University of Amsterdam

Faculty of Physics, Mathematics and Informatics

MSc System and Network Engineering

Research Project 2

Collecting, cataloguing and searching performance information of Cloud resources.

February 17, 2017

Author:
Olaf Elzinga

Supervisors:
dr. ir. Arie Taal
dr. Zhiming Zhao

Abstract

When deploying an application in the cloud, a developer often wants to know which of the wide variety of cloud resources is best to use. Most cloud providers only provide static information about different cloud resources which is often not enough because static information does not take into account the hardware and software that is being used or the policy that has been applied by the cloud provider. Therefore, dynamic benchmarking of cloud resources is needed to find out how a certain workload load is going to behave on a certain instance. However, benchmarking various cloud resources is a time consuming process. Thus, using a tool which automatically benchmarks various cloud resources will be of great use. To maximize the effectiveness of such a tool, it will be helpful to maintain an up to date cloud information catalogue, so that users can share and compare their benchmark results to the results of other users. In this paper we present the Cloud Performance Collector, a modular cloud benchmarking tool aimed to automatically benchmark a wide variety of applications. To demonstrate the benefit of the tool we did three experiments with three synthetic benchmark applications and one real-world application using the ExoGENI testbed. During the experiments we focused on measuring variation in performance when a new VM is provisioned and when the same VM is used over a longer period of time. We found out that most ExoGENI instances perform very stable over time, however there can be some difference in performance when a new VM instance is provisioned.

Contents

1 Introduction

Over the last decade, the usage of cloud computing is becoming increasingly popular. With the increasing amount of available instances and cloud providers it is becoming increasingly difficult for application developers to select the right cloud provider for their application. Most cloud providers provide static information (e.g. CPU cores, memory size, disk size, and disk type) of different kinds of virtual machines (VMs). However, when an application developer wants to deploy a mission-critical application in the cloud, the static information provided by the cloud provider is often insufficient, because static information does not take into account the hardware and software that is being used or the policy that has been applied by the cloud provider. Therefore, more precise information about cloud resource types and provisioning constraints is crucial to successfully deploy an application within the cloud. To help cloud customers find precise information about cloud resources, automated benchmarking tools would be of great use. Over the last few years many automated benchmark tools are proposed in literature, including [4, 7, 21, 18, 9, 12]. All proposed tools aim to help a single user to benchmark multiple instances and/or providers, so that the user is able to select the right instance according to their requirements. However, the performance of those instances may be different each time it is measured [8, 14]. Thus, it would be helpful if users can share their benchmark results and compare them with results of other users to get a better understanding of variation in performance.

In this paper we will look at how to test a given application component of cloud resources to maintain a cloud catalogue. By using the cloud information catalogue it would be possible for users to share and compare the results with the community.

The main research question is: '*How to test a given application component of cloud resources to maintain a cloud catalogue of dynamic cloud performance information?*'

- What are the existing methods for obtaining cloud resource performance information?
- How to benchmark any given application component in an easy way?

The outline of this paper is as follows. In section 2, we will review the state of the art for cloud performance collecting. Next, in section 3 we discuss the proposed Cloud Performance Catalogue. In section 4, we describes the experimental setup and give the results of the experiments. In section 5 the results of the experiments are discussed. Lastly, section 6 concludes the paper and offers avenues for future research.

2 State of the art review

In this section, we first specify the most important requirements for a tool that automatically collects cloud performance information which can be used to maintain a cloud catalogue. After that, we specify the requirement for reviewing automatic cloud benchmark tools proposed in literature. Where after, we decide if we are going to use an existing automatic cloud benchmark tool or build our own tool to answer the main research question.

2.1 Tool requirements

Out of the research questions and challenges described in section 1 we can enumerate several functional requirements:

1. **Custom benchmarking:** implementing new and custom benchmarks is an important feature of the tool. By using a well-defined way of specifying which application needs to be installed, configured and executed, it should be easy to implement any type of application to benchmark it on different cloud resources/providers.
2. **Scheduling benchmarks:** besides that it must be possible to directly execute a benchmark, it also must be possible to schedule benchmarks in order to do long-term analyses of cloud resources.
3. **Catalogue results:** The results of each benchmark that is collected must be stored into a cloud information catalogue.
4. **Searching and comparing results:** to make full use of the potential of the cloud information catalogue, it must be easy to search and compare information inside the catalogue. Both searching and comparing information can be done by either human (web-interface) or machines (API). The two different types of actor both have different ways of interacting with the catalogue to gain their need of information.

The requirements for selecting an automatic benchmark tool proposed in literature are:

1. **Publicly available**: most importantly, the application must be publicly available for usage.
2. **Open-source**: in order to make the community contribute to the tool as well, the application must be open-source. When applications are open-source and downloadable via for example Github, it helps to get the community involved with the project. The community can report bugs and make suggestions for new features which helps the applications stay secure and remain relevant.
3. **Maintainability**: an important aspect of the application is that it must be maintained frequently. The cloud evolves rapidly and new features are presented regularly and therefore it is important that the automated benchmark tools follow the new trends of the cloud and keep helping customers to select the right cloud resources.
4. **Support for IaaS providers**: in many cases users will compare multiple providers to select the best offer according to their wishes. Therefore, the application must support a

large amount of public and private IaaS providers. Without supporting a large amount of public IaaS providers, users will not see the relevance of the tool.

2.2 Automated benchmark tools proposed in literature

Over the last few years there are several automated cloud benchmarking tools proposed in literature.

Chhetri et al. [4] proposed the Smart CloudBench, which is a platform that automates the performance benchmarking of cloud infrastructure, helping potential consumers quickly identify the cloud providers that can deliver the most appropriate price/performance levels to meet their specific requirements. They looked at benchmarking from the consumer's perspective and focused on benchmarking the entire application stack instead of looking at individual components (e.g. CPU, RAM, Disk or Network performance).

In 2013, Cunha et al. [6, 7] proposed an automatic benchmark tool called the Cloud Crawler. The tool helps users to describe and automatically execute application performance test inside the cloud. New benchmarks are defined in a declarative domain-specific language called Crawl, which is based on YAML. The benchmarks defined by Crawl are executed through the execution engine called Crawler, which is responsible for automatically configuring executing and collecting the results of each evolution. To test the benefits of Cloud Crawler the authors did an experimental evaluation of social network applications in Amazon EC2 and Rackspace with different VM configurations and under different demand levels. An old version of Cloud Crawler is available on Github [5].

In 2014, the authors of [18, 19] presented the Cloud WorkBench (CWB), a concrete implementation of a cloud benchmarking Web services. CWB was designed and implemented to leverage the notion of IaC for cloud benchmarking, and is used to automate the benchmarking lifecycle from the definition to the execution of benchmarks[19]. CWB uses Vagrant to provision virtual resources and Opscode Chef to install and configure the benchmark tools. CWB can run benchmarks directly or schedule benchmarks within various public Infrastructure as a Service (IaaS) clouds. The latest version of Cloud Crawler can be downloaded from Github [17] and is licensed under the Apache License Version 2.0 (APLv2) [17].

Jayasinghe et al.[9] proposed Expertus, which is a flexible code generation framework for automated performance testing of distrusted applications in IaaS clouds. Expertus uses XML and XSLT templates for new benchmarks.

In 2013, Silva et al. presented the CloudBench [21]. CloudBench is an open-source framework that automates IaaS clouds to run controlled experiments, where complex applications are automatically deployed. The authors demonstrated CloudBench main characteristic trough the evaluation of an OpenStack installation, including experiments with approximately 1200 simultaneous VMs at an arrival rate of up to 400 VMs/hour [21]. CloudBench can be downloaded from Github [20] and is licensed under the Apache License Version 2.0 (APLv2) [20].

In 2015, Kratzke et al. [12] proposed EasyCompare. EasyCompare is not about selecting the best cloud provider but it is about selecting the most similar resource provided by different cloud providers. EasyCompare uses Euclidian distance measure to compare the different cloud

resources. To demonstrate EasyCompare, the authors evaluated the resources of two major public cloud providers namely Amazon EC2 and Google Compute Engine.

Table 1: Comparison of proposed automated benchmark tools and our requirements

	Publicly available	Open source	Custom benchmarks	Schedule	Provider support	Catalogue result
Cloud WorkBench	Yes	Yes (Apache 2.0)	Yes	Yes	Only public	No
Cloud Crawler	Yes	Yes	Yes	No	Only public	No
CloudBench	Yes	Yes (Apache 2.0)	No	No	Public and private	No
EasyCompare	No	-	No	No	Only public	No
Expertus	No	-	Yes	No	Only public	No
Smart CloudBench	No	-	No	Yes	Only public	No

2.3 Technical gaps

Table 1 compares the most important requirements (see section 2.1) of the tools proposed in literature described in the last section. None of the tools proposed in literature met our requirements, therefore we decided to create our own tool during this research. We identify a number of technical gaps that we try to bridge in this research:

1. **Catalogue the collected results**: None of the tools proposed in literature have the ability to be used with a cloud catalogue. Most related work focuses on collecting performance information and are using the collected data for performance evaluation research without looking at the possibility to catalogue the collected results to make it possible to share and compare data of others using that tool.

2. **Ability to add providers**: None of the tools have the ability to easily add providers to the tool. For example, the Cloud WorkBench uses Vagrant, which works well for the platforms Vagrant supports. However, when a provider is not supported by Vagrant one has to find an other way to add that provider. It would be helpful if a new provider could be implemented regardless of the type of software used to do so. Therefore, a well defined framework will be of great use to define a standard way of writing such a piece of code that controls the orchestration VMs.

3. **Possibility to add custom benchmarks**: Most of the tools proposed in literature do not provide a way to add custom benchmarks in an easy way. Therefore, it's important that the installation, configuration and the benchmarking process are defined in a powerful way and in a common language (e.g. JSON, YAML) so that it's easy for users to benchmark their scenario.

3 Cloud Performance Collector

In this section we will discuss the architecture of the Cloud Performance Collector (CPC). The relation between the CPC and user/cloud catalogue. After which, we will discuss how the CPC works and discuss the prototype we build.

3.1 System overview

Figure 1 illustrates the process of collecting, cataloguing and searching performance information of Cloud resources. Collecting the performance information is done by the CPC. Via the CPC, users can directly run or schedule benchmarks. When a benchmark is executed, the CPC will take care of the whole process; the CPC will provision the VM, install and configure the necessary software, run all the benchmarks and collect the results. When all benchmarks are finished, the VM will be released and the results will be catalogued inside the Cloud Catalogue.

To make it easy for developers to implement new features, the design includes three modules: the provider module, the deploy and run module, and the result module. The *provider module* makes it possible to provision and release VMs when the experiments are finished. The *deploy and run module* takes care of installing, configuring, and executing the benchmarks. The *results module* modules parses all the useful information out of the output of each benchmark application.

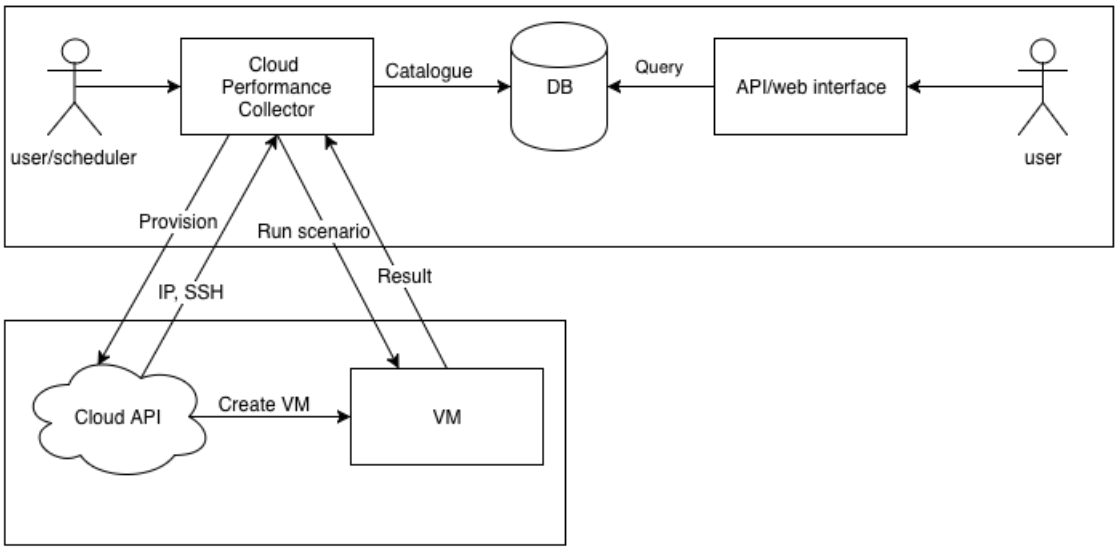


Figure 1: Architecture of the cloud catalogue

3.2 Cloud Performance Collector inner workings

The steps involved in the data collection process are shown in the sequence diagram in figure 2. In the first step, the experimenter (user) runs or schedules one or more benchmark scenarios. When a scenario is executed, the CPC will first provision the necessary resource via the cloud

Application Programming Interface (API). As soon as the VM instance is reachable, the software can be installed and configured depending on the layout of the scenario. After the successful installation/configuration of the software, the benchmarks can be executed. When a benchmark is finished, the results will be collected by the CPC. After all benchmarks are finished and all the results are collected, the CPC will release the VM to keep the time the VM is used to a minimum.

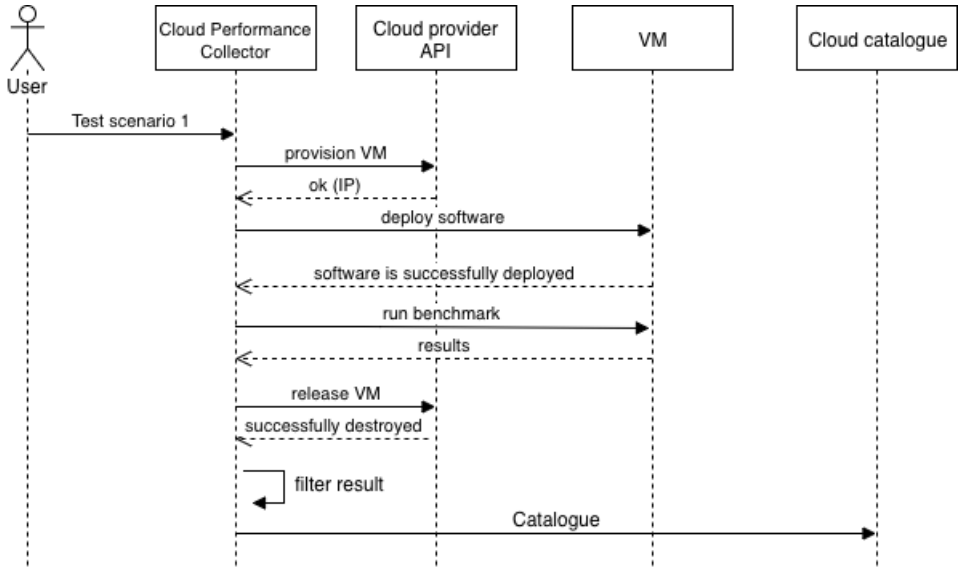


Figure 2: The CPC benchmark execution process

3.3 Prototype

To demonstrate the benefit of the design we build a prototype to do experiments with. The prototype is a command-line interface (CLI) tool written in bash. During the experiments ExoGENI will be used as provider, therefore the *provider module* will make use of the python script omni [1] to communicate with the API of ExoGENI. The *deploy and run module* which takes care of installing, configuring and executing the benchmarks will be done via Ansible [2]. Ansible is an open-source configuration management tool which uses SSH push commands to install, configure and run the necessary benchmarks. The scheduling of benchmarks is done via Linux cronjobs. The *results module* consists out of small bash scripts to filter the output and the catalogue function of the application will not be used during the experiments.

[1] `http://trac.gpolab.bbn.com/gcf/wiki/Omni`
[2] `https://www.ansible.com/`

4 Experiments and results

In order to illustrate the benefits of the CPC we conduct several experiments using the ExoGENI [3] testbed. ExoGENI is, in effect, a widely distributed networked infrastructure-as-a-service (NIaaS) platform geared towards experimentation and computational tasks [3].

During these experiments we aim to answer three questions:

- Will VM instances with the same specifications and image from the same provider give similar performance?
- Will the same VM instance with the same workload provide a constant level of performance over time?
- Will a given application component perform the same compared to the synthetic benchmarks?

The goal of the first question is to measure if there is a difference between different provisioned VMs, using the same specifications and image from the same provider.
The goal of the second question is, to find out whether a VM instance once provisioned performs constantly over a longer period of time. By comparing the results of the first question with the results of the second question we can analyze whether the performance variation is depending on the time of the day or the physical location of the VM instance or both.
The goal of the third and last question is to find out whether the results of the first two questions help to predict the performance of a real-world application. By using a real-world application, we can demonstrate how the CPC can test any given application component.

During the experiments we keep in mind that users are bound to certain time constraints, therefore the whole process from provision an instance until the release of an instance must be within one hour.

4.1 Experimental setup

All experiments were conducted on the ExoGENI testbed using the racks of: The National ICT Australia (NICTA), Raytheon BBN Technologies (BBN), and the University of Amsterdam (UvA). The experiments are conducted on the "current types" offered by ExoGENI [1]. During the experiments we will use three different instance types: XOMedium, XOLarge, and XOXLarge. Table 2 shows the specification of the current resource types offered by ExoGENI. All instances are using the same Ubuntu 14.04 image.

Table 2: Resource types offered by ExoGENI [1]

Resource Type	Resource Name	Cores	RAM	Disk(s)
VM	XOMedium	1	3G	25G
VM	XOLarge	2	6G	50G
VM	XOXLarge	4	12G	75G

4.2 Benchmark applications

During the experiments four applications are used, three synthetic benchmark tools, and one real-world application.

Sysbench [10] is a modular, cross-platform and multi-threaded benchmark tool to quickly get an impression about system performance. During our experiments we use sysbench to do a primality test on 100,000 natural numbers. Sysbench measures the time it takes to calculate those number in seconds. Since sysbench is a multi-threaded benchmark tool, we set the *–num-threads* to the n-thread available on that particular instance. We will use the standard version of sysbench available in the Ubuntu repository without special tuning.

The *STREAM* [15] benchmark is used to measure the performance of the main memory. STREAM is a synthetic benchmark which is designed to measure sustainable memory bandwidth using four vector-based operations:

1. COPY $a = b$
2. SCALE $a = q * b$
3. SUM $a = b + c$
4. TRIAD $a = b + q * c$

During our experiments the TRIAD operation is used. We will use the standard version of STREAM available in the phoronix test suite [13] without special tuning. The phoronix test suite is installed via the default Ubuntu repository.

IOzone [16] is a benchmark used to measure the read and write performance of the disk. To reduce the time it takes to complete both the read and write process, we make use of a file size of 2GB with a record size of 64Kb. We will use the standard version of IOzone available in the phoronix test suite [13] without special tuning.

To demonstrate that any type of application could be tested, we will use the application Montage[2] inside a Docker container. Montage is a toolkit for assembling Flexible Image Transport System (FITS) images into custom mosaics. We will use the dockerfile [11] to build and run Docker container.

Table 3 shows the metrics of all application used during the experiments.

Table 3: Application used during the experiments

Component	Application	Metrics
CPU	Sysbench	Duration (sec)
Memory	STREAM	Throughput MB/s
Disk	IOzone	Throughput MB/s
Application	Montage	Duration (sec)

4.3 Experiment 1: Performance variation of different VM instances

During the first experiment we will try to find out if VM instances with the same specifications and image from the same provider gives similar performance. During these experiment we will use a different VM instance every two measurements. By benchmarking the same instance twice before a new one is used we can see if the variation is caused by the fact that the instance is placed on a different physical server or by noisy neighbours.

4.3.1 CPU

Figure 3 shows the results of running the sysbench CPU benchmark. The instance running on the rack of NICTA are performing quite stable and have little to no performance variation when a different instance is used. In contrast to the instance running on the rack of NICTA, we have measured different performances when a new instance is provisioned on the rack of BBN. However, when we run the same benchmark for the second time on the same VM instance, we see a similar level of performance. Because the second measurement on the instances of the BBN rack show similar performance compared to the first measurement on that instance. It is likely that the instance is placed on a different physical server within the rack.

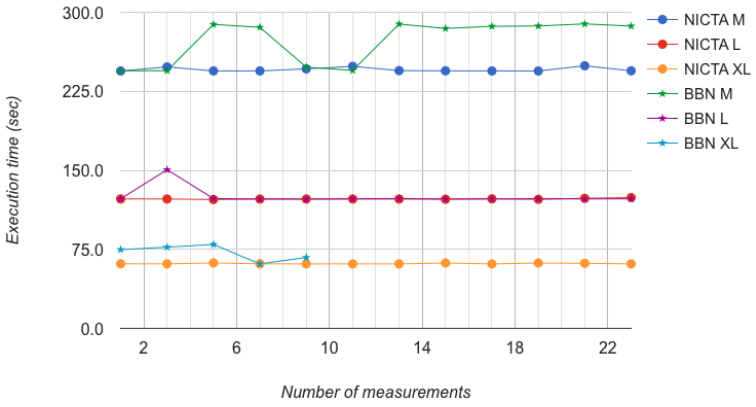


Figure 3: Variation in performance of different VM instances running sysbench

After ten measurements it was not possible to provision the BBN XL instances again. Therefore, during this experiment the data available of the BBN XL instance is limited. Similar problems occurred on provisioning instances on the rack of the UvA. Therefore, we decided to not include the instance of the UvA during this experiment.

4.3.2 Memory

Figure 4 shows the memory throughput measured by STREAM. In general, the results are comparable with the results of the CPU benchmarks with a couple of interesting differences. The Large and XL instance of NICTA show slightly decrease in performance in some measurements. The instance on BBN show the similar pattern compared to the CPU benchmark. Interesting

is that during a measurement during which the memory throughput is higher, the time it takes for sysbench to finish is longer. For example, the first four measurements on both the NICTA Medium and the BBN Medium instance show more or less the same result. During the fifth measurement of the BBN Medium instance, we see an increase in execution time during the CPU benchmark but also an increase in memory throughput.

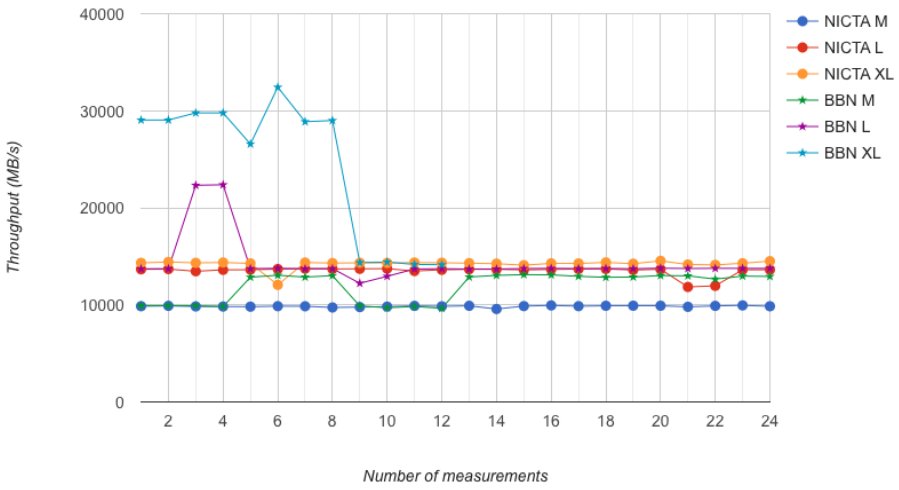
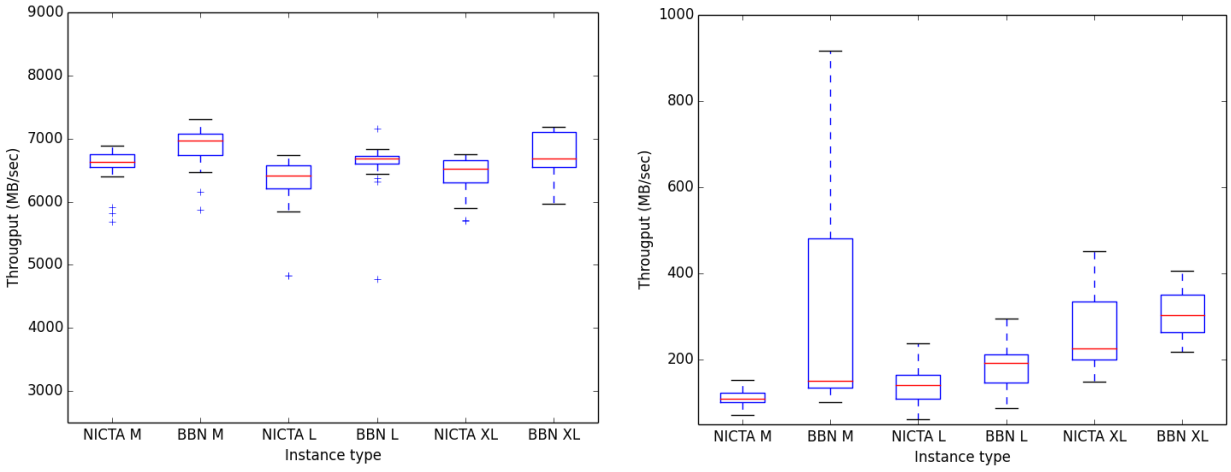


Figure 4: Variation in performance of different VM instances running STREAM



(a) Read (b) Write

Figure 5: Variation in performance of different VM instances running IOzone

4.3.3 Disk I/O

The performance disk I/O has the tendency to vary more compared to CPU and memory. During this experiment we can see this behaviour as well. Figure 5a shows the read performance

of the different instances, all instance perform similar, whereas the instances of BBN perform slightly higher. The write performance is shown in figure 5b. Compared to the read performance, the larger instance tends to perform a little bit better compared to the smaller instances. Interesting is that the BBN Medium instance shows a big variation in performance.

4.3.4 Provision and Benchmark time

Table 4 shows the minimal, maximum, standard deviation (SD), average, and relative standard deviation (RSD) of the time to provision and duration of the benchmarks of all instances during this experiment. For both provisioning and running the benchmarks, the instance on BBN were quicker. The shortest benchmark time measured was 07:02 minutes (BBN XL) and the longest benchmark time was 33:02 minutes (NICTA Medium). With regards to provision time, it took the least time for the BBN Medium instance (01:38 minutes) and the most time for the NICTA Large instance (9:59 minutes).

Table 4: Provision time and benchmark duration (in seconds) of experiment 1

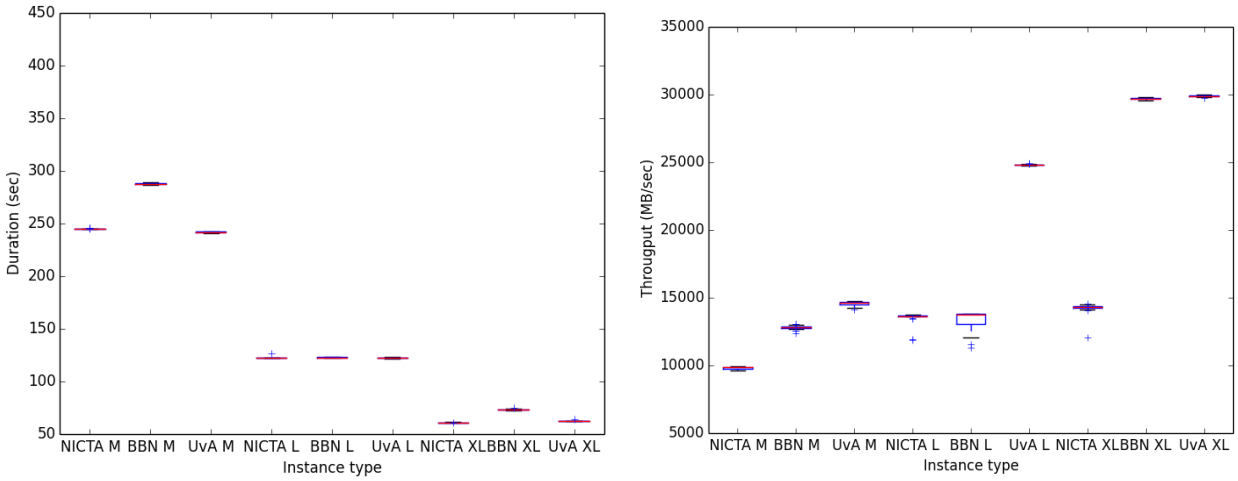
Instance	Provision time					Benchmark time				
	min	max	SD	Avg	RSD	min	max	SD	avg	RSD
Nicta M	130	1190	289	279	104%	1110	1982	227	1295	17.51%
BBN M	98	204	30	126	24%	716	1748	262	1055	24.85%
Nicta L	161	1199	290	284	102%	630	1746	286	919	31.17%
BBN L	100	166	25	124	20%	653	1387	190	815	23.33%
Nicta XL	172	462	78	238	33%	529	1721	263	733	35.93%
BBN XL	120	204	36	164	22%	422	951	155	607	25.64%

4.4 Experiment 2: Performance variation of a single VM instance over time

During the second experiment we will try to find out if the same VM instance with the same workload provide a constant level of performance over time. During the start of this experiment we will provision a VM instance for each provider/resource type and we will use that same VM instance for all measurement.

4.4.1 CPU

Figure 6a shows the results of running the sysbench CPU benchmark. All the measured instances show almost no variance in performance. The Large instance of all three racks perform on the same level. However, the Medium and XL instance of BBN performed less compared to the same instance of the other racks.



(a) Sysbench (b) STREAM

Figure 6: Variation in performance on the same VM instance running sysbench and STREAM

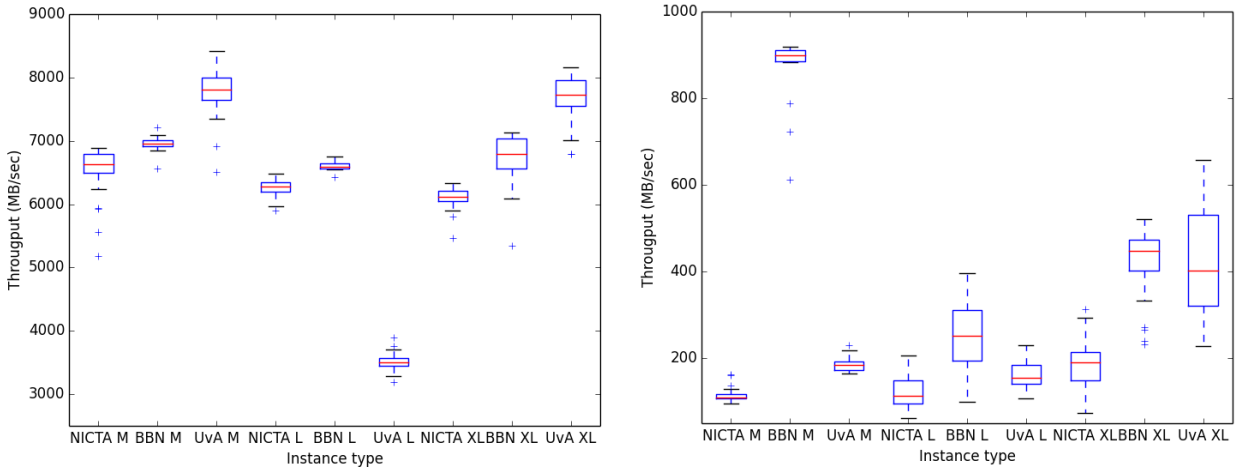
4.4.2 Memory

Figure 6b shows the variety in performance. Where as, the results CPU shows almost no variation in performance, the results of the memory show some small differences. The BBN Large instance shows significant difference and the NICTA Large and NICTA XL show some difference as well.

4.4.3 Disk I/O

The results of the disk I/O are similar to the first experiment. Still there is a lot of performance variation measured on all instances. However, the performance variation on some instances seem to be less compared to the first experiment.

Figure 5a shows the read performance of the different instances. In almost all cases (except for UvA Large) the instance running on the UvA have the highest performance followed by BBN. A possible explanation for the results of the UvA Large instance is the fact that the UvA rack is heavily used (which resulted in provisioning problems during the first experiment). Figure 5b shows the write performance of the different instances. In general, the larger instance tends to perform a little bit better compared to the smaller ones. Just like the first experiment, the write throughput of the BBN Medium is much higher compared to the write throughput of the other instances. However, this time the variation is much lower than during the first experiment.



(a) Read (b) Write

Figure 7: Variation in performance on the same VM instance running IOzone

4.4.4 Benchmark duration

Table 5 shows the minimal, maximum, SD, average, and RSD of the benchmark duration of all instances during this experiment. The fasted measurement was 05:22 minutes (UvA XL) and the slowest measurement 26:28 (NICTA Medium).

Table 5: Benchmark duration (in seconds) of experiment 2

Instance	min	max	SD	Avg	RSD
Nicta M	1177	1588	94	1277	7%
BBN M	727	891	39	763	5%
UvA M	860	982	44	926	5%
Nicta L	634	1458	150	1001	15%
BBN L	670	782	31	727	4%
UvA L	560	714	44	624	7%
Nicta XL	599	1450	222	945	23%
BBN XL	409	609	56	484	12%
UvA XL	322	524	60	416	14%

4.5 Comparing the first two experiments

To get a better understanding of performance variation of the first and second experiments we will compare the results of both experiments. In appendix A four tables can be found which compares the SD, average, and RSD of the first experiment with the SD, average, and RSD of the second experiment.

In table 6, the CPU results of both experiments are compared. With regards to the instance of

NICTA, we can see that the RSD of first experiment is higher but relatively close the RSD of the second experiment. In contrast with the instances on BBN, the RSD of the first experiment is much higher compared to the second experiments.

In table 7 the memory benchmark results of both experiments are compared. The RSD of both experiment on the instances running on NICTA are almost similar, except for the NICTA Medium instance which is slightly lower (0.11%). Similar to the results of the CPU benchmarks, the RSD of the first experiment is much higher compared to the second experiment.

Table 8 and table 9 compare the results of both experiments with regarding to disk I/O. We witnessed that during the comparison of the CPU and memory results, the RSD of the first experiment is almost always higher or equal to the RSD of the second experiment. However, this is not the case for the disk I/O. In 58% of cases the RSD of the first experiment is higher compared to the second experiment and in 42% of the cases the RSD of the first experiment is lower compared to the second experiment.

4.6 Experiment 3: Performance variation of a real-world application

Figure 8 shows results for the Montage application running inside a Docker container. During this experiment we used the same VM instance that is used during the second experiments. Similar to the results of the other experiments, during this experiment we can see that all instances show little performance variation over time. Interesting is that all medium instances are performing similar or better compared to their same provider counterpart.

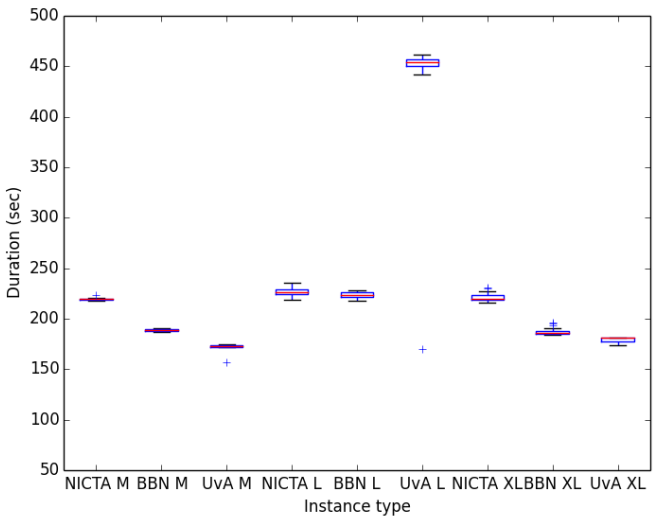


Figure 8: Results of running Montage inside a Docker container

5 Discussion

In this section the experiments are discussed.

5.1 Experiments

Will VM instances with the same specifications and image from the same provider give similar performance?

During our experiments we have seen that when a new VM instance is provisioned the performance can be similar but in some cases can differ. Based on the first experiment we found out that it depends on which ExoGENI rack is used. Our experiments show that NICTA provides similar performance when a new VM instance is provisioned compared to previous provisioned VMs. However, it was not the case for the BBN rack, were we have seen that the performance can vary quite a bit, probably depending on which physical server it's placed.

Will the same VM instance with the same workload provide a constant level of performance over time?

We have seen that with regards of CPU and memory the performance provided by all tested providers is constant over time. Although, during the memory benchmark the performance variation of three out of the nine instances were higher compared to the other six. With regards to the variation of disk performance we have seen that there is much more variation during both experiments. We have seen that the read performance of the UvA Large instance was significantly lower compared to all instances, a possible explanation for the low results is the fact that the UvA rack is heavily used. However, to really understand why the performance was significantly lower more testing is needed to identify if it has something to do with the VM type or the physical server the VM is hosted on. However, this issue perfectly illustrates the relevance of maintaining an up to date cloud catalogue to help users compare their results with the results of others.

Will a given application component perform the same compared to the synthetic benchmarks?

The Montage application read a large amount of images from disk and created one big image out of those images. Therefore, read performance is an important aspect of the application. When we compare the results of this experiment with the result of disk read performance of the second experiments, we can see that figure 7a and figure 8 have a lot of similarities. Both figures show that the UvA is the fasted except for the UvA Large instance. Both figures show that the Large instance of each rack is performing less compared the Medium and XL instance of that rack. Hence, we can conclude that it should be possible to use synthetic benchmark to show which instance is best for a specific application. However, it is important to understand the most important components of the application to compare its results to various results of synthetic benchmarks.

5.2 High disk performances

During the first two experiments we benchmarked the disk performance of various instances. The performance measured during this experiment were extraordinary high, especially the read throughput. An explanation for the extraordinary high result is the fact that we used a small file size to reduce the benchmark time. To goal of the experiment was to benchmark various components of a VM within a single hour, including provisioning, installing, configuring, and releasing the VM. Hence, we could conclude that benchmarking multiple components within an hour does not deliver reliable insight into the performance capability of storage devices. So when one wants to measure the actual performance of the disk (and not the performance of any form of caching available to the instance) it is recommendable to do longer benchmark which automatically increase the costs of benchmarking multiple instances on multiple provider/regions.

Kratzke et al. [12] measured comparable values with IOzone on GCE and AWS EC2. Kratzke even used a smaller file size of 512 MB compared to the 2 GB used during our experiments.

With regards to the cloud performance catalogue, it is important to differentiate between benchmark parameters so that the results of short benchmarks are not compared with the results of longer measurements.

5.3 ExoGENI issues

During the first experiment we encountered some problems with provisioning new VMs on the rack of BBN and the UvA. On the BBN rack it occurred on the XL instance after ten measurements, therefore we still used the data of those ten measurements. With regards to the UvA instance, it was more problematic. It occurred on all three instance two times. The first time we manually provisioned the VM stated benchmarking again. However, after the second measurement the VM is released and a new VM provisioned and again we faced the same problem. When we analysed the debug log file, we saw in all cases that the VMs were successfully provisioned but none of those VMs were reachable afterwards. The next day, we manually provisioned a VM again and found out that the rack did not have the resource to successfully provision the VM (by using the commando: "python omni.py sliverstatus").

We also encountered issue with VMs that became suddenly unavailable. It happened on instances running on the rack of the UvA, during the second experiment. Therefore, we had to create a new slice because in the old one it was not possible to provision a new VM instance for a certain amount of time.

6 Conclusion and future work

In this paper we looked at how to test a given application component of cloud resources to maintain a cloud catalogue of dynamic cloud performance information. We first evaluated existing methods for obtaining cloud resource performance information and found out that none of the proposed cloud benchmark tools are designed to work with a cloud catalogue. To bridge the gaps we found, we proposed the Cloud Performance Collector, a modular cloud benchmarking tool aimed to automatically benchmark a wide variety of applications. To demonstrate the benefit of the tool we did three experiments with three synthetic benchmark applications and one real-world application using the ExoGENI testbed. During the experiments we found out that some ExoGENI racks provide different performances depending on which physical server the VM instance is placed. However, other racks provide more or less the same performance when we provision a new VM with the same image and specifications. We have seen that once a VM instance is provisioned the performance with regards to CPU and memory is quite stable, however, with regards to disk I/O there is more variation in performance measured. We have demonstrated the relevance of the CPC by benchmarking a containerized real-world application which shows that larger instances with more resources does not always have to be the right choice for your application.

The results of this research opens up several avenues for future research.

During this research we evaluated the performance of various components of a single VM. It will be of great use to benchmark a single scenario for multiple VMs. Which can be basic experiments to measure, for example, the network performance between several regions/racks using an application like iperf. Or deploy real-world benchmark like measuring the amount of live stream viewers an instance can handle and then conclude if it is better to use more small instances or a couple of large instances.

We currently have used the CPC to evaluate the performance of ExoGENI. A logical step would be to add the support for various public and private cloud providers to increase the possibilities of the CPC. Furthermore, it would be interesting to add the support of not only traditional VM instances (running on a hypervisor like Xen and KVM) but also container based IaaS instances (running Docker) which are offered by big cloud providers like Amazon [3] and Google [4].

During this research we focused on collecting performance information of cloud resources to maintain an up to date cloud catalogue. Research is needed on how the collected performance information can be used to reliably inform users about the performance of certain instances. Important is that the right performance information is compared to each other to give the user reliable overview. For example, we saw during our experiment that the disk read throughput was extremely high which was probably caused by raid-controller or RAM caching or by using a small file size. Therefore, it is important to know which parameters (e.g. disk size, RAM size, test file size, raid-controller cache size/speed) need to be used during the comparison, to give reliable results. Furthermore, to increase the effectiveness of the cloud catalogue it would be of great use to look into using weights to select which cloud instance fits best for a certain application. For example, the requirement for application X is that it needs a lot of process power, but it does not need disk I/O. Thus, one wants to focus their search on instance with

[3] `https://aws.amazon.com/ecs/`

[4] `https://cloud.google.com/container-engine/`

process power and not on disk I/O. An interesting example of using weights to intelligently rank cloud offers is proposed by Varghese et al. [22, 23].

References

[1] Exogeni resource types. https://wiki.exogeni.net/doku.php?id=public:experimenters:resource_types:start.

[2] Montage project. http://montage.ipac.caltech.edu/.

[3] Welcome to exogeni. http://www.exogeni.net/.

[4] Mohan Baruwal Chhetri, Sergei Chichin, Quoc Bao Vo, and Ryszard Kowalczyk. Smart cloudbench–automated performance benchmarking of the cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 414–421. IEEE, 2013.

[5] M. Cunha. crawler. https://github.com/mathcunha/crawler, 2016.

[6] M Cunha, NC Mendonça, and A Sampaio. Cloud crawler: a declarative performance evaluation environment for infrastructure-as-a-service clouds. *Concurrency and Computation: Practice and Experience*, 29(1), 2017.

[7] Matheus Cunha, Nabor C Mendonça, and Americo Sampaio. A declarative environment for automatic performance evaluation in iaas clouds. *IEEE CLOUD*, 2013:285–292, 2013.

[8] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104–113. IEEE, 2011.

[9] Deepal Jayasinghe, Galen Swint, Simon Malkowski, Jack Li, Qingyang Wang, Junhee Park, and Calton Pu. Expertus: A generator approach to automate performance testing in iaas clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 115–122. IEEE, 2012.

[10] Alexey Kopytov. Sysbench: a system performance benchmark. *URL: http://sysbench.sourceforge. net*, 2004.

[11] S. Koulouzis. Montage dockerfile. https://github.com/skoulouzis/dockerfiles/tree/master/Montage, 2017.

[12] Nane Kratzke and Peter-Christian Quint. About automatic benchmarking of iaas cloud service providers for a world of container clusters. *Journal of Cloud Computing Research*, 1(1):16–34, 2015.

[13] Michael Larabel and M Tippett. Phoronix test suite. *h ttp://www. phoronix-test-suite. com*, 2011.

[14] Philipp Leitner and Jürgen Cito. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. *ACM Transactions on Internet Technology (TOIT)*, 16(3):15, 2016.

[15] John D McCalpin. Stream benchmark. *URL: http://www. cs. virginia. edu/stream/stream2*, 2002.

[16] William D Norcott and Don Capps. Iozone filesystem benchmark, 2003.

[17] J. Scheuner and P. Leitner. cloud-workbench. https://github.com/sealuzh/cloud-workbench, 2014.

[18] Joel Scheuner, Jürgen Cito, Philipp Leitner, and Harald Gall. Cloud workbench: Benchmarking iaas providers based on infrastructure-as-code. In *Proceedings of the 24th International Conference on World Wide Web*, pages 239–242. ACM, 2015.

[19] Joel Scheuner, Philipp Leitner, Jürgen Cito, and Harald Gall. Cloud work bench–infrastructure-as-code based cloud benchmarking. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 246–253. IEEE, 2014.

[20] M. Silva and M. Hines. Cloud rapid experimentation and analysis toolkit. `https://github.com/ibmcb/cbtool`, 2016.

[21] Marcio Silva, Michael R Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, and Dilma Da Silva. Cloudbench: experiment automation for cloud environments. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 302–311. IEEE, 2013.

[22] Blesson Varghese, Ozgur Akgun, Ian Miguel, Long Thai, and Adam Barker. Cloud benchmarking for performance. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 535–540. IEEE, 2014.

[23] Blesson Varghese, Ozgur Akgun, Ian Miguel, Long Thai, and Adam Barker. Cloud benchmarking for maximising performance of scientific applications. *IEEE Transactions on Cloud Computing*, 2016.

7 Appendices

A Experiments

Table 6: Comparing the results the experiment 1 and 2 (sysbench)

Instance	Experiment 1			Experiment 2		
Type	SD	Avg	RSD	SD	Avg	RSD
NICTA M	1.5	245.2	0.65%	0.16	245.11	0.07%
BBN M	19	272	7.31%	0.75	287.69	0.26%
UvA M	-	-	-	0.50	241.80	0.21%
NICTA L	0.83	123.08	0.68%	0.77	122.75	0.63%
BBN L	8.3	125.3	6.68%	0.11	122.89	0.09%
UvA L	-	-	-	0.35	122.49	0.29%
NICTA XL	0.29	61.37	0.47%	0.05	61.19	0.08%
BBN XL	7.8	71.3	10.98%	0.50	73.43	0.68%
UvA XL	-	-	-	0.31	62.73	0.49%

Table 7: Comparing the results the experiment 1 and 2 (STREAM)

Instance	Experiment 1			Experiment 2		
Type	SD	Avg	RSD	SD	Avg	RSD
NICTA M	78	9851	0.79%	88	9821	0.90%
BBN M	1502	11907	12.62%	156	12805	1.22%
UvA M	-	-	-	162	14569	1.12%
NICTA L	496	13501	3.68%	495	13499	3.67%
BBN L	2485	14357	17.31%	735	13346	5.51%
UvA L	-	-	-	40	24816	0.16%
NICTA XL	469	14217	3.30%	469	14217	3.30%
BBN XL	7518	24298	30.94%	55	29706	0.19%
UvA XL	-	-	-	59	29912	0.20%

Table 8: Comparing the results the experiment 1 and 2 (IOzone read)

Instance Type	Experiment 1			Experiment 2		
	SD	Avg	RSD	SD	Avg	RSD
NICTA M	314	6556	4.80%	432	6495	6.66%
BBN M	327	6865	4.77%	118	6962	1.70%
UvA M	-	-	-	426	7784	5.48%
NICTA L	505	6287	8.04%	140	6251	2.24%
BBN L	419	6584	6.38%	66	6605	1.01%
UvA L	-	-	-	148	3516	4.22%
NICTA XL	305	6440	4.74%	183	6096	3.01%
BBN XL	363	6734	5.40%	407	6723	6.05%
UvA XL	-	-	-	154	7653	2.01%

Table 9: Comparing the results the experiment 1 and 2 (IOzone write)

Instance Type	Experiment 1			Experiment 2		
	SD	Avg	RSD	SD	Avg	RSD
NICTA M	19	110	17.59%	17	115	15.16%
BBN M	323	358	90.39%	71	877	8.16%
UvA M	-	-	-	16	186	8.66%
NICTA L	44	144	31.13%	43	124	35.06%
BBN L	54	192	28.28%	79	245	32.15%
UvA L	-	-	-	30	161	18.98%
NICTA XL	94	266	35.54%	59	186	31.66%
BBN XL	57	306	18.83%	85	418	20.55%
UvA XL	-	-	-	133	429	31.07%