Performance Analysis of an OpenStack Private Cloud

Tamas Pflanzner[1], Roland Tornyai[1], Balazs Gibizer[2], Anita Schmidt[2] and Attila Kertesz[1]
[1]*Software Engineering Department, University of Szeged, 6720 Szeged, Dugonics ter 13, Hungary*
[2]*Ericsson Hungary, 1476 Budapest, Konyves Kalman krt. 11, Hungary*
{*tampfla,tornyai,keratt*}*@inf.u-szeged.hu,* {*balazs.gibizer, anita.schmidt*}*@ericsson.com*

Keywords:
Cloud Computing, Performance analysis, OpenStack

Abstract:
Cloud Computing is a novel technology offering flexible resource provisions for business stakeholders to manage IT applications and data responding to new customer demands. It is not an easy task to determine the performance of the ported applications in advance. The virtualized nature of these environments always represent a certain level of performance degradation, which is also dependent on the types of resources and application scenarios. In this paper we have set up a performance evaluation environment within a private OpenStack deployment, and defined general use cases to be executed and evaluated in this cloud. These test cases are used for investigating the internal behavior of OpenStack in terms of computing and networking capabilities of its provisioned virtual machines. The results of our investigation reveal the performance of general usage scenarios in a local cloud, give an insight for businesses planning to move to the cloud and provide hints where further development or fine tuning is needed in order to improve OpenStack systems.

1 INTRODUCTION

Cloud Computing is a diverse research area, its novel technology offers on-demand access to computational, infrastructure and data resources operated remotely. This concept has been initiated by commercial companies to allow elastic construction of virtual infrastructures, and its technical motivation has been introduced in [Buyya et al., 2009][Vaquero et al., 2008]. Cloud solutions enable businesses to outsource the operation and management processes of IT infrastructure and services, therefore their applicants can concentrate on their core competencies. Nevertheless it is not an easy task to determine the performance of the ported applications in advance. The virtualized nature of these environments always represent a certain level of performance degradation, which is also dependent on the types of resources used and application scenarios applied.

In this paper we have set up a performance evaluation environment using Rally [Ishanov, 2013] within a private Mirantis [Mirantis, 2015b] OpenStack deployment, and defined general use cases to be executed and evaluated in this local cloud. The main contributions of this paper are (i) the automated Rally performance evaluation environment, and (ii) the predefined set of test cases used for investigating the internal behavior of OpenStack in terms of computing and networking capabilities of its provisioned virtual machines. The results of our investigation reveal the performance of general usage scenarios in a private cloud, give an insight for business stakeholders planning to move to the cloud and provide hints where further development is needed in OpenStack.

The remainder of this paper is as follows: Section 2 gives an overview of the related works, and Section 3 introduces the installation of our private cloud. Section 4 defines the test cases and presents their evaluation. Finally, Section 5 concludes the paper.

2 RELATED WORK

Cloud monitoring is closely related to benchmarking, and nowadays it is a widely studied re-

search area and several solutions have emerged both from the academic and commercial fields. Fatema et al. [Fatema et al., 2014] created a survey of 21 monitoring tools applicable for cloud systems. They introduced the practical capabilities that an ideal monitoring tool should possess to serve the objectives in these operational areas. Based on these capabilities, they also presented a taxonomy and analysed these monitoring tools to determine their strengths and weaknesses. Most of these cloud monitoring tools offer their services at the Software as a Service (SaaS) level that can be used to monitor third party cloud installations. To realize this, third party clouds must support the installation and execution of SaaS agents. Many cloud monitoring tools are capable of monitoring at the infrastructure and application levels, while some others can only monitor one of those levels.

Concerning cloud benchmarking, Ficco et al. [Ficco et al., 2015] defined the roles of benchmarking and monitoring of service performance in Cloud Computing, and presented a survey on related solutions. They argued that in general benchmarking tools should be more flexible, and the usage of a single performance index is not acceptable and workload definition should be customizable according to user specific needs. Leitner et al. [Leitner and Cito, 2014] performed a benchmarking of public cloud providers by setting up hypotheses relating to the nature of performance variations, and validated these hypotheses on Amazon EC2 and Google Compute Engine. With this study they showed that there were substantial differences in the performance of different public cloud providers. Our aim is to investigate a local, private cloud based on OpenStack.

The primary goal of the CloudHarmony [Cloudharmony, 2014] is to make cloud services comparable, therefore they provide objective, independent performance comparisons between different cloud providers. Using these data, customers can quickly compare providers and have reasonable expectations for cloud performance. However, CloudHarmony can only provide quantitative performance data in a raw form produced by benchmark tools and cannot present refined qualitative information created from processed benchmark results.

Ceilometer [OpenStack, 2015a] is an OpenStack project designed to provide an infrastructure to collect measurements within OpenStack so that only one agent is needed to collect the data. The primary targets of the project are monitoring and metering, but the framework can be extended to collect usage for other needs. Rally [Ishanov, 2013] is a more advanced solution for benchmarking and profiling OpenStack-based clouds. Its tools allow users or developers to specify some kind of synthetic workload to stresstest OpenStack clouds and get the low-level profiling results. Rally is able to collect monitored information about executing specific scenarios, like provisioning a thousand virtual machines (VM), and shows how a cloud performs on average in that environment. Since cloud operators typically do not run user workloads, therefore Rally provides an engine that allows developers to specify real-life workloads and runs on existing OpenStack clouds. The results generated from these kinds of benchmarks are more high level, but they allow users to identify bottlenecks on a specific cloud. In our work we used and extended Rally scenarios to benchmark our private cloud.

3 SETTING UP A PRIVATE CLOUD BASED ON OPENSTACK

OpenStack [OpenStack, 2015c] is a global collaboration of developers and cloud computing technologists producing the ubiquitous open source cloud computing platform for public and private clouds. It aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and feature rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. It has 13 official distributions [OpenStack, 2015b], and we have chosen Mirantis [Mirantis, 2015b] for the base distribution of our private cloud, since it is the most flexible and open distribution of OpenStack. It integrates core OpenStack, key related projects and third party plugins to offer community innovations with the testing, support and reliability of enterprise software.

When calculating resources for an OpenStack environment, we should consider the resources required for expanding our planned environment. This calculation can be done manually with the help of the example calculation [Mirantis, 2014a] or by an automatic tool, like the Bill of Materials calculator. The OpenStack Hardware Bill of Materials (BOM) calculator [Mirantis, 2014b] helps anyone building a cloud to identify how much

Table 1: Hardware parameters of our private OpenStack cloud.

	Type 1	Type 2
System	IBM BladeCenter HS21	BladeCenter LS21
CPU	8x 2.66GHz Xeon E5430	4x 2.4GHz Opt. 2216HE
RAM	4x 2GB, 8GB total	4x 1GB, 4GB total
DISK	1 drive, 68.4 GB total	1 drive, 35GB total
INTERFACE	2x 1.0 Gbps	

Number of nodes by type	
3x Type 1	1x Type 2
2x Type 1 + 8 GB RAM, 16 GB total	1x Type 2 + 500 GB DISK
2x Type 1 + 700 GB DISK	

hardware and which server model they need to build compute services for a cloud. In our case we had some dedicated resources for setting up our planned cloud, therefore we only had to perform a validity check [Mirantis, 2015a] to be sure that our hardware pool is capable of hosting an OpenStack cloud. The parameters of our dedicated hardware are shown in Table 1.

Mirantis consists of three main components [Mirantis, 2015b]: (i) Mirantis OpenStack hardened packages, (ii) Fuel for OpenStack, and (iii) Mirantis Support. The hardened packages include the core OpenStack projects, updated with each stable release of OpenStack, and supporting a broad range of operating systems, hypervisors, and deployment topologies, including support for high availability, fixes for reported but yet not merged defects to the community source, and Mirantis-developed packages, such as Sahara and Murano. Fuel is a lifecycle management application that deploys multiple OpenStack clouds from a single interface and then enables users to manage those clouds post deployment. One can add nodes, remove nodes, or even remove clouds, restoring those resources to the available resources pool, and it also eases the complexities of network and storage configurations through a simple-to-use graphical user experience. It includes tested reference architectures and an open library to ease configuration changes.

An OpenStack environment contains a set of specialized nodes and roles. When planning an OpenStack deployment, a proper mix of node types must be determined and selected what roles will be installed on each, therefore each node should be assigned by a role denoting a specific component. Fuel is capable of deploying these roles to the nodes [Mirantis, 2015c] of our system. The most important nodes are the followings [Mirantis, 2015d]: a Controller node initiates orchestration activities and offers important services like identity management, web dashboard and scheduler. A Compute node handles the VMs lifecycle and includes the nova-compute service that creates, manages and terminates virtual machine instances. Considering storage nodes, Cinder LVM is the default block storage backend for Cinder and Glance components [OpenStack, 2015e]. Block storage can be used for database storage, expandable file system or providing a server with access to raw block level devices. Ceph is a scalable storage solution that replicates data across the other nodes, and it supports both object and block storage. The absolute minimum requirement for a highly-available OpenStack deployment is to allocate 4 nodes: 3 Controller nodes, combined with storage, and 1 Compute node. In production environments, it is highly recommended to separate storage nodes from controllers to avoid resource contention, isolate failure domains, and to be able to optimize hardware configurations for specific workloads.

To start the deployment process, we created some initial cloud installations with different configurations, in which we did not use all the available machines dedicated for our private cloud. These different configurations aimed at both non-HA and HA systems, and we experimented with different network topologies like the basic nova-network flat DHCP and neutron with GRE segmentation. We also deployed the first environments with the default LVM storage, but later we switched to Ceph. Once we arrived to a reliable distribution of components, we created a short documentation about the configuration of our planned cloud system, and shared it with our colleagues at Ericsson. In order to arrive to a more enterprise-like cloud deployment, we changed the network settings to separate the management network from the storage and Compute network, because the storage network can produce big load of

network traffic and it can slow down the management network. As a result we removed the storage roles from the controller nodes. Since we did not have big hard drives in these nodes, we did not lose significant storage capacity. Though in the OpenStack documentation the storage role is not recommended for the controllers, in a small cloud (having 4-10 nodes) it can be reasonable. Finally we arrived to the deployment shown in Table 2.

4 PERFORMANCE ANALYSIS OF OPENSTACK

After reviewing and considering benchmarking solutions from the literature, we selected Rally [OpenStack, 2015f] as the main benchmarking solution for the performance analysis of our private OpenStack cloud. We defined several scenarios to analyze the performance characteristics of our cloud. In some cases we also used the Python API of OpenStack [OpenStack, 2015d] to create specific test scenarios. In the following subsections we introduce these scenarios, and present the result of our experiments.

4.1 Benchmarking scenarios

OpenStack is a really big ecosystem of cooperative services, and when something fails, performs slowly or does not scale, it is really hard to answer questions on what, why and where it has happened. Rally [OpenStack, 2015f] can help to answer these questions, therefore it is used by developers to make sure that a newly developed code works fine and helps to improve OpenStack. Some typical use cases for Rally can help to configure OpenStack for a specific hardware, or they can show the OpenStack quality by time with historical data of benchmarks. Rally consists of 4 main components: Server Providers to handle VMs, Deploy Engines to deploy the OpenStack cloud, Verification to run tempest (or other tests), collect the results and present them in a human readable form, and Benchmark engine to write parameterized benchmark scenarios.

Our goal is to provide test cases that can measure the performance of a private cloud, could help in finding bottlenecks and can be used to ensure that our cloud will be working as expected in a close to real life utilization. The VM lifecycle handling (start, snapshot and stop), user handling, networks and migration will be in the focus of our benchmarking tests. In the first round of experiments we will benchmark specific parts of the cloud without stress testing other parts, and later these tests will be repeated with artificially generated stress on the system. As a future work, these test cases could be used to compare different infrastructure configurations, for example to compare the native OpenStack and Mirantis default settings, or other custom configurations.

In all scenarios we will use three types of VM flavors: (i) small (fS) - 1 VCPU; 1536 MB RAM, (ii) medium (fM) - 2 VCPU; 3072 MB RAM and (iii) big (fB) - 4 VCPU; 6144 MB RAM. The following OS images will be used for the testing VMs: Ubuntu, Xubuntu, CirrOS. Our basic test scenarios are the followings:

1. VM start and stop: The most basic VM operations are to start and stop a VM. In this scenario we perform these operations, and measure the time taken to start a VM and decommissioning it. The VM will be booted from image and from volume too.

2. VM start, create snapshot, stop: Creating a snapshot is an important feature of a cloud. Therefore in this scenario we start a VM, save a snapshot of the machine, then decommission it. The two subscenarios are when the VM is booted from an image and from a volume.

3. Create and delete image: The image creation and deletion are usual operations. In this case we measure the time taken to create a new VM image, and to delete an existing VM image file.

4. Create and attach volume: In this scenario we will test the storage performance by creating a volume and attaching it to a VM.

5. Create and delete networks: In this scenario we examine the networking behavior of the cloud by creating and removing networks.

6. Create and delete subnets: In this case we will measure subnet creation and deletion by creating a given number of subnets and then delete them.

7. Internal connection between VMs: In this scenario we will measure the internal connection reliability between VMs by transferring data between them.

8. External connection: In this scenario we will measure the external connection reliability by downloading and uploading 1 GB data from and to a remote location.

Table 2: Deployment parameters of our private OpenStack cloud.

Distribution	Mirantis 5.0.1 (OpenStack Icehouse)
Extra components	Ceilometer, High Availibilty (HA)
Operating System	Ubuntu 12.04 LTS (Precise)
Hypervisor	KVM
Storage backend	Ceph
Network (Nova FlatDHCP)	Network #1: Public, Storage, VM Network #2: Admin, Management
Fuel Master node	1x Type 2
Controller nodes	2x Type 1
Controller, telemetry, MongoDB	1x Type 1
Compute, Storage - Ceph nodes	2x Type 1+ 8GB RAM 2x Type 1 + 700GB DISK
Storage Cinder	1x Type 2 + 500GB DISK

9. Migration: Migration is also an important feature of a cloud, therefore we will test live migration capabilities in this scenario.

To fully test the cloud environment, we need to examine the performance in scenarios with artificially generated background load, where specific operations could also affect the overall performance. Therefore we will examine the following cases:

- Concurrency: User handling and parallel operations are important in a cloud, so we will execute several scenarios concurrently.
- Stress: We will use dedicated stressing VMs (executing Phoronix benchmarks) to intensively use the allocated resources of the cloud, and measure how the VM behavior will change compared to the original scenarios.
- Disk: We will also perform scenarios with varying disk sizes. The basic test case scenarios will be executed in different circumstances, which are specified by the above three factors. As a result we have 8 test categories, but not all of them will be used for each scenario. Table 3 shows the defined test case categories.

Table 3: Test case categories for cloud benchmarking.

Category	Concurrency	Stress	Disk
1	NO	NO	NO
2	NO	NO	YES
3	YES	NO	NO
4	YES	NO	YES
5	NO	YES	NO
6	NO	YES	YES
7	YES	YES	NO
8	YES	YES	YES

Concerning the built-in Rally scenarios, we had to create JSON parameter files that specify the details of the actual test. Nevertheless for an actual scenario we had different test cases, which had to be defined by different JSON parameters. Therefore we developed a Java application that is able to generate custom JSON description files for the different cases. The Java application has a Constants class, where the JSON parameters can be modified in one place, like the used image for the VMs or the flavors. The BaseScenario class represents a general scenario and defines some general methods like using different flavors or setting the concurrency of a test case. We created some other classes, which are used for the JSON generation with GSON (Google Java library to convert Java object to JSONs). Every scenario has its own Java class, for example the Scenario01 class, where we can define additional capabilities that are extensions to the general BaseScenario. To run test cases in an automated way, we used a bash script.

Because of the intensive development progress in the Rally development, we tried to use the latest version, but we also wanted to have all the used versions, in case an exact test recreation would be needed. That is why we have multiple Rally folders with different versions. The script iterates on all the folders in the Scenarios folder and generates HTML reports.

Concerning Scenario 7 and 8, we planned to use custom scripts inside Rally. We created these scripts, but we experienced problems related to network access during executing these cases. Rally generates special user accounts for each case, and sometimes in the custom scripts not all created entities can be modified or accessed. To overcome these problems, we used the

Table 4: Summary of the performance analysis results.

Scenario number	No Stress						Stress					
	No Concurrency			Concurrency			No Concurrency			Concurrency		
	fS	fM	fB	fS	fM	fB	fS	fM	fB	fS	fM	fB
1/Image	87	86	92	180	163	158	119	105	126	307	164	191
1/Volume	101	101	109	290	208	301	120	118	126	278	233	307
2	183	189	185	316	329	288	208	202	210	397	421	373
3	8			11			7			12		
5	0.304			0.549			0.287			0.426		
6	0.598			0.858			0.624			0.923		
7	Upload / Download			N/A								
	32.079 / 246.398			N/A								
8	61.703			N/A								
9	90	96	97	165	181	N/A	110	109	N/A	203	214	N/A

OpenStack Python API to create custom scripts for these scenarios and execute them without using Rally.

4.2 Evaluation results

In this subsection we present the results of our performance analysis of our private OpenStack cloud installed at the Software Engineering Department (SED) of the University of Szeged, Hungary. Table 4 summarizes the measured values for all cases (in seconds), while Figure 1 shows the charts generated by Rally for a specific test cases of Scenario 2.

4.3 Discussions

In Scenario 1 more than 95% of the execution time was spent on VM booting. The flavor of the VM made minimal difference in time, but in the concurrent cases the measurements with big flavors resulted in only 60% success ratio. Also in the concurrent cases the measured booting time was in average twice as much as in the non-current cases (4 VMs were started in the concurrent tests at the same time).

Within this scenario we also investigated booting from volume instead of an image. We found that the average booting time took 10% more in non-concurrent cases, and more than 40% execution time increase in concurrent cases, and we also experienced higher deviations. The number of errors were also increased, the usual error type was: Block Device Mapping is Invalid.

Concerning different flavors for Scenario 2 we arrived to a similar conclusion, i.e. in the measured times there was minimal difference, but the concurrent test cases with big flavors had only 30% success rate. The image creation and VM booting had around 45% of the measured time each (as shown in Fig. 1). The concurrent executions almost doubled the measured time of the scenarios. For the second round of experiments using stressing VMs on the nodes, we experienced around 10% increase for non-concurrent cases and 30% increase for the concurrent ones.

In Scenario 3 image creation took most of the time, while deletion had 20% to 40% of the overall measurement time. In the concurrent cases it took around 1.5 times more to perform the same tasks. It is interesting that for the concurrent scenarios image deletion took longer than in the non-concurrent cases, compared to the performance degradation of image creation cases.

Concerning Scenario 4, all measurements have failed (due to timeout operations). After investigating the problem, we found that it seems to be a bug in Rally, since attaching volumes to VMs through the web interface works well, and can be performed within 1-2 seconds. Therefore we did not detail these results in Table 4.

In Scenario 5, for the concurrent cases we experienced around 10% increase in execution time for creating and deleting networks. The creation and deletion ratio not changed in a significant way, the deletion ratio raised from 30% to 35%.

In Scenario 6, we examined subnet management. Both in the non-concurrent and concurrent cases we experienced 40% failures due to tenant network unavailability. The concurrent cases took a bit more than twice as much time to perform.

Scenarios 7 and 8 have been implemented in customs scripts using the OpenStack Python API. The results for data transfers show that uploading to an external server was 10 times faster

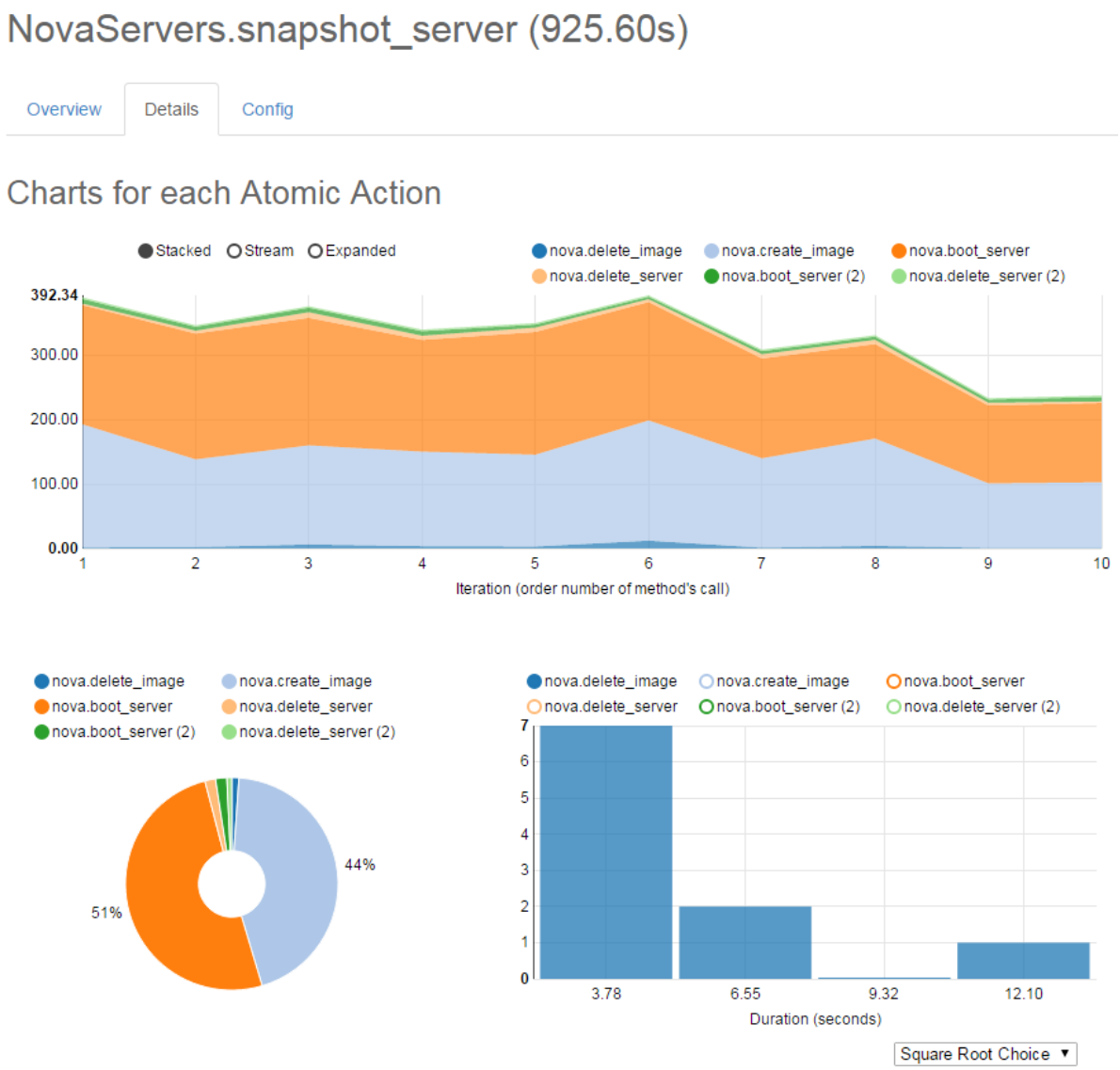

Figure 1: Detailed results and charts for the concurrent test case of Scenario 2 with medium VM flavor.

in average (because it was within the same building) than the downloading from a server (located in Germany). Concerning the internal connection between VMs within the cloud we found that it was twice slower than the external upload to a remote server within the building. During the data transfers we experienced a couple of errors with the following types: 113 - 'No route to host' and 111 - 'Connection refused'. These cases were rerun.

During the evaluation of Scenario 9 we had a hardware failure in one of the computing nodes, which resulted in high number of errors. Concerning the successful cases, we experienced nearly 50% time increase in concurrent cases to the non-concurrent ones.

5 CONCLUSION

Cloud computing offers on-demand access to computational, infrastructure and data resources operated from a remote source. This novel technology has opened new ways of flexible resource provisions for businesses to manage IT applications and data responding to new demands from customers. Nevertheless it is not an easy task to determine the performance of the ported applications in advance.

In this paper we proposed a set of general cloud test cases and evaluated a private OpenStack cloud deployment with a performance evaluation environment based on Rally. These test cases were used for investigating the internal behavior of OpenStack components in terms of computing and networking capabilities of its provisioned virtual machines.

The results of our investigation showed the performance of general usage scenarios in a local cloud. In general we can conclude that stressing a private cloud with targeted workloads does introduce some performance degradation, but the system returns to normal operation after the stressing load. We also experienced failures in certain cases, which means that fresh cloud deployments need to be fine-tuned for certain scenarios. We believe that we managed to give an insight of cloud behavior with our test cases for businesses planning to move to the cloud. In our future work will continue investigating OpenStack behavior with additional test cases derived from real world applications.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from Ericsson Hungary Ltd.

REFERENCES

[Buyya et al., 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst*, 25(6):599–616.

[Cloudharmony, 2014] Cloudharmony (2014). Cloudharmony website, http://cloudharmony.com, dec. 2014.

[Fatema et al., 2014] Fatema, K., Emeakaroha, V. C., Healy, P. D., Morrison, J. P., and Lynn, T. (2014). A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918 – 2933.

[Ficco et al., 2015] Ficco, M., Rak, M., Venticinque, S., Tasquier, L., and Aversano, G. (2015). Cloud evaluation: Benchmarking and monitoring. In *Quantitative Assessments of Distributed Systems*, pages 175–199. John Wiley & Sons, Inc.

[Ishanov, 2013] Ishanov, K. (2013). Openstack benchmarking on softlayer with rally.

[Leitner and Cito, 2014] Leitner, P. and Cito, J. (2014). Patterns in the chaos - a study of performance variation and predictability in public iaas clouds. *CoRR*, abs/1411.2429.

[Mirantis, 2014a] Mirantis (2014a). Calculation for openstack deployments, http://docs.mirantis.com/openstack/fuel/fuel-5.0/pre-install-guide.html#hardware-calculation, december 2014.

[Mirantis, 2014b] Mirantis (2014b). Hardware calculator for openstack deployments, https://www.mirantis.com/openstack-services/bom-calculator/, december 2014.

[Mirantis, 2015a] Mirantis (2015a). Confirm hardware for openstack deployments, http://docs.mirantis.com/openstack/fuel/fuel-5.0/user-guide.html#confirm-hardware, december 2015.

[Mirantis, 2015b] Mirantis (2015b). Mirantis software website, https://software.mirantis.com/, december 2015.

[Mirantis, 2015c] Mirantis (2015c). Openstack deployment guide, http://docs.mirantis.com/openstack/fuel/fuel-5.0/user-guide.html#create-a-new-openstack-environment, december 2015.

[Mirantis, 2015d] Mirantis (2015d). Planning guide for openstack deployments, http://docs.mirantis.com/openstack/fuel/fuel-5.0/pre-install-guide.html, december 2015.

[OpenStack, 2015a] OpenStack (2015a). Calculation for openstack deployments, http://docs.openstack.org/developer/ceilometer/, october 2015.

[OpenStack, 2015b] OpenStack (2015b). Openstack distributions, http://www.openstack.org/marketplace/distros, december 2015.

[OpenStack, 2015c] OpenStack (2015c). Openstack project website, http://www.openstack.org, december 2015.

[OpenStack, 2015d] OpenStack (2015d). Openstack python clients, https://wiki.openstack.org/wiki/openstackclients, december 2015.

[OpenStack, 2015e] OpenStack (2015e). Openstack roadmap, http://www.openstack.org/software/roadmap/, december 2015.

[OpenStack, 2015f] OpenStack (2015f). Rally wiki page, https://wiki.openstack.org/wiki/rally, october 2015.

[Vaquero et al., 2008] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55.