Cloud Computing: Performance Study in an Eucalyptus Private Cloud

Vijayasanthi M\(^1\), Satyendra Kumar T\(^2\), Jaya Chandra S\(^3\)
\(^1\)Anna University & Sri Perumbudur, Tamilnadu-602105, India.
\(^2\)J.N.T.University & karakambadi road, Tirupati-517507, India.
\(^3\)J.N.T.University & Renigunta road, Tirupati-517507, India.

Abstract—Cloud computing is a new computing paradigm that, just as electricity was firstly generated at home and evolved to be supplied from a few utility providers, aims to transform computing into an utility. It is being forecasted that more and more users will rent computing as a service, moving the processing power and storage to centralized infrastructures rather than located in client hardware. Consumers are not used to renting computing capacity. The question of how to measure performance is already a major issue for cloud computing customers. This thesis demonstrates that current metrics for the performance of offerings by cloud providers are subject to imprecision and variability. This thesis will try to clarify concerns about performance in cloud computing, analyzing the factors that make the performance of clouds unpredictable and suggesting ways to solve this problem. The performance degradation due to virtualization and the lack of isolation between virtual machines were empirically evaluated in an Eucalyptus testbed based on the KVM virtualizer. Drawing upon previous research, all the parts of the problem, from the behaviour of specific application types when hosted in clouds to a proposal for a new generation of SLAs with performance guarantees, will be discussed.

Keywords—private cloud, eucalyptus, KVM, Performance parameters, testbed.

I. INTRODUCTION

As it has been pointed out before, varying performance is among the most worrying characteristics of cloud providers for enterprise customers and as such it has been studied before. Previous studies have focused on studying the performance of public clouds, especially Amazon’s EC2. one of the first benchmarks of Amazon’s cloud, found that there are inconsistencies between the speed of a first request and a second one, and that between 10% and 20% of all queries super decreased performance that were at least 5 times slower than the mean. Tests of Microsoft’s Windows Azure platform show results that indicate that when the number of concurrent clients increases the performance drops.

The greatest degradation is seen in the networking performance, where the variability sometimes makes the TCP bandwidth decrease to a quarter of its mean value. The fact that these previous tests have been done on public clouds is logical as these are the most popular and pure instances of cloud computing. Additionally, the major advantages of the cloud computing new paradigm are associated with such public clouds. However, accurate benchmarking was difficult due to the lack of a controlled environment where the load in the server and network are exactly known at all times. Although these earlier experiments have been very insightful, they lack a higher degree of repeatability. Therefore, in order to identify and pin down with exactitude the effects of resource sharing between virtual machines, experiments must be performed in a controlled environment without background loads other than the load added for the purposes of benchmarking. This is the reasoning behind the choice of a private cloud setup for my test environment. For this research I selected the de facto standard for private clouds: the Eucalyptus open source private cloud.

In the following sections a thorough explanation of the essential software that comprise the whole private cloud used will be presented, along with a practical configuration guide that addresses the problems that arose during the setup process and the necessary steps that must be taken to realize a running cloud.

II. SOFTWARE COMPONENTS

A. Eucalyptus

Eucalyptus was the software platform of choice, but it is not the only private cloud opening today. Similar software can be found from other vendors, among which Open Nebula stands out. Eucalyptus as open-source software, differs from Open Nebula primarily in non-fundamental features for the purpose of this research: more specifically Open Nebula an API to extend the core capabilities and the instruction interface.
On the other hand, this could be one of the best reasons for the adoption of Eucalyptus, as although the API does not enable extension of core capabilities the API used to interact with the cloud is the same as in the Amazon cloud [17], making easier the process of building an hybrid cloud that in low or moderate usage would operate as a private cloud, but that could expand to utilize a public cloud during peaks in load. The choice of Eucalyptus was based on the superior quantity of documentation available, as this greatly eased my learning curve.

The installation package chosen was the Ubuntu version, supplied with the Ubuntu Server 10.04 LTS. This package facilitates the installation of the Eucalyptus’ platform core components, and implements a few add-on features on top of them.

B. Euca2ools

Euca2ools is the open-source version of the set of management utilities and command-line tools used with Amazon’s EC2 [17-18] and S3 services, called Amazon’s EC2 API tools. They implement a large list of image, instance, storage, network, and security management features including:

- Query of availability zones (i.e. called “clusters” in Eucalyptus)
- SSH key management (add, list, delete)
- VM management (start, list, stop, reboot, get console output)
- Security group management
- Volume and snapshot management (attach, list, detach, create, bundle, delete)
- Image management (bundle, upload, register, list, deregister)
- IP address management (allocate, associate, list, release)

C. Hybridfox

Hybridfox is an open source extension for the Mozilla Firefox web browser that helps manage both Amazon EC2 and Eucalyptus user accounts from a single interface. It is an alternative for a cloud user to the command-line tools, and although it also implements administrator tools it does not cover all the functionality of these tools. It was used my experiments as the interface for hypothetic end user running on a Mac OS X environment. The main capabilities of Hybridfox are:

- Creating instances of a VM with a Private IP address.
- Support for Eucalyptus 1.5.x as well as 1.6.x
- Other usability enhancements

D. KVM

Kernel-based Virtual Machine (KVM is a full virtualization solution for Linux. It is based upon CPU virtualization extensions (i.e. extending the set of CPU instructions with new instructions that allow writing simple virtual machine monitors). KVM is a new Linux subsystem (the kernel component of KVM is included in the mainline Linux kernel) that takes advantage of these extensions to add a virtual machine monitor (or hypervisor) capability to Linux. Using KVM, one can create and run multiple virtual machines that will appear as normal Linux processes and are integrated with the rest of the system. It works on the x86 architecture and supports hardware virtualization technologies such as Intel VT-x and AMD-V.

Eucalyptus supports running on either Xen or KVM virtualization. Because Xen (which appeared first in 2007) has been around for longer than KVM, and also is the underlying virtualization system of the biggest cloud vendor, Amazon, there is much more research regarding Xen. This fact and the promising features of KVM’s integration with the Linux kernel caused me to choose to run KVM over Xen in my cloud testbed.

KVM has built in support for live migration, which refers to the ability to migrate a virtual machine from one host to another one without interruption of service. This migration is performed transparently to the end-user, without deactivating network connections or shutting down the applications running in the virtual machine. Detailed information about KVM, along with the performance analysis of the testbed private cloud.

III. EUCALYPTUS MODULES

The Eucalyptus cloud platform is composed of the five software building blocks. Details of this software are described below.

A. Node controller (NC)

An Eucalyptus node is a VT-x enabled server capable of running an hypervisor, in our testbed this was KVM. A Node Controller (NC) runs on each node and controls the life cycle of virtual machine instances running on the node, from the initial execution of an instance to the termination of this instance. Only one NC is needed in each node, and it is responsible for controlling all the virtual machines executing on a single physical machine [10]. The NC interacts with the OS and the hypervisor running on the node, and it interacts with a Cluster Controller (CC).
The NC is also responsible for querying the OS running on the node to discover and map the node’s physical resources (CPU cores, memory size, available disk space) and reporting this data to a CC.

B. Cloud controller (CLC)

The Cloud controller (CLC) is at the top of the hierarchy in a private cloud, and represents the entry point for users to the entire cloud infrastructure. Each Eucalyptus cloud needs one and only one CLC, installed in the physical server that acts as a front-end to the whole infrastructure. It provides an external web services interface, compliant with Amazon’s Web Services’ interfaces, and interacts with the rest of Eucalyptus components on the other side. The CLC is responsible for authenticating users, monitoring instances running in the cloud, deciding in which cluster a requested instance will be allocated, and monitoring the overall availability of resources in the cloud.

C. Storage controller (SC)

The Storage controller provides persistent block storage for use by the instances, in a similar way to Amazon’s Elastic Block Storage (EBS) service. The main purpose of the block storage service is providing the instances with persistent storage. This storage can be used for the creation of a snapshot: capturing a instance’s state at a given moment for later access.

IV. SYSTEM AND NETWORKING CONFIGURATION

A. System design

The latest version of Eucalyptus is prepared to run on very slim resources, for example it can be run on a single physical machine. A single physical machine configuration of Eucalyptus is limited in a lot of ways (e.g. it cannot be used to create isolated virtual networks, therefore being useless for testing network isolation between virtual machine instances. See Table 1), due to the fact that Eucalyptus currently limits the networking modes that can be used in a single machine configuration only to SYSTEM mode. Therefore, although it is not a system requirement, a multiple machine setup is needed to fully test Eucalyptus functionality. For this research a two machine configuration was used:

Front-end server, running the cloud controller, cluster controller, Walrus storage service, and storage controller. Node, running the node controller.

B. Network design

The networking as implemented is outlined in the text that follows. An overview of this network is shown in Figure 1.

Figure 1: Networking outline of the private cloud

The two servers (marked node and front-end in the Figure) were interconnected using an ethernet crossover cable, creating a private network. The front-end had two Ethernet interfaces so that it was able to simultaneously access the private network and the local area network of the lab (this later network will be called the public network). This public network in turn has access to the internet, but it is important to highlight that the node itself does not directly access to the internet [5]. By configuring the routing tables of both machines appropriately, the node directs all of its traffic to and from the internet through the front-end. During the experiments, the clients of the cloud were located generally on the public network. Additionally, we successfully tested the ability of a machine connected to the internet to access the cloud, launch virtual machines, and communicate with them from hosts attached to the internet.

The networking configuration and the resulting connectivity can be quite different depending on which Eucalyptus mode is used. There are four such modes and the main distinguishing feature among them is the level of control and management features overed to the cloud administrator. In increasing order of configuration features these modes are: SYSTEM, STATIC, MANAGED-NOVLAN, and MANAGED. Only the two most feature rich modes, MANAGED and MANAGED-NOVLAN, were used in the experiments [6].
Additional information about both of these modes is given in the following sections, as their details are important in order to understand the experiments. It is also worth saying that, although this four level classification is inherent and particular to Eucalyptus, most cloud platforms over very similar management techniques and methods.

### Table 1
**Eucalyptus Networking Modes**

<table>
<thead>
<tr>
<th>Networking Type</th>
<th>DCHP server running on the network?</th>
<th>CC runs own DCHP server?</th>
<th>Instance isolation</th>
<th>Private IPs</th>
<th>Ingress filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>Required</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STATIC</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MANAGED NOVLAN</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MANAGED</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

V. **Testing Performance Isolation In Cloud Computing**

This section presents the results along with introducing the methodology used to test the performance isolation capability of an KVM-powered Eucalyptus private cloud. The main issue is the impact on the performance of a particular virtual machine instance when another VM is making intensive use of one or more physical, and therefore shared, resources. The physical resources being considered are the CPU, memory, disk, and network interface. For each of these, the results of an experiment will be presented followed by a discussion of the feasible causes for these results, supported by additional references to previous studies when necessary. As it has been said earlier, a customized EMI was created to run these tests. This EMI was composed of a clean installation of Ubuntu Server 10.04 and a number of benchmarking and testing programs. In the following subsections, a brief explanation of how each of these testing programs works will be given. If not expressed different, every experiment was performed with two running virtual machine instances of the customized EMI, with: 15 GB disk, 512 MB RAM, and 1 CPU core.

### A. CPU test

To test the effects of processor interference due to background load (i.e. another instance resident on the same server), if any, it should be analysed how each instance uses the processor. This is a consequence of how Eucalyptus assigns the processor to each instance. In order to learn this, some sort of practical test must be performed as there is not any reference to processor allocation techniques in the Eucalyptus documentation. A simple program in C was created to find out whether there is processor sharing between instances or not. This program consists of a while loop that calls the function gettimeofday() and saves the output in a text file. By graphing the timestamps collected one should see a continuous increasing line. Otherwise, if there is any processor sharing, some gaps will appear along the graph. In this case the result shows no gaps, so it must be assumed that each instance has a single CPU core available all of the time.

To test the CPU performance without involving other resources, the benchmark must be designed to be strictly CPU-bound, without disk or memory accesses. Therefore, an arithmetical only task is the perfect fit for this kind of test. There is a well known example of this: the calculation of digits of the number Pi. Measuring the completion time for a specific number of digits will provide direct information about the performance of the processor. With the test instance, that will be called from now on the well behaving instance, idle otherwise, the benchmark is run 100 times, each time calculating 524288 digits of the number Pi and obtaining 100 samples of the completion time.

Afterwards, the whole process is repeated but with a co-located instance running a CPU intensive task, modelled using the program “Lookbusy” [13]. This is an application that can generate and maintain for a desired amount of time a predictable fixed amount of load on a CPU. Lookbusy is also a strictly CPU-bound task but, to ensure it actually is, a simple verification is done: we measure the completion time of a run of the program. It can be observed that the real time equals the user time, which means that there was no I/O involved in the process. Given this check, the program is set to generate a 100% CPU load. This will use all of the processor resources allocated to the background instance, the “misbehaving” instance, to its maximum. Following this, another 100 samples of the completion time for the calculation of the number Pi to the same number of digits as before are collected and compared to the first set of samples. The results can be graphed in Figure 2.
As shown in Figure 2 and summarized in Table 2, the mean completion time for the idle background case reports that a vast majority of samples are grouped around 1720 ms. Whereas, with a misbehaving processor hogging the CPU resources operating in the same server, this mean time increases slightly. The absolute increase, in mean, does not suppose a heavy worsening of the performance. What is remarkable though, is how the variability increases.

As can be better seen in the graph: there are several outliers, a great number of outliers for a set of samples of this size, leading to a standard deviation eight times greater than the original. This result is significant because, as noted earlier, each instance receives an entire core.

This is not always achievable, depending on the available physical processor resources. The amount of processor assigned to a VM is called the Virtual CPU (VCPU).

In this experiment each VCPU equals exactly one physical core, without sharing. However, it is common to have a VCPU that is half or less than one core of a multiple core CPU. Given the measurements from this experiment, it is reasonable presuming that when processor sharing exists the jitter would be far higher.

The memory performance stress test is based upon a bandwidth test, as this is what distinguishes between types of memories. To measure the memory bandwidth the STREAM memory benchmark has been used, following recommendations of previous studies in this area [4]. In this measurement a data set of 100 samples, from 100 consecutive runs of the benchmark, were collected. Every run consists of 10 trials selecting the best result, which directly translates into the sample.

The size of the RAM memory allocated to both instances is 512 MBytes. In this case the misbehaving instance simulates a RAM intensive task of another cloud user’s application resident in the same server.

This is modelled also using the STREAM memory benchmark, although the data is always gathered from the well-behaving instance. This is shown in Figure 3 and summarized in Table 3.

As can be extracted from the results, this problem has a similar result as the case of the processor interference. There is again a decrease in the average throughput of the memory, in this case of less than 5%; which in most cases would not be a determinant factor. A bigger worry is the points far from the mean. The standard deviation, in presence of a second virtual machine instance, is more than 20 times (or more than 9 percentage points) higher [15].

<table>
<thead>
<tr>
<th></th>
<th>1 VM</th>
<th>2 VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean (ms)</td>
<td>1716.48</td>
<td>1754.90</td>
</tr>
<tr>
<td>std dev (ms)</td>
<td>5.39</td>
<td>41.47</td>
</tr>
<tr>
<td>std dev (%)</td>
<td>0.31</td>
<td>2.36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1 VM</th>
<th>2 VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean (MB/s)</td>
<td>3711.70</td>
<td>3540.68</td>
</tr>
<tr>
<td>std dev (MB/s)</td>
<td>13.83</td>
<td>321.69</td>
</tr>
<tr>
<td>std dev (%)</td>
<td>0.37</td>
<td>9.09</td>
</tr>
</tbody>
</table>

As shown in Figure 2 and summarized in Table 2, the mean completion time for the idle background case reports that a vast majority of samples are grouped around 1720 ms. Whereas, with a misbehaving processor hogging the CPU resources operating in the same server, this mean time increases slightly. The absolute increase, in mean, does not suppose a heavy worsening of the performance. What is remarkable though, is how the variability increases.

As can be better seen in the graph: there are several outliers, a great number of outliers for a set of samples of this size, leading to a standard deviation eight times greater than the original. This result is significant because, as noted earlier, each instance receives an entire core.

This is not always achievable, depending on the available physical processor resources. The amount of processor assigned to a VM is called the Virtual CPU (VCPU).

In this experiment each VCPU equals exactly one physical core, without sharing. However, it is common to have a VCPU that is half or less than one core of a multiple core CPU. Given the measurements from this experiment, it is reasonable presuming that when processor sharing exists the jitter would be far higher.

The memory performance stress test is based upon a bandwidth test, as this is what distinguishes between types of memories. To measure the memory bandwidth the STREAM memory benchmark has been used, following recommendations of previous studies in this area [4]. In this measurement a data set of 100 samples, from 100 consecutive runs of the benchmark, were collected. Every run consists of 10 trials selecting the best result, which directly translates into the sample.

The size of the RAM memory allocated to both instances is 512 MBytes. In this case the misbehaving instance simulates a RAM intensive task of another cloud user’s application resident in the same server.

This is modelled also using the STREAM memory benchmark, although the data is always gathered from the well-behaving instance. This is shown in Figure 3 and summarized in Table 3.

As can be extracted from the results, this problem has a similar result as the case of the processor interference. There is again a decrease in the average throughput of the memory, in this case of less than 5%; which in most cases would not be a determinant factor. A bigger worry is the points far from the mean. The standard deviation, in presence of a second virtual machine instance, is more than 20 times (or more than 9 percentage points) higher [15].
This fact could be seen as a much worse tolerance of the memory than the processor to a background load. Because, while it is true that the glitches in memory throughput reach very low values, they go to even half of the mean bandwidth, they happen much more rarely than for the processor.

Currently, memory management techniques that apply to KVM-based virtual machines are the same as those the Linux kernel that applies to any other process. The memory of a virtual machine is stored in memory as any other Linux process, managed by the standard Linux memory allocator. This is due to the fact that KVM is a Linux kernel extension. The kernel has a feature called Kernel Same-page Merging (KSM), that scans the memory of each virtual machine, detects where virtual machines have identical memory pages and merges these into a single page shared between the virtual machines. This results in space saving given as it allows storing only a single copy of each page. If at any time a guest wants to change this shared page it will be provided with a private copy of its own. Due to the characteristics of the test, with the misbehaving instance running exactly the same benchmark as the well-behaving one, it is assumable that there can be a significant number of replicated memory pages. Therefore there is a chance that the KSM feature may be interceding frequently, in turn could leading to lower memory throughput. This is only a hypothesis, and should be proved by rerunning this benchmark with a different memory load generator for the misbehaving instance, but it was not given much importance because of the relatively low number of outliers that appeared in this test.

C. Network test

Having seen the disk I/O interference problems, it is expected to find similar issues in the process of sharing another resource: the network adapter. What would be fitting to the purpose of cloud computing is a network sharing methodology that divides the available bandwidth and provides network isolation maintaining network latency values when multiple VMs share the link.

To test the network isolation capability announced by Eucalyptus, the system was switched to the MANAGED mode. Now, Eucalyptus creates a virtual network for every new instance of VM launched, the same way as described in previous sections. The VM instances maintain connectivity via the LAN through bridges that share the network interface of the node.

The network bandwidth measurements were performed using Iperf [66], an open source network testing tool that can create TCP and UDP data streams and measure the throughput of the network that is carrying them.

It implements a client and server scheme to measure the network performance between two ends. Experimentally, it is easily checked that the KVM virtualized system has no built-in system of bandwidth fair-sharing between VMs: every time concurrent TCP connections to clients in the LAN are started from two VMs, each of them gets a different share of the link bandwidth and has the ability to starve the other depending on which connection begins first. Besides KVM, the most popular hypervisor nowadays and the one being used by Amazon EC2[20], Xen, does not feature any bandwidth sharing mechanism either. Therefore, in current clouds, the network sharing system must be implemented by the network administrator. However, Eucalyptus does make it easier to implement some sort of solution thanks to its VLAN separation of VMs. One alternative is to create a traffic shaping mechanism taking advantage of this Eucalyptus’ feature, using the Linux traffic control tool (tc) [3].

The Linux traffic control tool allows attaching queuing disciplines, called “qdisc”, to the network devices. These qdiscs can either have multiple child classes or not, in which case they have only a single parent class. These classes form a system of channels or flows in which the outgoing packets are classified, and over each flow some rules to limit the bandwidth or other parameters, such as priority, can be specified. Each class, in turn, has a default queuing discipline that can be changed. Generally, the tc tool filters are used to classify packets among all the flows but, as in the system of this research Eucalyptus has already facilitated this by separating different VMs in different VLANs, hence no filters will be needed. The queuing discipline chosen is the Hierarchical Token Bucket (HTB) [20], whose design suits this experiment since it facilitates guaranteeing bandwidth to classes. This is only for the outgoing traffic as the incoming traffic utilizes another methodology called ingress filtering or policing (i.e. basically dropping packets when the number of them received exceeds the bandwidth specified in the filter). To check the current traffic control discipline of one of the virtual networks using tc you use the command:

tc -s -d qdisc show dev vnet0

A short script was created to configure each virtual interface shaping and policing discipline. This script is:

#!/bin/bash
DEV=vnet0
#outgoing traffic
sudo tc qdisc add dev $DEV root handle 1: htb default 12
sudo tc class add dev $DEV parent 1: classid 1:12 htb rate 47mbit #incoming traffic
sudo tc qdisc add dev $DEV handle ffff: ingress
sudo tc filter add dev $DEV parent ffff: protocol ip prio 50 u32 match ip / src 0.0.0.0/0 police rate 50mbit burst 10k drop flowid :1

This script sets the outgoing bandwidth share of one VM instance to 47 Mbps, and the incoming share to 50 Mbps. These values where found experimentally by choosing the highest possible value that did not cause packet loss when testing with iperf in UDP mode (that is because in TCP mode Iperf does not allow choosing the bandwidth used in each test, instead by choosing the highest value possible, without showing the packet loss results). In practice, this values turn out to be 45 Mbps both in the uplink and in the downlink. Before moving on, some testing was done to be sure that the traffic shaping setup works. Starting simultaneous connections from two VMs shows that now each one obtains its fair-share of bandwidth every time. Therefore, traffic shaping solves the problem of limited bandwidth while leading to fair sharing pretty well.

Although bandwidth is now being fairly divided, the isolation of latency and packet loss remain to be proved. To test the network performance isolation, two clients located on the laboratory LAN (the external network of the cloud) were used. One of them was connected with the well-behaving VM instance and the other one with the misbehaving one. Since the goal was to examine both transmitting and receiving by the server, to see if the traffic shaping could enforce the desired isolation, two types of stress test were designed. In the first, the misbehaving VM acts as the sender while in the second one it acts as the receiver. Either when it is sending or when the client that connects with it sends, the background traffic load is created as concurrent UDP connections at different data rates.

Initially, a large set of baseline round-trip time (RTT) measurements between a client and a VM were collected by sending 9000 pings in three sets of 3000 with an idle interval of several minutes between each set. This operation was repeated with the misbehaving VM acting as if it hosted a network intensive application, in two phases: first sending and then receiving. The methodology of the stress latency test is to collect RTT measurements between one client and the well-behaving VM while another client is connected to the misbehaving one.

For both the sending and receiving stress tests the latency results have average values that grow in direct proportion to the data rate of the misbehaving connection (see table 4). Although the increase is not very significant the fact that there is any difference given that the bandwidth is being fairly-shared correctly was surprising. What is more worrying is that the maximum RTT values of the baseline measurements are nearly the same in every set of samples collected, but, when there is background network activity, there are consistently higher maximum values that reach as much as ten times the baseline maximum value. This can be seen in the deviation of the ping times (see table 4): while the deviations were the same for the three sets of samples in the case of the baseline measurement, the deviations were more than double this value for every set of samples collected with background network activity.

<table>
<thead>
<tr>
<th>SENDER (Mbps)</th>
<th>avg (ms)</th>
<th>max (ms)</th>
<th>dev (ms)</th>
<th>RECEIVER (Mbps)</th>
<th>avg (ms)</th>
<th>max (ms)</th>
<th>dev (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0,688</td>
<td>0,792</td>
<td>0,044</td>
<td>baseline</td>
<td>0,688</td>
<td>0,792</td>
<td>0,044</td>
</tr>
<tr>
<td>baseline</td>
<td>0,687</td>
<td>0,788</td>
<td>0,043</td>
<td>baseline</td>
<td>0,687</td>
<td>0,788</td>
<td>0,043</td>
</tr>
<tr>
<td>40</td>
<td>0,847</td>
<td>1,933</td>
<td>0,093</td>
<td>40</td>
<td>0,796</td>
<td>2,601</td>
<td>0,129</td>
</tr>
<tr>
<td>40</td>
<td>0,851</td>
<td>2,870</td>
<td>0,102</td>
<td>40</td>
<td>0,799</td>
<td>6,455</td>
<td>0,111</td>
</tr>
<tr>
<td>40</td>
<td>0,847</td>
<td>1,112</td>
<td>0,101</td>
<td>40</td>
<td>0,798</td>
<td>2,002</td>
<td>0,116</td>
</tr>
<tr>
<td>45</td>
<td>0,924</td>
<td>6,804</td>
<td>0,220</td>
<td>45</td>
<td>0,823</td>
<td>3,292</td>
<td>0,126</td>
</tr>
<tr>
<td>45</td>
<td>0,900</td>
<td>4,772</td>
<td>0,167</td>
<td>45</td>
<td>0,828</td>
<td>3,213</td>
<td>0,219</td>
</tr>
<tr>
<td>45</td>
<td>0,919</td>
<td>1,375</td>
<td>0,099</td>
<td>45</td>
<td>0,824</td>
<td>8,103</td>
<td>0,117</td>
</tr>
</tbody>
</table>

Since this test showed the variation in the RTT that appears when there is a co-located network intensive VM, further experiments in this direction were performed. Using Iperf, a connection to the well-behaving machine was maintained while the other VM sent or received flows of UDP packets, packet loss and jitter results were obtained. The test involves incoming and out coming packets and during each part of the experiment both VMs were transmitting in the same direction as this is the case of most interference. Figure. 4 depicts the amount of jitter (which refers to the variability of the packet latency across a network) for various combinations of data rates of both VMs.
Overall, jitter was quite proportional to the throughput of the competing VM. Although increasing slightly, jitter remains low when the competing VM is sending or receiving at a fraction of its fair-share of bandwidth, but when it uses all of its share, the jitter doubles in the outgoing case and is almost two orders of magnitude higher in the incoming case. Both cases show levels of jitters above what is normal for a small ethernet LAN [12]. Nonetheless, the fact that the incoming case reaches 2.5 ms of jitter at 40 Mbps is clearly an important issue.

The methodology was based on finding the factors which the performance of cloud-hosted applications depend on. Firstly, the impact on performance of a virtualized and shared physical server was tested throughout the implementation of a private cloud. This experiment, in contrast to similar benchmarks found in the literature about the topic, was executed in a controlled environment rather than in a public cloud, where setup options are limited and background load is generally difficult to know and uncontrollable and therefore allowed drawing some clear conclusions. The analysis of these measurements yielded the following conclusions:

- There is always a performance decrease due to co-located virtual machines running resource-intensive tasks.
- The drop in performance is slight for CPU and memory intensive workload and very significant for disk and network I/O intensive workloads.
- Variability of the measurements increases dramatically in every case when there is high background load.
- Network performance isolation is non-existent in the KVM virtualizer nor implemented in the Eucalyptus platform.
- New disk scheduling mechanisms for virtualized environments should be developed because traditional schedulers, which are currently in use, were not designed to suit virtualized operation.
- Alternatively, fairness in access to disk could be implemented by using network-attached storage.

The overall conclusion of this thesis is that cloud computing is in general prepared to successfully host most typical web applications and with great cost savings, but those applications with strict latency requirements or other network performance requirements, those that require working with large datasets, or those whose needs for availability are critical need to be studied carefully. In these cases, consideration of specific performance requirements, different for each type, and compensation models for violation of the SLA is crucial. Even with the evolution of SLAs, all-purpose clouds might not be willing to other guarantees for the most demanding applications, thus there will be an opportunity for clouds targeted at and designed for particular workloads.

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.
REFERENCES