvor konfigurierten gewünschten Grades der Volatilität fahren die virtuellen Maschinen während der Simulation immer wieder herunter und kurze Zeit später wieder hoch. Es existiert also eine Start-Stopp-Logik, welche die Maschinen zu diskreten Zeitpunkten stoppt bzw. wieder startet.

Sobald alle virtuellen Maschinen ihre Arbeit beendet haben, stehen die Simulationsergebnisse in einem definierten Verzeichnis auf dem Frontend des Clusters zur Verfügung und können im Anschluss analysiert werden.

Die Instanzen der virtuellen Maschinen beenden sich automatisch nach erfolgreichem Abschluss ihrer zu bearbeitenden Aufgabe. Sind alle Maschinen heruntergefahren ist dies in der P2PSim-Anwendung sichtbar und die Simulation ist beendet.

3. Implementierung

3.1 Verwendete Technologien

- Java (Implementierung des Hauptprogramms)
- mySQL (die Daten aus dem FTA liegen im Original in Form einer mySQL-Datenbank vor)
- SQLite (um die Datenbank standalone, also ohne Server, von der Java-Anwendung aus zugreifbar zu machen)
- Bash (es werden verschiedene #bin/bash-Skripte verwedet)
- Linux-Programme (cp, sed, ssh, scp, bc, expr, sleep, grep, test)
- VirtualBox (auf den einzelnen Nodes des Clusters läuft jeweils eine Instanz einer virtuellen Maschine)
- Debian (Linux-Distribution, welche als Betriebssystem für die virtuellen Maschinen sowie als Betriebssystem für den Deployment-Server verwendet wird)
- Torque Resource Manager (ein Queuing-System)
- SSH
- SCP
- Microsoft Excel (Erstellung von Histogrammen)

3.2 Daten vom FTA (Sichtung und Aufbereitung)

Das Failure Trace Archive (http://fta.inria.fr/apache2-default/pmwiki/index.php) beinhaltet eine zentrale Datenbank für Aufzeichnungen aus parallelen und verteilten Systemen. Es ist frei zugänglich und ist insbesondere für den Zweck bestimmt, für Entwickler von Algorithmen und Modellen von Peer-to-Peer-Systemen zu Analysezwecken zur Verfügung zu stehen.

3.2.1 Aufbereitung der Daten

Folgende Datensätze stehen derzeit im FTA zur Verfügung:

System	Type	# of Nodes	Target Component	Period	Year
SETI@home	Desktop Grid	226,208	CPU	1.5 years	2007-2009
Overnet	P2P	3,000	host	2 weeks	2003
Microsoft	Desktop	51,663	host	35 days	1999
LANL	SMP, HPC Clusters	4750	host	9 years	1996-2005
HPC2	HPC Clusters	256	IO	2.5 years	1996-2005
HPC4	Supercomputers	152516	every thing	~1 year	2004-2006
PNNL	HPC Cluster	980	host, network	4 years	2003-2007
NERSC	HPC Clusters	NA	IO	5 years	2001-2006
Skype	P2P	4000	host	1 month	2005
Web sites	Web servers	129	host	8 months	2001-2002
DNS	DNS servers	62,201	host	2 weeks	2004
PlanetLab	P2P	200-400	host	1.5 year	2004-2005
Grenouille03	DSL	4800	host	1 year	2003
Grenouille05	DSL	4800	host	1 year	2005
EGEE	Grid	2500 queues	CE queue	1 month	2007

Grid'5000	Grid	1288	host	1.5 years	2005-2006
Notre Dame	Desktop Grid	700	CPU, host	6 months	2007
ucb94	Desktop Grid	85	CPU	46 days	1994
sdsc03	Desktop Grid	275	CPU	1 month	2003
lri05	Desktop Grid	40	CPU	1 month	2005
deug05	Desktop Grid	40	CPU	1 month	2005
cae06	Grid	686	CPU	35 days	2006
cs06	Grid	725	CPU	35 days	2006
glow06	Grid	715	CPU	33 days	2006
teragrid06	Grid	1001	host	8 months	2006-2007

Tabelle 1: Datensätze Failure Trace Archive

Die Mehrzahl der Datensätze stehen als mySQL-Datenbanken zum Download bereit, was eine Analyse der Daten im Vergleich zum Raw-Format und zum Tabbed-Format deutlich vereinfacht. Die Datenbanken bestehen aus einer Vielzahl von Tabellen, welche wiederum unterschiedlichste Daten enthalten. Wichtig für diese Studienarbeit sind hieraus folgende Daten:

- node_id (int11) = Eindeutige Identifikationsnummer eines Knotens
- mem_size (double) = Größe des Arbeitsspeichers des jeweiligen Knotens in MB
- dfpop_speed (double) = Geschwindigkeit des Prozessors des jeweiligen Knotens in Double Precision Floating Point Operations per Second
- num_procs (double) = Anzahl Prozessoren des jeweiligen Knotens

Die Sichtung der 13 GB großen mySQL-Datenbank (mittels einer lokal installierten XAMPP-Instanz) ergab, dass einzig das Archiv „SETI@Home“ (http://setiathome.berkeley.edu/) genügen viele vollständige Datensätze bietet, die alle vier genannten Einträge enthalten. Die Daten stammen aus den Jahren 2007 bis 2009 und sind damit im Vergleich zu den anderen Daten des FTA die aktuellsten. Somit werden sie im weiteren Verlauf dieser Studienarbeit verwendet.

Um die Größe der Datenbank zu verringern und um die Übersichtlichkeit zu erhöhen, wurde eine neue Tabelle erstellt, welche ausschließlich die oben genannten vier Spalten enthält. Folgender SQL-Befehl war hierfür notwendig:

```
INSERT INTO node

SELECT node_.node_id, node_perf.dfpop_speed, node_.mem_size, node_.num_procs

FROM node_, node_perf

WHERE node_.node_id = node_perf.node_id

ORDER BY node_id
```

Im Anschluss daran wurden ungültige Datensätze entfernt (z.B. bei fehlenden Werten, bei Werten gleich Null oder bei Werten gleich dem Maximums des Double-Bereichs). Darüber hinaus wurden Werte entfernt, die für die Simulation im weiteren Verlauf nicht praktikabel verwendet werden können. So ist eine Mindestgeschwindigkeit des Prozessors und eine Mindestgröße des Arbeitsspeichers notwendig, damit die Instanz der virtuellen Maschine betrieben werden können. Die entstandene Tabelle enthält nach den Bearbeitungen nun noch 219719 Datensätze, ursprünglich waren es 226208.

Im Folgenden wird die erstellte Datenbank von mySQL nach SQLite konvertiert, da ein Zugriff auf die Daten ohne einen Server stattfinden soll. Das Konvertieren erfolgt mit dem Tool „ESF Database Migration Toolkit“ (http://www.easyfrom.net/de/). Bei der Migration wird bei jedem TEXT-Wert das erste Zeichen in ein „t“ transformiert (Freeware-Beschränkung). Da hier jedoch ausschließlich Zahlenwerte (int11 und double) verwendet werden resultieren hieraus keinerlei Einschränkungen bei der Migration.

3.2.2 Histogramme

Um die Verteilung der Daten in der SETI-Datenbank überschaubarer zu machen, wurden diese mit Hilfe von Histogrammen visualisiert. Hierbei werden auf der x-Achse die zu betrachtenden Werte (Prozessorleistung, Größe des Arbeitsspeichers und Anzahl der Prozessoren) in Bereiche eingeteilt und auf der y-Achse mittels Balken dargestellt, wie viele der Knoten aus der SETI-Datenbank in den jeweiligen Bereich fallen.

Die Einteilung der Bereiche erfolgt bei der Prozessorleistung linear in jeweils 250 Mega-Double-FPOPs-Schritten von 0 bis 4500. Bei der Größe des Arbeitsspeichers und bei der Anzahl der Prozessoren werden der Übersichtlichkeit halber die in der Praxis auftauchenden üblichen Grenzwerte verwendet (256, 512, 1024, 2048, 4096 und 8192 bei der Größe des Arbeitsspeichers und 1, 2, 4, 8, 16, 32, 64 und 96 bei der Anzahl der Prozessoren).

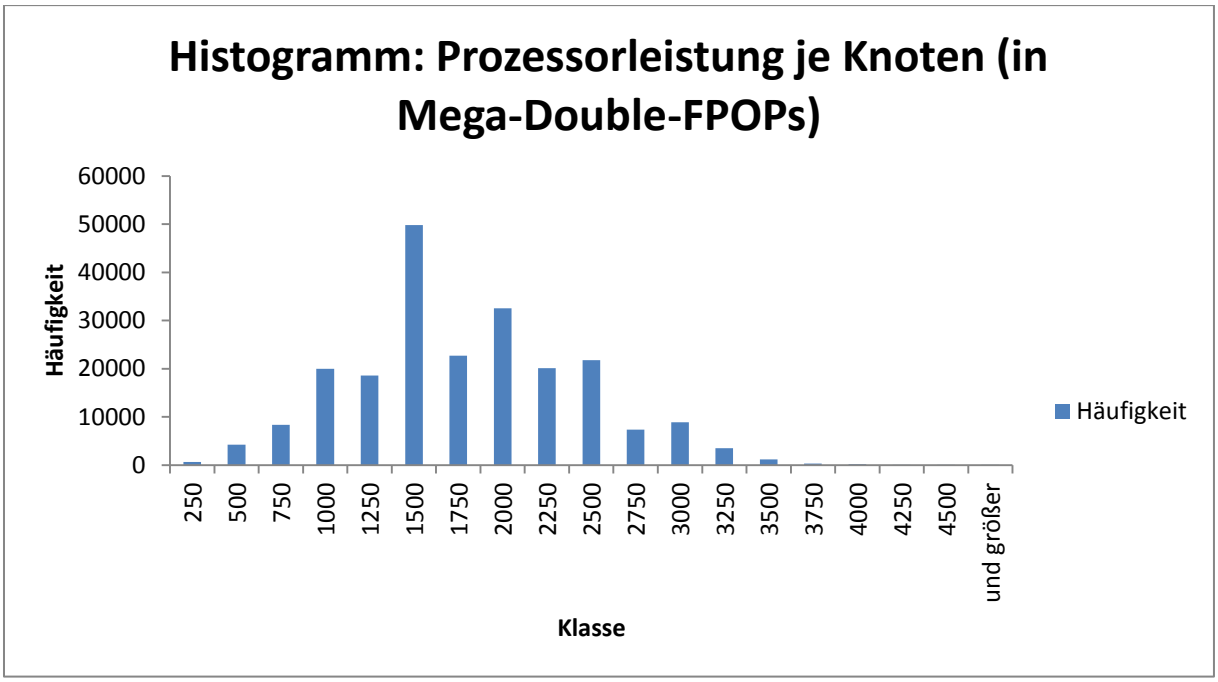


Abbildung 1: Histogramm Prozessorleistung je Knoten

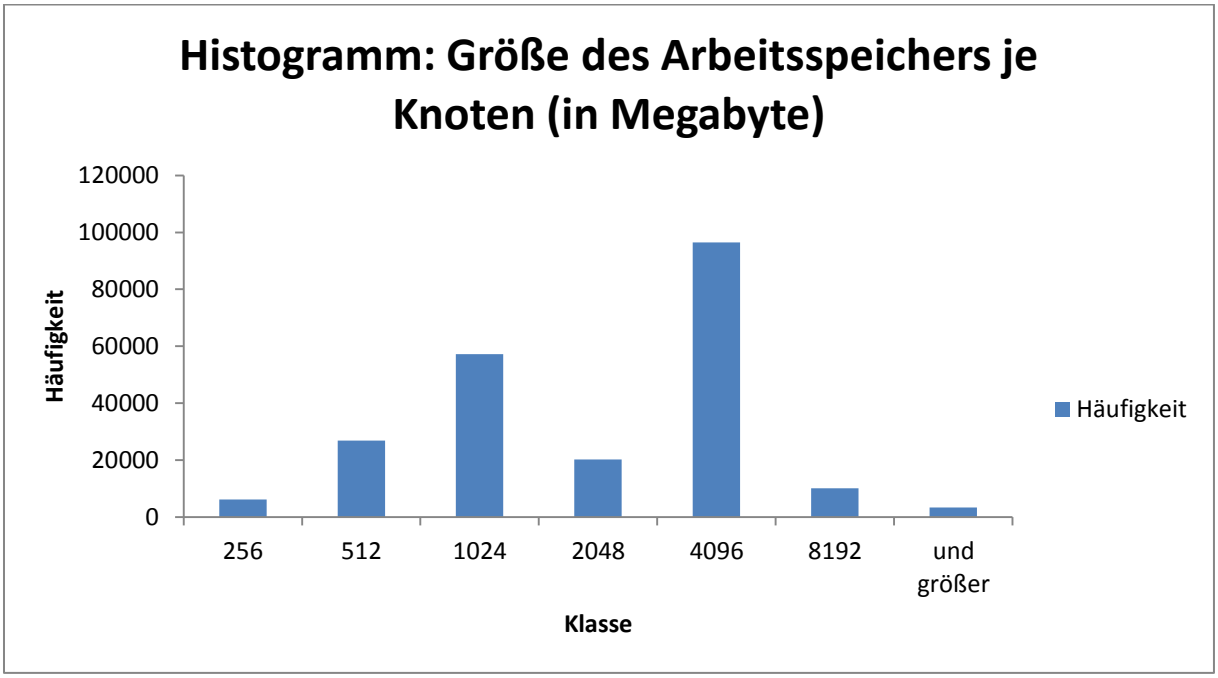


Abbildung 2: Histogramm Größe des Arbeitsspeichers je Knoten

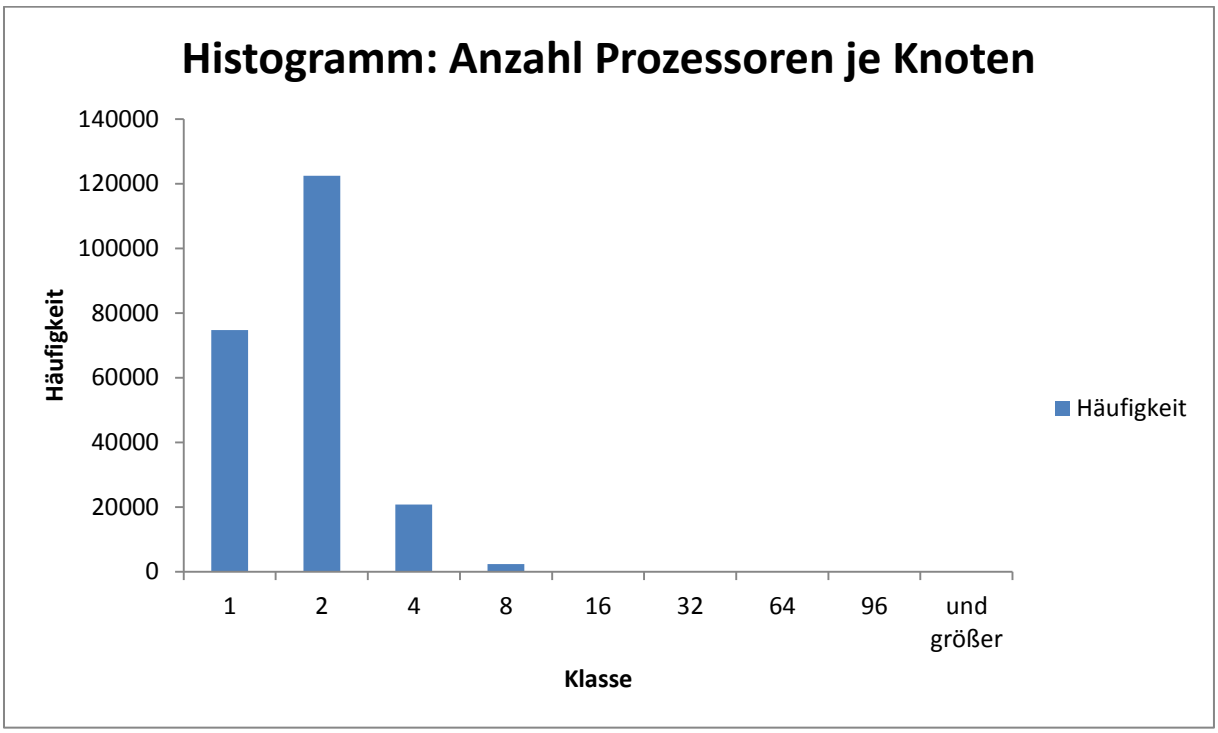


Abbildung 3: Histogramm Anzahl Prozessoren je Knoten

3.2.3 Prozessorgeschwindigkeit

Die Prozessorgeschwindigkeit findet sich in der FTA-Datenbank unter der Spalte dfpop_speed wieder, wobei es sich um Fließkommaoperationen mit doppelter Genauigkeit pro Sekunde handelt.

Zunächst wurden die Werte um den Faktor 5 verkleinert, so dass die resultierenden Werte ungefähr einer Prozessorgeschwindigkeit in Megahertz entsprechen. Grundlage dieser Umrechnung ist ein Paper von Derrick Kondo, Artur Andrzejak und David P. Anderson mit dem Titel „On Correlated Availability in Internet-Distributed Systems“ (http://boinc.berkeley.edu/boinc_papers/grid08.pdf) in dem folgende Aussage zu finden ist: „We see that nearly 10,000 hosts of speed 2 * 10^9 FPOPS (2GHz) are 90-100% available.“

Die Software Virtual Box ermöglicht es bei der Definition von virtuellen Maschinen die verwendete Prozessorzeit prozentual einzustellen (in der XML-Definition ist dies im Abschnitt für die CPU die Einstellung cpuexecutionCAP=“100“, wobei der Wert zwischen 1 und 100 liegen darf). Mit dieser Einstellung wird festgelegt, wie viel Prozessorzeit die CPU des Hosts für die Emulation der virtuellen CPU aufbringt.

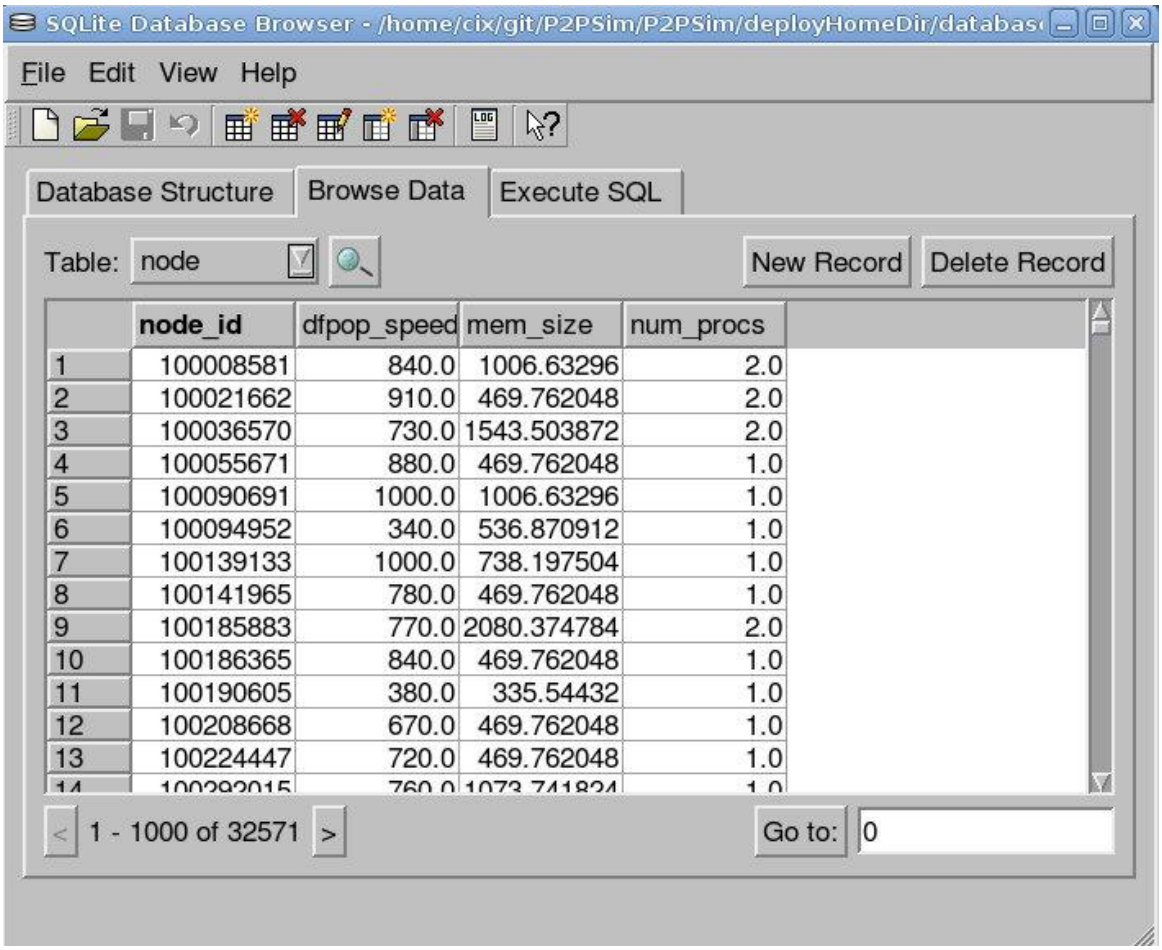
Da die auf dem Rechencluster des IPVS verwendeten Prozessoren (AMD Opteron 254) über eine Taktrate von 1000 MHz verfügen, die vorliegenden Daten aber durchaus höhere Taktraten aufweisen, ergeben sich im weiteren Verlauf nun zwei mögliche Vorgehensweisen:

1. Ersatzloses Verwerfen von Datensätzen, welche eine Taktrate > 1 GHz vorweisen
2. Lineares Skalieren aller Datensätze, so dass die höchste vorkommende Taktrate nach der Skalierung genau 1 GHz entspricht.

Implementiert wurde Möglichkeit 1. Zum einen liegen auch nach der Kappung bei 1 GHz noch hinreichend viele Datensätze vor (von zuvor 219.719 Datensätzen bleiben 32.571 Datensätze übrig), zum anderen wird das Verhältnis Prozessorgeschwindigkeit zu Arbeitsspeicher dadurch nicht verfälscht.

Eine Implementierung von Möglichkeit 2 wäre ebenfalls möglich und wird in Kapitel 5 (Ausblick) als Vorschlag für eine Erweiterung der Funktionalität von P2PSim gegeben. Hierzu liegt im Installationsverzeichnis im Unterordner databaseFTA die Datei nodes_complete.db, welche alle 219.719 Datensätze enthält.

Die nachfolgende Tabelle zeigt einen Ausschnitt aus der Seti-Datenbank, welche für die Verwendung von P2PSim wie oben beschrieben vorbereitet worden ist. Sie enthält die 4 Spalten „node_id“, „dfpop_speed“, „mem_size“ und „num_procs“. Insgesamt liegen 32571 Datensätze vor.



	node_id	dfpop_speed	mem_size	num_procs
1	100008581	840.0	1006.63296	2.0
2	100021662	910.0	469.762048	2.0
3	100036570	730.0	1543.503872	2.0
4	100055671	880.0	469.762048	1.0
5	100090691	1000.0	1006.63296	1.0
6	100094952	340.0	536.870912	1.0
7	100139133	1000.0	738.197504	1.0
8	100141965	780.0	469.762048	1.0
9	100185883	770.0	2080.374784	2.0
10	100186365	840.0	469.762048	1.0
11	100190605	380.0	335.54432	1.0
12	100208668	670.0	469.762048	1.0
13	100224447	720.0	469.762048	1.0
14	100292015	760.0	1073.741824	1.0

Tabelle 2: Ausschnitt aus der bearbeiteten Seti-Datenbank

3.3 Klassendiagramm

Bei P2PSim handelt es sich um eine Java-Anwendung, welche über die in Bild 1 dargestellten Klassen verfügt.

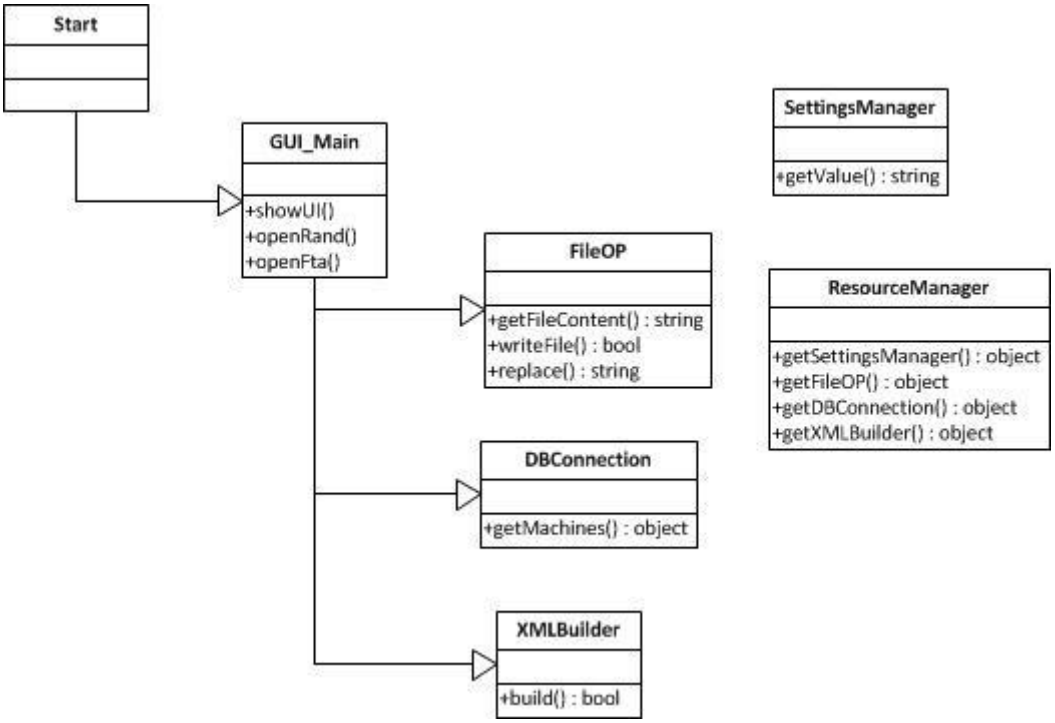


Abbildung 4: Klassendiagramm

Die Klasse „Start“ dient als Einstiegspunkt für das Programm.

Die Klasse „GUI_Main“ implementiert die grafische Benutzerschnittstelle und wird durch die Klasse „Start“ instanziiert. Neben verschiedenen Funktionen für die Darstellung der GUI enthält die Klasse noch die Funktion „execShellCommandBool“, welche es ermöglicht, Bash-Befehle an das System abzusetzen, welches die P2PSim-Anwendung ausführt. Je nachdem, ob die Ausführung des Befehls erfolgreich war oder nicht wird ein Boolean-Wert zurückgegeben (true: erfolgreich, false: nicht erfolgreich). Die Funktionalitäten der Schaltflächen in der GUI werden ebenfalls durch diese Klasse realisiert.

Die Klasse „FileOP“ ist für Operationen mit Textdateien zuständig. Sie beinhaltet Funktionen zum einlesen einer Textdatei (getFileContent), zum Schreiben einer Textdatei (writeFile) und zum Ersetzen von Strings in einer Textdatei (replace).

Die Klasse „DBConnection“ implementiert die Verbindung zu der mit dem Programm ausgelieferten, bearbeiteten SETI-SQLite-Datenbank. Als einzige Funktion steht getMachines zur Verfügung, welche die gewünschte Anzahl an Datensätzen (zufällig ausgewählt) zurückgibt.

Die Klasse „XMLBuilder“ realisiert die Erstellung der XML-Definitionen für die virtuellen Maschinen. Hierzu wird die originale XML-Datei mehrfach kopiert und die entsprechenden Werte werden in die neuen Dateien eingetragen. Als Datenquelle für die Werte dienen entweder die SETI-Datenbank oder die in der GUI gemachten Einstellungen zu Prozessorgeschwindigkeit und Arbeitsspeichergröße.

Die Klasse ResourceManager dient als Schnittstelle für den Zugriff auf die verwendeten Ressourcen (auf die Klassen SettingsManager, GUI_Manager, FileOP und XMLBuilder). Die Gewährleistung eines exklusiven Zugriffs erfolgt angelehnt an das Design Patterns „Singleton“ (vgl. S. 127 ff. in Design Patterns : E. Gamma, R. Helm, R. Johnson, J. Vlissides).

Die Klasse „SettingsManager“ organisiert die Verbindung der Java-Anwendung und der Textdatei „settings.txt“, in welcher der Benutzer vor Programmstart individuelle Einstellungen vornimmt. Alle notwendigen Zusatzinformationen, die benötigt werden, sind in dieser Datei hinterlegt. Auf diese Informationen wird über die Funktion getValue() zurückgegriffen.

3.4 Ablaufdiagramm

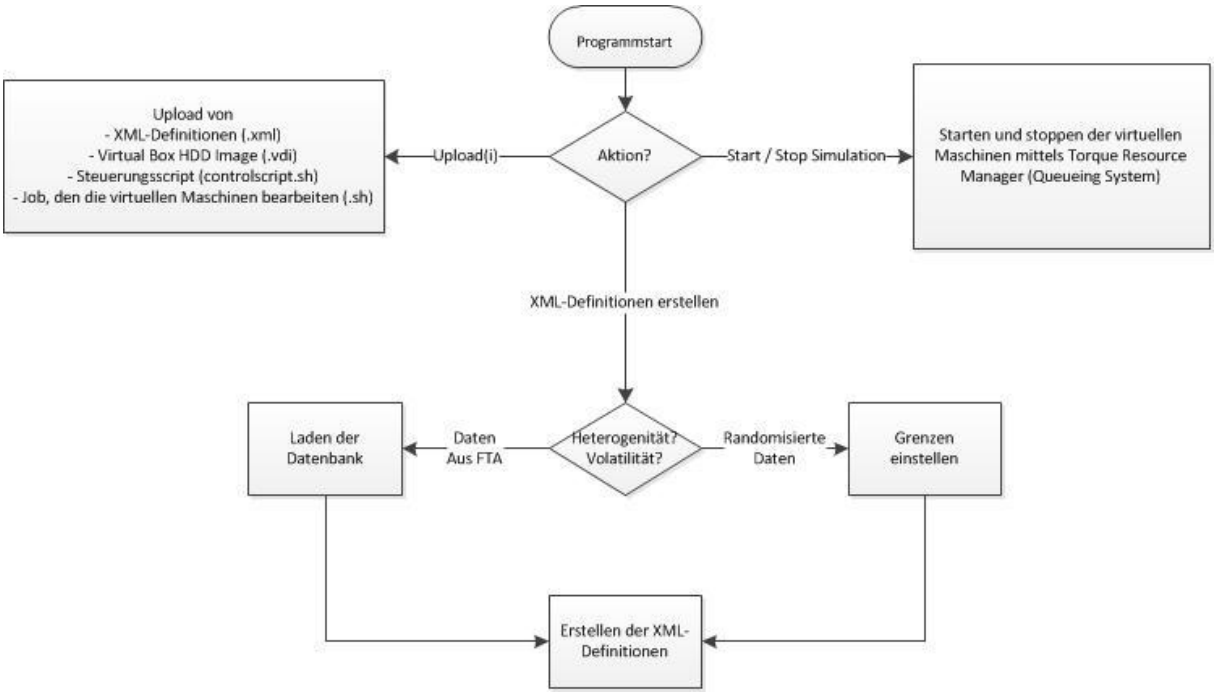


Abbildung 5: Ablaufdiagramm

Wie dem Ablaufdiagramm zu entnehmen ist, hat der Benutzer drei grundlegende Möglichkeiten:

- Erstellen der für die Simulation benötigten XML-Definitionen für die virtuellen Maschinen
- Hochladen der für die Simulation benötigten Dateien
- Starten und stoppen der Simulation auf dem Rechencluster
- Einstellen der gewünschten Volatilität

Darüber hinaus besteht zu jedem Zeitpunkt die Möglichkeit, Programmeinstellungen zu speichern bzw. zu laden.

3.5 Rechencluster des IPVS

Für das weitere Verständnis des Programmes ist es notwendig, zunächst einige Informationen zu Architektur und Funktionsweise des Clusters des IPVS anzugeben.

Der Rechencluster des IPVS besitzt ein Frontend und eine Vielzahl an Rechenknoten.

Der Zugriff auf das Frontend erfolgt per SSH (ssh <user>@vsdsl.informatik.uni-stuttgart.de). Wie auf Linux-Systemen üblich befindet sich das Benutzerverzeichnis unter /home/<user>. Dieses Verzeichnis ist sowohl vom Frontend aus zugreifbar, als auch von den einzelnen Rechenknoten (compute-0-1 usw.).

Der Zugriff auf die Rechenknoten erfolgt ebenfalls per SSH (z.B. ssh <user>@compute-0-1). Wie bereits erwähnt kann von den Rechenknoten aus direkt auf das Verzeichnis /home/<user> zugegriffen werden, wobei jeder Knoten die gleiche Sicht auf dieses Verzeichnis hat. Änderungen in diesem Verzeichnis sind also systemweit. Darüber hinaus besitzt jeder Rechenknoten ein lokales Verzeichnis /state/partition1, welches exklusiv jeweils diesem Rechenknoten zugeordnet ist. Änderungen, die in diesem Verzeichnis gemacht werden, haben also nur Auswirkungen auf diesen speziellen Knoten.

Die auf dem Cluster installierten Programme stehen systemweit zur Verfügung.

3.7 Erstellen der XML-Definitionen

Das Erstellen der XML-Definitionen übernimmt die Java-Klasse XMLBuilder.java. Hierbei wird mit Hilfe des Linux-Programms „cp“ jeweils die Originalvorlage „VBOX-DefinitionXML.xml“ kopiert und im Ordner „machines“ abgelegt, wobei eine fortlaufende Nummer an den Dateinamen angehängt wird (z.B. „VBOX-DefinitionXML13.xml“).

Zunächst wird mit Hilfe des Linux-Programms „sed“ die UUID (eindeutige Identifikationsnummer für die virtuelle Maschine) angepasst. In der Originalvorlage endet die UUID mit der Ziffernfolge „1111111111“. Diese wird dahingehend modifiziert, dass für jede neu erstellte XML-Definition 1 zu dieser Zifferfolge addiert wird. Dieses Vorgehen ist notwendig, damit die neuen XML-Definitionen im weiteren Verlauf in der Software „Virtual Box“ als neue Maschinen importiert werden können. Jede Maschine muss zwangsweise eine sich von allen anderen Maschinen unterscheidende UUID besitzen.

Nun werden die beiden Werte für die Größe des Arbeitsspeichers (Platzhalter im Original: „RAMSizeUpdate“) und für die Prozessorgeschwindigkeit (Platzhalter im Original: „CPUSizeUpdate“) angepasst. Je nachdem, ob eine zufällige Generierung der Werte in bestimmten Grenzen gewählt wurde oder ob als Datenquelle das FTA dienen soll, wird der neue Wert in die zuvor kopierte XML-Definition eingesetzt.

Der gesamte Vorgang wird entsprechend der gewählten Anzahl der zu simulierenden Maschinen wiederholt.

3.8 Hochladen der benötigten Dateien auf den Server

Alle Dateien, die auf das Frontend des Clusters hochgeladen werden, werden dort in dem Verzeichnis abgelegt, welches in der Datei „settings.txt“ unter dem Punkt „FrontendDirectory“ angegeben worden ist.

Hochgeladen werden folgende Dateien:

- die XML-Definitionen für die virtuellen Maschinen („VBOX-DefinitionXML<i>.xml“)
- die Steuerungsdateien für die Knoten des Rechenclusters („control<i>.sh“)
- die von den virtuellen Maschinen zu bearbeitende Aufgabe („job.sh“)

Das Hochladen der Dateien erfolgt per SCP direkt aus der Klasse „GUI_Main“ heraus.

Es empfiehlt sich, manuell die beiden Dateien „onStartup.sh“ und „rc.local“ ebenfalls auf den Cluster hochzuladen, da diese später in die erste virtuelle Maschine, die manuell installiert werden muss, eingefügt werden müssen.

3.9 Steuerungsskripte

Allgemeines

Durch Klicken auf die Schaltfläche „generate control scripts“ werden lokal Skripte erstellt, welche den weiteren Programmablauf und die Logik auf dem Server steuern. Im Allgemeinen unterschiedliche Ausführungen dieses Skripts werden im weiteren Verlauf auf dem Cluster mit dem Torque Resource Manager an die an der Simulation teilnehmenden Knoten als Arbeitsauftrag gequeued. Ausgeführt werden diese Skripte also auf den einzelnen Rechenknoten des Clusters.

Folgende Aufgaben werden durch das Script bewerkstelligt:

- Kopieren des von den virtuellen Maschinen zu bearbeitenden Jobs vom Frontend zum Rechenknoten.
- Registrieren einer neuen virtuellen Maschine namens „VM<i>“, wobei i entsprechend der gewünschten Anzahl der zu erstellenden Maschinen hochgezählt wird.
- Klonen des originalen Festplattenimages in ein neues Festplattenimage namens „HDD<i>.vdi“, wobei i entsprechend der gewünschten Anzahl der zu erstellenden Maschinen hochgezählt wird.
- Einbinden des neu erzeugten Festplattenimages in die neu erzeugte virtuelle Maschine.
- Starten der virtuellen Maschine (--type headless bedeutet, dass es sich hierbei um einen Hintergrundprozess handelt welcher unsichtbar bleibt).
- Simulation der vom Benutzer gewählten Volatilität (wird nachfolgend noch näher erläutert).
- Nach Beenden der Instanz der virtuellen Maschine: Entfernen des SATA-Controllers (und somit Freigeben des Festplattenimages).
- Nach Beenden der Instanz der virtuellen Maschine: Löschen des Festplattenimages.
- Nach Beenden der Instanz der virtuellen Maschine: Entfernen der virtuellen Maschine aus der Software Virtual Box.

Simulation der Volatilität

Unter dem Programmpunkt „volatility“ hat der Benutzer die Möglichkeit auszuwählen, wie hoch die prozentuale Wahrscheinlichkeit sein soll, dass eine Maschine innerhalb von 24 Stunden ausfällt. Hierbei bedeutet 100 (%), dass jede Maschine innerhalb von 24 mindestens einmal ausfällt und 0 (%), dass keine Maschine innerhalb von 24 Stunden ausfällt.

Die Simulation der Volatilität übernehmen die Steuerungsskripte, wobei der dafür zuständige Abschnitt im Folgenden dargestellt und erläutert wird.

Abschnitt aus der Datei “control.sh”:

```
#simulation of the volatility
while [ 1 ]
	do ablaufzeit=`echo "$RANDOM*32767+$RANDOM%(100/$VOL*24*60*60-10*60)+10*60" | bc`
	counter=0
	while [ 1 ]
		do sleep 1
		vboxmanage list runningvms | grep "\"VM$MACHINE\"" >/dev/null
		if [ $_ ]   # maschine beendet?
		then exit 0
		fi
		counter=`expr $counter + 1`
		if [ $counter -ge 	$ablaufzeit ]
			then vboxmanage controlvm VM$MACHINE poweroff
			sleep 120
			vboxmanage startvm VM$MACHINE --type headless
			sleep 120
			counter=0
			continue
		fi
	done
done
```

Die Simulation der Volatilität erfolgt durch zwei verschachtelte WHILE-Schleifen, welche jeweils so lange laufen, bis sie explizit verlassen werden (entweder durch continue oder durch exit 0).

Die Variable „ablaufzeit“ wird beim Betreten der äußeren Schleife mit Hilfe des Programms bc ermittelt. Hierbei handelt es sich um eine Zufallszahl die entsprechend dem gewünschten Grad der Volatilität berechnet wird.

Falls der Grad 100 (%) betragen soll, so liegt diese Zufallszahl zwischen 0 und 86400 (Anzahl der Sekunden eines Tages).

Falls der Grad 50 (%) beträgt, so liegt die Zufallszahl zwischen 0 und 2 * 86400, wodurch die Wahrscheinlichkeit, dass die virtuelle Maschine innerhalb von 24 Stunden ausfällt halbiert wird.

Allgemein ausgedrückt ist die Formel zur Berechnung der Zufallszahl folgende:

Sekunden = Anzahl Sekunden eines Tages * (100 / Grad der Volatilität)

Aus praktischen Gründen sollten Werte zwischen 0 und 600 Sekunden jedoch nicht vorkommen, so dass jede virtuelle Maschine mindestens 10 Minuten Zeit für das Hochfahren hat.

In der inneren Schleife (welche ungefähr ein Mal pro Sekunde durchlaufen wird) wird mittels „vboxmanage list runningvms“ und dem Linux-Programm „grep“ nun überprüft, ob die virtuelle Maschine bereits mit ihrer Arbeit fertig ist und sich runtergefahren hat. Dies ist notwendig, damit der Rechenknoten des Clusters umgehend danach wieder freigegeben wird, da ansonsten das Skript noch so lange weiterlaufen würde, bis die oben ermittelte Anzahl an Sekunden abgelaufen ist.

Wenn also die Instanz der virtuellen Maschine nicht mehr aktiv ist wird das Skript abgebrochen (exit 0). Ansonsten wird überprüft, ob die mitzählende Variable „counter“ den zuvor ermittelten Zufallswert erreicht hat. Wenn dies der Fall ist wird die Maschine heruntergefahren und nach 120 Sekunden wieder neu gestartet. Nach weiteren 600 Sekunden wird die innere Schleife verlassen und das Skript beginnt sozusagen wieder von vorne.

3.10 Starten und Stoppen der Simulation

Starten

Durch einen Klick auf den Schaltfläche „start simulation“ wird die Simulation gestartet. Hierzu werden die zuvor generierten Steuerungsskripte mit Hilfe des Torque Resource Managers als Aufgaben an die teilnehmenden Rechenknoten verteilt (Verbindung vom Client zum Frontend des Clusters per SSH).

Der TRM sorgt automatisch dafür, dass jedes der Steuerungsskripte auf einem der Rechenknoten ausgeführt wird. Der weitere Programmablauf wird nun durch diese Skripte gesteuert und läuft ohne grafische Benuteroberfläche im Verborgenen. Auf jedem teilnehmenden Rechenknoten wird die Instanz der virtuellen Maschine „headless“ (ohne UI) gestartet. Sobald diese Instanz hochgefahren ist, wird per SCP der Job angezogen (in das knotenlokale Verzeichnis, welches in der Datei „settings.txt“ unter „NodeDirectory“ angegeben worden ist) und ausgeführt. Die Ergebnisse des Jobs werden temporär in einem definierten Ordner innerhalb der virtuellen Maschine abgelegt und bei erfolgreicher Beendigung des Jobs zu zurück in den Benutzerordner des Frontends kopiert.

Stoppen

Die Simulation wird mittels folgenden Java-Befehlen gestoppt:

```
String cmd1 = "\"bash -c \\\"/usr/local/bin/qdel \\\\\\\$(/usr/local/bin/qselect -u " +
SM.getValue("SshUser") + ")\\\"\"";

String cmd = "ssh " + SM.getValue("SshUser") + "@" + SM.getValue("SshServer") + " " + cmd1;

System.out.println(cmd);

success = success && GUI_Main.execShellCmdBool(cmd);
```

In der Variablen „cmd1“ wird der auszuführende Linux-Befehl als String aufgebaut. Da dieser aus zwei aufeinanderfolgenden und voneinander abhängenden Befehlen besteht, muss mittels „bash –c“ eine Bash um diese beiden Befehle gewrappt werden. Die auszuführenden Befehle, um im TRM alle Aufgaben eines Benutzers zu löschen lauten „qdel“ und „qselect -u abc123“, wobei abc123 für den entsprechenden Benutzernamen steht. Hier wird mittels des SettingsManagers (SM) der richtige Benutzername eingesetzt.

Im Folgenden wird „cmd1“ an „cmd“ angehängt, wodurch „cmd1“ per SSH auf dem Frontend des Clusters ausgeführt werden kann.

Zur Kontrolle wird der Inhalt von „cmd“ auf die Standardausgabe des Clients ausgegeben, welcher P2PSim ausführt, und anschließend mittels der Funktion GUI_Main.execShellCmdBool() ausgeführt. Ob die Ausführung erfolgreich war ist im Anschluss am Wert der Boolean-Variable „success“ abzulesen.

3.11 Auszuführender Job

Der Job, der von den virtuellen Maschinen im Zuge der Simulation bearbeitet werden soll, muss ein Bash-Skript sein (welches z.B. andere Skripte aufrufen, binäre Dateien ausführen oder andere Aufgaben übernehmen kann).

Damit die Ergebnisse dieses Jobs später an zentraler Stelle auf dem Cluster zusammengetragen werden können, muss die Ausgabe des Skripts, z.B. erstellte Dateien, an einen definierten Ort erfolgen.

Das Skript befindet sich, wenn es in der Instanz der virtuellen Maschine ausgeführt wird, dort in folgendem Ordner:

/home/P2PSim.

Die Ergebnisse müssen in folgenden Ordner geschrieben werden (hierfür muss in dem Skript Sorge getragen werden, es muss also vorher entsprechend angepasst werden):

/home/P2PSim/results.

Nach erfolgreicher Bearbeitung des Jobs werden die Ergebnisse aus der virtuellen Maschine heraus auf das Frontend des Clusters kopiert (diese Aufgabe übernimmt das Skript „onStartup.sh“, welches im Autostart der virtuellen Maschine vorhanden ist). Anschließend wird die virtuelle Maschine automatisch heruntergefahren. Hierzu muss am Ende des Job-Skripts folgender Befehl eingefügt werden:

shutdown –h now.

4. Benutzungsanleitung

4.1 Installation

4.1.1 Installation der virtuellen Maschinen auf den Knoten des Clusters

Die virtuellen Maschinen werden in Form von Instanzen einer Virtual Box - Debian-Installation (ohne GUI) auf den Knoten des Clusters implementiert. Es muss auf dem Cluster vor Benutzung von P2PSim eine virtuelle Maschine namens „VM“ installiert werden. Von dieser Maschine werden im weiteren Verlauf dann die XML-Definitionsdatei sowie das Festplattenimage benötigt.

Vorbereitungen:

Das nachfolgende Deployment der virtuellen Maschine erfolgt anhand dieser Beschreibung:

http://www.perkin.org.uk/posts/create-virtualbox-vm-from-the-command-line.html

Erstellen der virtuellen Festplatte (Name: HD1.vdi, Größe: 4 GB):

```
[kiesewsn@compute-0-1 P2PSim]$ vboxmanage createhd --filename HD1.vdi --size 4096
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Disk image created. UUID: 8ca618ab-1b42-4355-ab7b-67ca28c5cde2
```

Erstellen und Registrieren der virtuellen Maschine:

```
[kiesewsn@compute-0-1 P2PSim]$ vboxmanage createvm --name VM1 --ostype "Debian_64"
--register
Virtual machine 'VM1' is created and registered.
UUID: ed1862fb-fc0c-40b8-b354-af3091848288
Settings file: '/home/kiesewsn/VirtualBox VMs/VM1/VM1.vbox'
```

SATA-Controller erstellen und die zuvor erstellte virtuelle Festplatte anbinden:

```
[kiesewsn@compute-0-1 P2PSim]$ VBoxManage storagectl VM1 --name "SATA Controller" -
-add sata --controller IntelAHCI

[kiesewsn@compute-0-1 P2PSim]$ VBoxManage storageattach VM1 --storagectl "SATA
Controller" --port 0 --device 0 --type hdd --medium HD1.vdi
```

DIE-Controller erstellen und Installationsmedium anbinden:

```
[kiesewsn@compute-0-1 P2PSim]$ VBoxManage storagectl VM1 --name "IDE Controller" --
add ide
```

```
[kiesewsn@compute-0-1 P2PSim]$ VBoxManage storageattach VM1 --storagectl "IDE
Controller" --port 0 --device 0 --type dvddrive --medium debian-6.0.6-amd64-
netinst.iso
```

Das Installationsmedium (Debian-Netinstall ISO-Datei) wurde zuvor in das Verzeichnis /state/partition1/kiesewsn/P2PSim mittels wget runtergeladen.

Verschiedene Settings vornehmen:

```
[kiesewsn@compute-0-1 P2PSim]$ VBoxManage modifyvm VM1 --ioapic on

[kiesewsn@compute-0-1 P2PSim]$ VBoxManage modifyvm VM1 --boot1 dvd --boot2 disk --
boot3 none --boot4 none

[kiesewsn@compute-0-1 P2PSim]$ VBoxManage modifyvm VM1 --memory 1024 --vram 128

[kiesewsn@compute-0-1 P2PSim]$ VBoxManage modifyvm VM1 --nic1 bridged --
bridgeadapter1 e1000g0
```

Folgende Beschreibung hilft dabei, den bis zu diesem Zeitpunkt auftauchenden Netzwerkfehler zu beheben:

http://yessajah.wordpress.com/2012/01/16/failed-to-opencreate-the-internal-network-verr_intnet_flt_if_not_found/

Anpassen der Netzwerk-Adapter-Einstellungen:

```
[kiesewsn@compute-0-1 P2PSim]$ VBoxManage modifyvm VM1 –bridgeadapter1 eth0
```

Installation des Gastsystems:

Die virtuelle Maschine ist nun so weit vorbereitet, dass diese gestartet werden kann und das (Linux-) Betriebssystem installiert werden kann. Da die Instanzen der virtuellen Maschinen später „headless“ gestartet werden, also ohne grafische Benutzerschnittstelle, ist es nicht nötig, eine Desktop-Umgebung wie beispielsweise Gnome oder KDE zu installieren.

Anpassungen im Gastsystem:

Damit die virtuelle Maschine nach dem Hochfahren damit beginnt, den vorgesehenen Job zu bearbeiten und im Anschluss daran die entstandenen Ergebnisse und die Log-Datei in das Frontend-Verzeichnis „/home/<user>/P2PSim/results“ schreibt, muss das Script „onStartup.sh“ in den Autostart der virtuellen Maschine eingefügt werden. Hierzu muss in der virtuellen Maschine in der Datei „/etc/rc.local“ das Skript „onStartup.sh“ gestartet werden. Das Skript „onStartup.sh“ kopiert per SCP den Job in die Instanz der virtuellen Maschine, führt den Job aus und schreibt nach Beendigung des Jobs die Ergebnisse zurück auf das Frontend.

Bevor das Skript „onStartup.sh“ in die virtuelle Maschine kopiert wird, müssen die 3 Variablen „SSH_USER“, „SSH_ADRESS“ und „FRONTENDDIR“ entsprechend der tatsächlichen Gegebenheiten (und konform zu den Einstellungen in der Datei „settings.txt“) angepasst werden.

Damit das Skript „onStartup.sh“ direkt in der virtuellen Maschine verfügbar ist, muss es in dieser in den Ordner „/home/P2PSim“ eingefügt werden. Falls dieser Ordner noch nicht existiert, muss dieser angelegt werden. (ACHTUNG! Der zuvor genannte Ordner befindet sich innerhalb einer laufenden Instanz der virtuellen Maschine, also auf dem Festplattenimage dieser. Es ist hier NICHT der Benutzerordner auf dem Frontend des Clusters bzw. auf den Rechenknoten des Clusters gemeint.)

Darüber hinaus muss folgender Ordner in der virtuellen Maschine angelegt werden:

/home/P2PSim/results.

Hinweis: Sowohl das Skript „onStartup.sh“ als auch das Skript „rc.local“ sind nach der Installation von P2PSim im Ordner „/home/<user>/P2PSim/scripts“ zu finden. Damit „onStartup.sh“ wie gewünscht funktioniert muss der öffentliche Schlüssel der virtuellen Maschine an den Cluster bekannt gegeben werden (sshkeygen -t rsa -b 2048; ssh-copy-id user@server). Siehe hierzu im folgenden Abschnitt 4.1.2 die genauere Beschreibung zu diesem Vorgang.

Anpassungen der XML-Definition für das Gastsystem:

Die Konfiguration der virtuellen Maschinen erfolgt durch XML-Definitionen, in welchen unter anderem der Name der virtuellen Maschine, die UUID des Images der virtuellen Festplatte, die Größe des Arbeitsspeichers und die Geschwindigkeit des Prozessors angegeben werden.

Änderungen an der zugehörigen XML-Definition:

Damit die XML-Definition der erstellten virtuellen Maschine mit P2PSim verwendet werden kann, sind innerhalb der Datei folgende Modifikationen notwendig (farbig markiert ist im Folgenden der genaue Textbereich, der angepasst werden muss, wobei jeweils exakt die farbig markierten Strings in der XML-Datei eingetragen werden müssen):

- Anpassen des Platzhalters für die Größe des Arbeitsspeichers (<Memory RAMSize="RAMSizeUpdate" PageFusion="false"/>)
- Anpassen des Platzhalters für die Geschwindigkeit des Prozessors (<CPU count="1" executionCap="CPUSizeUpdate" hotplug="false">)
- Anpassen der UUID für die virtuelle Maschine, so dass diese mit der Ziffernfolge „1111111111“ endet.

An Stelle der bei der Installation des Gastsystems gewählten Größe des Arbeitsspeichers muss der String „RAMSizeUpdate“ eingefügt werden. Diese Anpassung ist notwendig, um es P2PSim zur Laufzeit zu ermöglichen, in den generierten neuen XML-Definitionen an dieser Stelle individuelle Werte einzufügen.

An Stelle der bei der Installation des Gastsystems gewählten zugewiesenen Prozessorzeit muss der String „CPUSizeUpdate“ eingefügt werden. Diese Anpassung ist notwendig, um es P2PSim zur Laufzeit zu ermöglichen, in den generierten neuen XML-Definitionen an dieser Stelle individuelle Werte einzufügen.

Im Abschnitt <StorageControllers> müssen nun die eingebundenen Medien entfernt werden. Hierzu werden die kompletten Abschnitte zwischen und inklusive <AttachedDevice> und </AttachedDevice> gelöscht. Die UUID des eingebundenen Festplattenimages (im SATA-Controller) muss in der Datei settings.txt eingetragen werden.

Die fertig angepasste XML-Definitions-Datei muss im Installationsverzeichnis von P2PSim im Unterordner „machines“ abgelegt werden unter dem Namen „VBOX-DefinitionXML.xml“.

Weiter zu beachten ist, dass das originale Festplattenimage nicht gelöscht werden darf.

4.1.2 Installation des Programms auf einem Linux-Client

Entpacken des Archivs

Das Programm wird als Dateiordner ausgeliefert, gepackt als tar.gz-Archiv. Dieses muss in ein beliebiges Verzeichnis auf einem Linux-System entpackt werden (z.B. nach „~/P2PSim“). Die Ordnerstruktur, die beim entpacken des Archivs entsteht darf nicht verändert werden.

settings.txt anpassen

Im Unterordner „settings“ befindet sich die Datei settings.txt. In dieser müssen die Einstellungen im ersten Abschnitt (adjust manually) angepasst werden auf die tatsächliche Umgebung:

- SshServer: Die Adresse (bzw. URI) des Cluster-Frontends
- SshUser: Der Benutzer, welcher per SSH Zugriff auf das Cluster-Frontend hat
- P2PSimHomeDir: Das Verzeichnis auf dem Client, in dem sich die entpackten Programmdateien befinden
- FrontendDirectory: Das Verzeichnis des Clusters, in das P2PSim Dateien hochladen darf und auf das auch die Nodes Zugriff haben („/home/<user>/P2PSim“)
- NodeDirectory: Das Verzeichnis auf den Knoten, welches von P2PSim benutzt werden soll („/state/partition1/<user>/P2PSim“)
- HDDUUID: Die UUID des Festplattenimages, welches bei der Installation der virtuellen Maschine auf dem Cluster angelegt wird

Die Einstellungen im zweiten Abschnitt (will be adjusted automatically) sind nicht für die manuelle Konfiguration vorgesehen. P2PSim passt diese Einstellungen automatisch im laufenden Betrieb an.

Ordnerstruktur im Installationsverzeichnis

Folgende Ordnerunterstruktur befindet sich nach dem Entpacken der Anwendung auf dem Client:

- /home/<user>/P2PSim (Hauptverzeichnis des Programms)
- /home/<user>/P2PSim/bin (hier finden sich die binären Java-Dateien)
- /home/<user>/P2PSim/scripts (hier finden sich die verwendeten Skripte)
- /home/<user>/P2PSim/machines (hier befindet sich die Originalvorlage sowie die generierten XML-Definitionen für die virtuellen Maschinen)
- /home/<user>/P2PSim/settings (beinhaltet die Textdatei settings.txt, welche alle wichtigen Programmeinstellungen enthält, z.B. die Serververbindungsdaten)
- /home/<user>/P2PSim/databaseFTA (beinhaltet die SQLite-Datenbanken mit den Daten aus dem Failure Trace Archive)

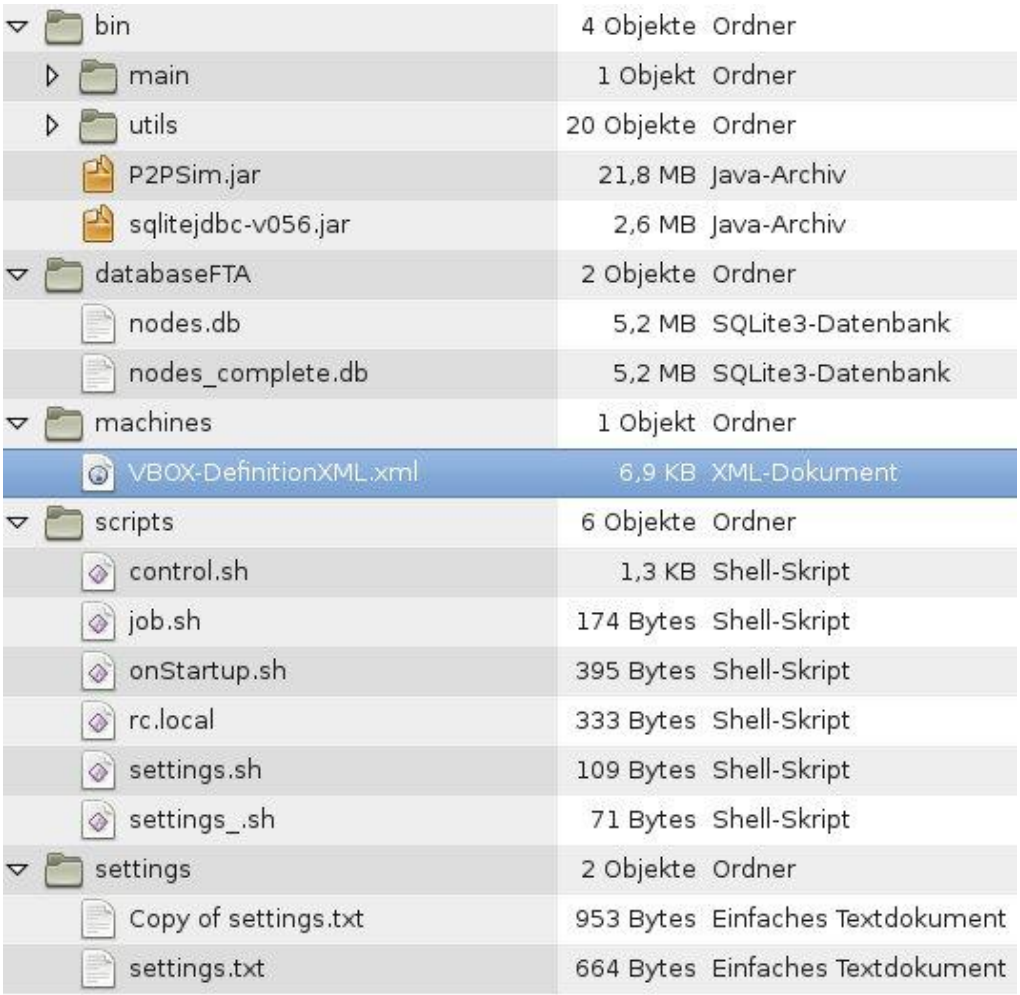


Abbildung 6: Ordnerstruktur

Schlüssel bekannt machen

Damit die Anwendung funktioniert muss der Client einen SSH Zugriff auf das Cluster-Frontend erhalten, bei dem nicht nach einem Passwort gefragt wird. Hierzu muss auf dem Server der öffentliche Schlüssel des Client hinterlegt werden. Die kann sehr leicht mit folgendem Linux-Befehlen bewerkstelligt werden:

ssh-keygen -t rsa -b 2048

und

ssh-copy-id user@server

Hierbei muss „user“ durch den Benutzer auf dem Cluster ersetzt werden und „server“ durch den Namen oder die IP-Adresse des Clusters.

Der o.g. Befehl muss auf dem Computer ausgeführt werden, auf dem P2PSim gestartet wird, und zwar unter dem Benutzer, der später das Programm verwendet.

Es ist ratsam vor Verwendung von P2PSim den Erfolg dieses Befehls zu testen, indem eine SSH-Session gestartet wird (ssh user@server). Hierbei darf nicht nach dem Passwort gefragt werden.

Sollte der Befehl ssh-copy-id nicht zur Verfügung stehen, kann alternativ auch der folgende Befehl verwendet werden:

ssh-keygen -t rsa -b 2048

cat ~/.ssh/id_dsa.pub | ssh root@andere-linux-kiste ‘cat >> .ssh/authorized_keys

4.2 Benutzung

Nachfolgend wird die Verwendung von P2PSim sowie der Ablauf des Programms erläutert.

4.2.1 Starten des Programms

Die Hauptanwendung wird durch folgenden Befehl auf einen Linux-System gestartet (die Datei P2PSim.jar befindet sich im Unterordner „bin“):

java -jar P2PSim.jar

Es öffnet sich die grafische Benutzerschnittstelle der Anwendung mit deren Hilfe der Benutzer im Folgenden die komplette Bedienung des Programms durchführen kann (Voraussetzung hierfür ist, dass die Einstellungen in der Datei „settings.txt“ zuvor angepasst worden sind, vgl. Abschnitt 4.1.2).

P2PSim macht regen Gebrauch von Linux-Programmen. Diese Programme werden über Befehle gestartet, die im Hauptprogramm zusammengesetzt und dann über die Funktion „GUI_Main. execShellCmdBool()“ ausgeführt werden. Diese Befehle und deren Rückgabewert werden auf die Standardausgabe geschrieben, so dass es sich empfiehlt, das Terminal mit dem die Anwendung gestartet wurde, geöffnet zu lassen. Der Benutzer hat so die Möglichkeit, die aufgerufenen Programme nebst den zugehörigen Parametern zu überprüfen. Insbesondere bei auftauchenden Problemen ist dies sehr hilfreich.

4.2.2 Die grafische Benutzerschnittstelle

Die folgende Abbildung zeigt die grafische Benutzerschnittstelle, die sich dem Benutzer nach Starten der Anwendung P2PSim zeigt.

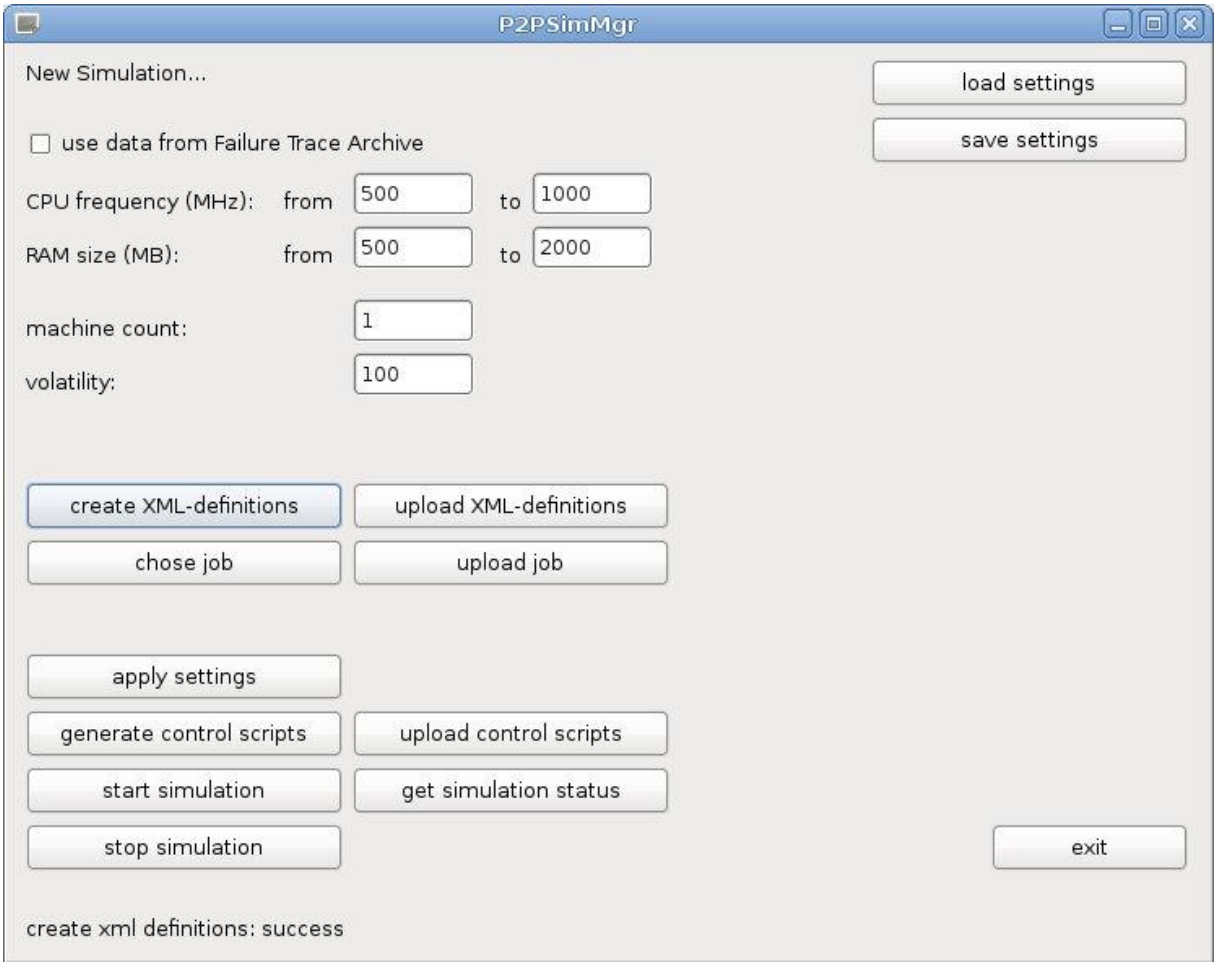


Abbildung 7: Grafische Benuterschnittstelle

Oben rechts befinden sich die Schaltflächen um den Programmzustand zu speichern bzw. wieder zu laden. Einem Ladevorgang muss zuvor irgendwann einmal ein Speichervorgang vorangegangen sein.

Im linken oberen Bereich hat der Benutzer die Möglichkeit, die gewünschten Einstellungen für die Simulation vorzunehmen. Wird das Schaltkästchen „use data from Failure Trace Archive“ aktiviert, so sind die Eingabefelder für die Prozessorgeschwindigkeit und die Größe des Arbeitsspeichers gesperrt, da die Werte hierfür dann automatisch aus der Seti-Datenbank ermittelt werden.

Auf jeden Fall eingegeben werden muss in den darunter liegenden Eingabefeldern die Anzahl der Maschinen, die für die Simulation generiert werden sowie der Grad der Volatilität.

Die nachfolgenden Schaltflächen im linken unteren Bereich sind der Reihe nach zu betätigen, also von links oben nach rechts unten. Sie stellen in ihrer Reihenfolge den logischen weiteren Programmablauf dar. Die XML-Definitionen für die virtuellen Maschinen sind also zunächst zu generieren, bevor sie hochgeladen werden. Im Anschluss daran wird ein Job ausgewählt (hierbei muss es sich um ein Bash-Skript handeln, siehe hierzu auch Kapitel 3.11), welcher ebenfalls hochgeladen wird.

Aus programminternen Gründen muss die Schaltfläche „apply settings“ betätigt werden, bevor mit der Generierung der Steuerungsskripte begonnen werden kann. Nachdem diese hochgeladen worden sind, kann die Simulation durch die Betätigung der entsprechenden Schaltfläche gestartet werden.

Die folgenden zwei Schaltflächen („get simulation status“ und „stop simulation“) sind optional, d.h. sie müssen für eine erfolgreich ablaufende Simulation nicht betätigt werden, da diese nach hinreichend langer Zeit von alleine terminiert und alle Instanzen der virtuellen Maschinen sowie ihrer dafür temporär erstellten Daten im Zuge dessen wieder gelöscht werden.

Ganz unten in der grafischen Benutzerschnittstelle befindet sich die Statuszeile. Zum Programmstart ist diese nicht sichtbar, da dort zu diesem Zeitpunkt nichts angezeigt wird. Jede Betätigung einer Schaltfläche erzeugt jedoch in dieser Statuszeile ein Feedback, das den Benutzer über Erfolg (oder Misserfolg) der ausgeführten Aktion informiert. In der oben dargestellten Abbildung wurde beispielsweise die Schaltfläche „create XML-definitions“ betätigt und die Statuszeile zeigt an, dass diese Dateien erfolgreich erzeugt worden sind.

Laden von Einstellungen:

Falls zuvor bereits ein Programmzustand gespeichert worden ist, kann dieser durch Betätigen der Schaltfläche „load settings“ erneut geladen werden. Geladen werden hierbei folgende Einstellungen:

- Anzahl der virtuellen Maschinen
- Untere Grenze für die Prozessorgeschwindigkeit
- Obere Grenze für die Prozessorgeschwindigkeit
- Untere Grenze für die Größe des Arbeitsspeichers
- Obere Grenze für die Größe des Arbeitsspeichers
- Benutzung der Daten aus dem FTA ja / nein
- Grad der Volatilität

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings "load settings: success" bzw. " load settings: no success".

Speichern von Einstellungen:

Der Programmzustand kann jederzeit persistent gemacht werden, so dass dieser z.B. nach einem Neustart des Programms geladen werden kann. Hierbei werden die Einstellungen, die der Benutzer gemacht hat, im unteren Abschnitt in der Datei „settings.txt“ gespeichert. Gespeichert werden hierbei folgende Einstellungen:

- Anzahl der virtuellen Maschinen
- Untere Grenze für die Prozessorgeschwindigkeit
- Obere Grenze für die Prozessorgeschwindigkeit
- Untere Grenze für die Größe des Arbeitsspeichers
- Obere Grenze für die Größe des Arbeitsspeichers
- Benutzung der Daten aus dem FTA ja / nein
- Grad der Volatilität

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings "save settings: success" bzw. " save settings: no success".

Eingabe der Daten für Arbeitsspeicher und Prozessor:

Zunächst entscheidet sich der Benutzer, ob er als Datengrundlage für die zu erstellenden virtuellen Maschinen die Daten aus dem Failure Trace Archive verwenden möchte (Fall 1), oder ob die Daten zufällig innerhalb der einzugebenden Grenzen generiert werden sollen (Fall 2). Dies geschieht über das Schaltkästchen „use data from Failure Trace Archive“.

Für Fall 1 sind die entsprechenden Textfelder gesperrt, d.h. der Benutzer hat keine Möglichkeit, dort Eingaben zu machen. Somit wird verhindert, dass eine manuelle Eingabe gewünscht ist, jedoch versehentlich das darüber liegende Schaltkästchen aktiviert ist.

Für Fall 2 müssen nachfolgend die Unter- und Obergrenzen für die Prozessorgeschwindigkeit und die Größe des Arbeitsspeichers vorgenommen werden. P2PSim generiert dann automatisch zufällige Werte mit gleicher Wahrscheinlichkeit für den gesamten eingegrenzten Bereich.

Anzahl der Maschinen

Hier gibt der Benutzer an, wie viele virtuelle Maschinen erzeugt werden sollen und somit im weiteren Verlauf an der Simulation teilnehmen. Es ist darauf zu achten, dass auf dem Cluster mindestens ebenso viele Rechenknoten für den Benutzer zur Verfügung stehen. Dies muss vorab geprüft werden, da eine automatische Überprüfung von P2PSim nicht durchgeführt wird. Sollten weniger Rechenknoten zur Verfügung stehen als virtuelle Maschinen erzeugt werden sollen, würde das dazu führen, dass einige Rechenkerne nach erfolgreichem Durchlauf der Simulation einer virtuellen Maschine im Anschluss eine weitere virtuelle Maschine erzeugen. Dieser Umstand ist darin begründet, dass die Rechenknoten über den TRM bedient werden.

Eingabe des Grades der Volatilität:

Der Benutzer hat über die grafische Bedienschnittstelle die Möglichkeit, die in der Simulation auftretende Volatilität der teilnehmenden virtuellen Maschinen einzustellen.

Prozentual kann angegeben werden, wie viele virtuelle Maschinen pro Tag ausfallen sollen.

0 (%)	=	keine Maschine fällt aus
100 (%)	=	alle Maschinen fallen aus

Dieser Grad der Volatilität wird von P2PSim in die Steuerungsskripte übertragen, wo sie dazu führen, dass zu einem zufällig ermittelten Zeitpunkt innerhalb einer Zeitspanne die virtuelle Maschine heruntergefahren und mit kurzer Verzögerung wieder hochgefahren wird. Diese Zeitspanne beträgt bei 100 (%) ungefähr 24 Stunden. Bei weniger als 100 (%) beträgt sie 24 Stunden * 100 / ausgewählter Grad, also z.B. 48 Stunden bei Grad 50 (%).

4.2.3 Auszuführende Aktionen

P2PSim ist so aufgebaut, dass der Arbeitsfluss „von oben nach unten“ in der grafischen Benutzerschnittstelle stattfindet.

So müssen beispielsweise die im oberen Abschnitt vorzunehmenden Einstellungen vorgenommen werden, bevor die XML-Definitionen (Schaltfläche „create XML-definitions“) erstellt werden können. Dies sollte sich der Benutzer nochmals klarmachen, damit im weiteren Verlauf nicht mit Daten gearbeitet wird, die nicht aktuell sind.

Zu jeder Aktion, die der Benutzer ausführt (= Betätigen einer Schaltfläche) erscheint unten im Programm in der Statuszeile eine Rückmeldung. Hierbei ist darauf zu achten, dass jede Aktion mit einem „success“ quittiert wird. Dies ist Voraussetzung für die korrekte Funktion von P2PSim.

Sollte eine Aktion mit „no success“ abgebrochen werden, so ist zunächst der Fehler zu identifizieren und zu beheben, bevor weitergearbeitet werden kann (z.B. würde ein nicht bekannt gemachter Schlüssel vom Client zum Cluster zu einem „no success“ bei allen Upload-Aktionen führen). Hierbei bietet die Standardausgabe des Terminals, mit dem P2PSim gestartet wurde, eine nützliche Hilfestellung, da dort sowohl die aufgerufenen Linux-Befehle als auch deren Rückgabewerte angezeigt werden.

„apply settings“

Sobald die Einstellungen zur Größe des Arbeitsspeichers und der Geschwindigkeit des Prozessors vorgenommen worden sind (oder alternativ ausgewählt worden ist, dass Daten aus dem FTA verwendet werden sollen) und die Anzahl der zu generierenden Maschinen angegeben worden ist, muss die Schaltfläche „apply settings“ betätigt werden. Dies bewirkt, dass die vorgenommenen Einstellungen für den weiteren Programmablauf übernommen werden.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings "apply settings: success" bzw. "apply settings: no success".

„create XML-definitions“

Ein Klick auf diese Schaltfläche veranlasst die Anwendung dazu, einen Satz von XML-Definitionen für die virtuellen Maschinen entsprechend der gewünschten Anzahl („machine count“) zu generieren. Voraussetzung hierfür ist, dass zuvor die Schaltfläche „apply settings“ betätigt worden ist.

Es werden Kopien von der Vorlagedatei „VBOX-DefinitionXML.xml“ erstellt und im selben Ordner (machines) abgelegt. Die neuen Dateien tragen die Namen „VM<i>.xml“, wobei i eine fortlaufende Nummer beginnend mit 0 ist.

In den Kopien werden die Werte für die Geschwindigkeit des Prozessors und für die Größe des Arbeitsspeichers automatisch angepasst.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings "create xml definitions: success"bzw. "create xml definitions: no success".

„upload XML-definitions“

Mittels dieser Schaltfläche werden die zuvor erstellten XML-Definitionen auf das Frontend des Clusters hochgeladen. Der Zielort wird bestimmt durch den Wert, der in der Datei „settings.txt“ für den Eintrag „FrontendDirectory“ gesetzt ist. Von diesem Ort aus werden sie dann automatisch durch das entsprechende Steuerungsskript „control<i>.sh“ gewählt, um die i-te virtuelle Maschine zu konfigurieren.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings " upload xml definitions: success" bzw. " upload xml definitions: no success".

„chose job“

Hier wird der Job ausgewählt, welcher im weiteren Verlauf von der Simulation bearbeitet werden soll. Ein Klick auf den Schaltfläche öffnet ein Dialog zur Auswahl des Jobs (es muss sich bei dem Job um ein Bash-Skript handeln).

In der Statuszeile wird angezeigt, welche Datei gewählt worden ist.

„upload job“

Der zuvor gewählte Job wird durch Klicken dieser Schaltfläche auf das Frontend des Clusters hochgeladen. Der Zielort wird bestimmt durch den Wert, der in der Datei „settings.txt“ für den Eintrag „FrontendDirectory“ gesetzt ist.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „upload job: success" bzw. "upload job: no success".

„generate control scripts“

Ein Klick auf diese Schaltfläche veranlasst die Anwendung dazu einen Satz von Steuerungsskripten zu erstellen. Diese werden im Programmverzeichnis von P2PSim im Unterordner „scripts“ abgelegt. Die Skripte steuern den Programmablauf auf dem Rechencluster. Sie werden als Aufgaben an die Rechenknoten verteilt. Hierbei erhält jeder Rechenknoten genau eines dieser Skripte.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „generate control scripts: success" bzw. " generate control scripts: no success".

„upload control scripts“

Die zuvor gewählten Steuerungsskripte werden durch Klicken dieser Schaltfläche auf das Frontend des Clusters hochgeladen. Der Zielort wird bestimmt durch den Wert, der in der Datei „settings.txt“ für den Eintrag „FrontendDirectory“ gesetzt ist. Dies ist notwendig, damit sie über den Torque Resource Manager des Clusters an die Rechenknoten verteilt werden können.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „upload control scripts: success" bzw. "upload control scripts: no success".

„start simulation“

Sobald diese Schaltfläche betätigt wird werden die Steuerungsskripte über den Torque Resource Manager des Clusters an die teilnehmenden Rechenknoten verteilt („qsub“). Wenn die Knoten frei sind, was für einen ordnungsgemäßen Ablauf der Simulation notwendig ist, beginnen sie sofort damit, die Skripte auszuführen.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „start simulation: success" bzw. "start simulation: no success".

„stop simulation“

Diese Schaltfläche veranlasst P2PSim dazu über den Torque Resource Manager des Clusters alle Bearbeitungen der Steuerungsskripte zu beenden (und somit alle Instanzen der virtuellen Maschinen). Hierbei wird der Befehl „qdel“ verwendet, wobei dieser alle Aufträge abbricht (bzw. diese vorzeitig auf den Status „completed“ setzt), die dem entsprechenden Benutzer zugeordnet sind. Hat diese Aktion Erfolg sind im Anschluss keine Rechenknoten des Clusters mehr durch P2PSim belegt.

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „stop simulation: success" bzw. "stop simulation: no success".

“get simulation status”

Diese Funktion ermöglicht es dem Benutzer, den aktuellen Status der Simulation zu ermitteln. Auf der Standardausgabe des Systems werden folgende Informationen ausgegeben:

Die Ausgabe des Befehls „qstat“ für den aktuellen Benutzer (zeigt alle Jobs des TRM an, die dem aktuellen Benutzer zugeordnet sind, wobei es folgende Stati gibt: C = COMPLETE, Q = GEQUEUED, R = RUNNING).

Der Erfolg dieser Aktion wird in der Statuszeile angegeben durch die Strings „get simulation status: success" bzw. " get simulation status: no success".

4.2.4 Simulationsergebnisse

Die Simulation ist erfolgreich abgeschlossen, sobald alle virtuellen Maschinen mit der Bearbeitung ihrer Jobs fertig sind bzw. wenn die virtuellen Maschinen aufgrund von simulierten Ausfällen heruntergefahren worden sind.

Die Ergebnisse der Simulation werden innerhalb der virtuellen Maschine im Ordner „/home/P2PSim/results“ abgelegt. Mittels des Skripts „onStartup.sh“ wird der gesamte Inhalt dieses Ordners per SCP in das Unterverzeichnis „results“ des Arbeitsverzeichnisses auf dem Frontend kopiert.

Um eine Zuordnung der Simulationsergebnisse zu Zeitpunkt und Ort zu ermöglichen wird pro Rechenknoten ein neues Unterverzeichnis angelegt, welches als Namen die Verkettung aus Zeitstempel und Knotenname erhält. Ein Beispiel hierzu ist „2012-05-05compute-0-1“.

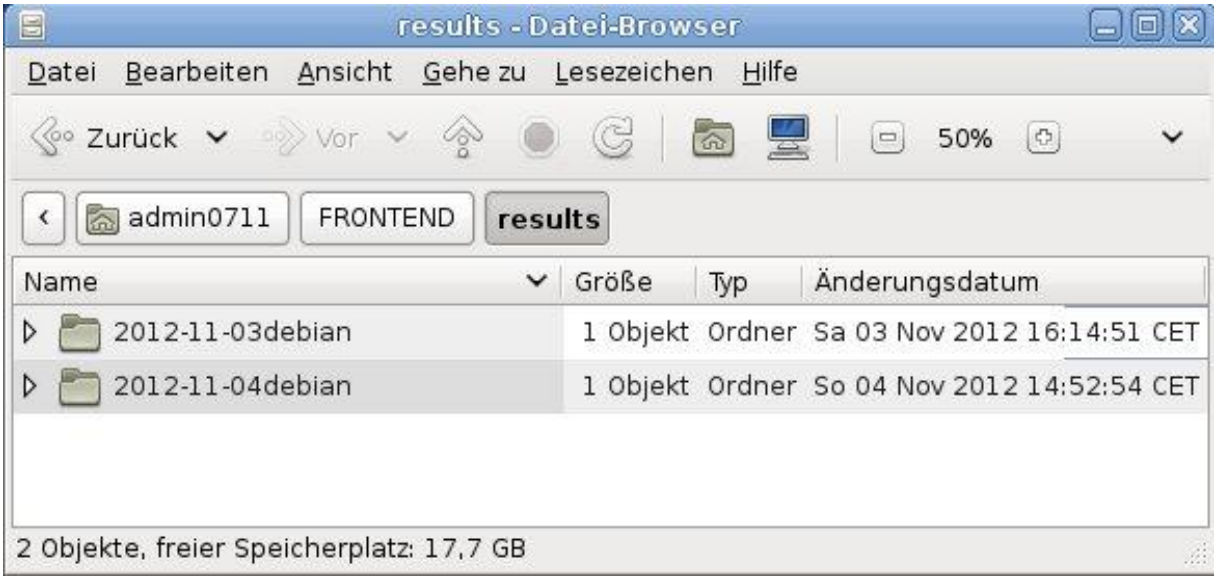


Abbildung 8: Ergebnisse der Simulation

5. Ausblick

Wie bei vielen anderen Softwareprojekten lässt auch P2PSim noch Platz für Verbesserungen und Erweiterungen. Im Folgenden sollen abschließend einige Vorschläge hierzu vorgestellt werden.

Wie bereits in Abschnit 3.2.3 bestehen zwei Möglichkeiten, die Prozessorgeschwindigkeiten, die in der SETI-Datenbank vorliegen, für P2PSim nutzbar zu machen. Die erste Möglichkeit (Verwerfen von Datensätzen, deren äquivalente Prozessorgeschwindigkeit 1000 MHz überschreitet) wurde bereits implementiert. Die zweite Möglichkeit (Skalieren der Datensätze aus der SETI-Datenbank) kann leicht nachträglich noch implementiert werden. Die Daten hierfür liegen in der Tabelle „nodes_complete.db“ im Verzeichnis „databaseFTA“. Ob die zugehörigen Werte für die Größe des Arbeitsspeichers ebenfalls skaliert werden sollten muss allerdings noch evaluiert werden.

Eine Verbesserung in der Benutzerfreundlicheit kann erreicht werden, wenn für die in der GUI verwendeten Schaltflächen eine Enable / Disable – Logik implementiert wird. So sollten Schaltflächen nur aktiviert und damit klickbar sein, wenn alle Vorraussetzungen für die durch sie aufgerufene Funktion zuvor erfüllt worden sind. Somit wird verhindert, dass versehentlich mit Daten gearbeitet wird, die nicht mehr aktuell sind und z.B. aus einer vorangegangenen Benutzung der Software stammen.

Es sollte noch eine Aufräum-Routine implementiert werden. Diese kümmert sich darum, dass nach der Verwendung von P2PSim sowohl auf dem Linux-Client als auch auf dem Cluster (Frontend und Nodes) keinerlei Dateien mehr vorhanden sind. Hierbei sind folgende Dateien automatisiert freizugeben und zu löschen:

- Im Installationsverzeichnis von P2PSim im Unterordner „scripts“ alle temporär erstellten Skripte (control<i>.sh)
- Im Installationsverzeichnis von P2PSim im Unterornder „machine“ alle temporär erstellten XML-Definitionen (VBOX-DefinitionXML<i>.xml)
- Alle Dateien im Benutzerverzeichnis auf dem Frontend des Clusters (in der Datei settings.txt der Eintrag „FrontendDirectory“)
- Alle Dateien in den verwendeten knotenlokalen Verzeichnissen auf dem Cluster (/state/partition1)

Derzeit gibt es in der Datei settings.txt noch den Eintrag „P2PSimHomeDir“. Dieser legt den Installationsort der Anwendung fest, also das Verzeichnis in das das Archiv extrahiert wird. Dieses könnte beispielweise beim Programmstart automatisch ermittelt und der entsprechende Wert in die Datei eingetragen werden.

Literaturverzeichnis

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, Massachusetts 2012, ISBN 0-201-63361-2

Derrick Kondo, Artur Andrzejak, David P. Anderson: On Correlated Availability in Internet-Distributed Systems. 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, Sept 29 - Oct 1 2008

Gian Paolo, Nikos Drakos, Ross Moore: PeerSim HOWTO Build a new protocol for the PeerSim 1.0 simulator. 2005

Ozalp Babaoglu, Toni Binci, Márk Jelasity, and Alberto Montresor: Firefly-inspired heartbeat synchronization in overlay networks. In Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07), Boston, MA, USA, July 2007.

Ingmar Baumgart, Bernhard Heep, Stephan Krause: OverSim A scalable and flexible overlay framework for simulation and real network applications, Proceedings of the 9th International Conference on Peer-to-Peer Computing (IEEE P2P'09), p. 87-88, Seattle, WA, USA, Sep 2009. DOI: 10.1109/P2P.2009.5284505.

Ingmar Baumgart, Bernhard Heep, Stephan Krause**:** OverSim A Flexible Overlay Network Simulation Framework, Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, p. 79-84, Anchorage, AK, USA, May 2007. DOI: 10.1109/GI.2007.4301435.

D. Stingl, C. Groß, J. Rückert, L. Nobach, A. Kovacevic, R. Steinmetz: PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems. HPCS '11, 2011