Evaluation of resource allocation schemes on OpenStack

Jerico Moeyersons
Supervisors: Prof. dr. ir. Filip De Turck, Prof. dr. Bruno Volckaert
Counsellor: Pieter-Jan Maenhaut

***Abstract*—One of the major challenges of cloud computing is to efficiently divide and manage resources over different customers, also referred to as resource allocation. Over recent years a lot of resource allocation schemes have been designed and tested by means of simulation tools such as the CloudSim framework, but very rarely on a real cloud testbed. Large experiments using real cloud testbeds are both expensive and time-consuming, especially when there are some faults in the new scheme. In this thesis, we try to offer an alternative to test and evaluate new allocation schemes on a small OpenStack cloud environment. OpenStack is a free, open-source software platform for cloud computing, deployed as Infrastructure-as-a-Service. With the aid of a scalable OpenStack application and a monitor web application, new resource allocation schemes can be plugged into an OpenStack cloud and evaluated. A Proof-of-Concept, with a Round Robin allocation scheme, is introduced to verify how easy and fast an evaluation works with this approach. After this evaluation, we can conclude that with OpenStack, two test scenarios and a monitoring application, a new resource allocation scheme can be easily and quickly evaluated.**

Keywords*—Cloud computing, OpenStack, resource allocation, Round Robin, monitoring

I. Introduction

With cloud computing, efficient resource management is of great importance. A cloud application can benefit from efficient resource management so it'll have enough resource to work well, will be more reliable and fault tolerant, etc. and thus good resource allocation algorithms are needed. In recent years, many new cloud resource allocation schemes have been designed and evaluated but only half of them are evaluated using a real cloud testbed. The others are only validated using cloud simulation frameworks such as CloudSim [1]. Though, simulations are needed for the design and development of new allocation strategies, evaluations on physical hardware can bring new insights such as how new resource allocation schemes react on the heterogeneity of physical resources. Running experiments on a public cloud is both expensive and time-consuming, causing that not every new allocation strategy will be evaluated on a public cloud because a simulator can do it faster.

One of the strengths of the cloud is the possibility to scale up and down or in and out. When instances, hosted by the cloud, are scaled up or down, additional resources are added or removed from that instance. This is an easy and fast approach but not every application can benefit from it because these applications need to be designed to deal with more resources. The other approach, scaling out or in is far more interesting but more complex. When a cloud application is scaled out, more instances are added to split the needed calculations over the available instances and when a cloud application is scaled in, some instances are deleted. This approach requires a specific software architecture so it can benefit from more instances but useful to test new resource allocation schemes.

In this thesis, a setup is presented for easy and quick evaluation of new allocation strategies on top of an OpenStack cloud [2] together with an OpenStack application and a monitor setup. To verify this setup, a Proof-of-Concept (PoC) with an existing allocation scheme, Round Robin, is evaluated.

II. Related work

Some work that can be related to this research is the Raspberry Pi cluster explained in Maenhaut et al [3]. In this paper, a cluster of Raspberry Pi's gives the opportunity to validate new allocation strategies instead of using a simulator. Through the provided web interface, new clusters can be easily configured. The way how the clusters are monitored is strongly related to the used monitoring system in this thesis, described in Section V.
Other work is RT-OpenStack [4], a CPU-resource management system plugged into OpenStack. The system consists of three components, namely the integration of a real-time hypervisor, a real-time scheduler and a VM-to-host mapping strategy. The interesting part of this work is the way how this system is used in combination with an OpenStack cloud but this thesis won't use custom management systems and hypervisors to evaluate new resource allocation schemes.
Beside this, there are also different simulators such as OpenStack EMU [5], a simulated OpenStack environment with a SDN controller and a large scale network emulator CORE. The quantity of simulators to evaluate new resource allocation schemes is one of the most import reasons of this thesis because of the need to evaluate these schemes on physical hardware instead of simulated hardware.

III. What is OpenStack?

OpenStack [2] is an open-source software platform for cloud computing, under the Apache License 2.0, that controls large

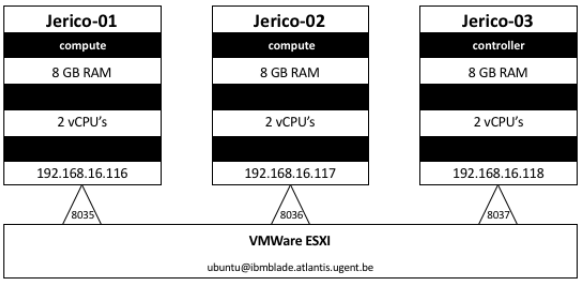

Fig. 1: Structure of the test environment

pools of compute, storage and networking resources throughout a datacenter. OpenStack works with popular open-source technologies so it's ideal for heterogeneous infrastructure. Most of the time, it is deployed as Infrastructure-as-a-Service (IaaS) whereby virtual servers, containers and other resources are made available to the customers. The platform itself consists of multiple, interconnected components managing the different resources (such as CPU, memory, storage, network, etc). All these components are controlled through a RESTful API, a web based dashboard or through command line tools. The most relevant components of OpenStack for this thesis are Nova, responsible for managing the compute resources, Heat, responsible for the orchestration services and Aodh and Ceilometer, both responsible for the telemetry services. Apart from these components, there are many other ones created by the OpenStack community but these are less related to this thesis.

IV. Test environment

For this thesis, OpenStack is deployed on 3 Ubuntu 16.04.02 LTS systems, each with 8 GB of memory, 2 vCPUs and 10 GB of storage. The OpenStack test environment is deployed using DevStack [6], a collection of bash scripts that automatically deploys an OpenStack cloud with less configuration, ideal for development or test environments. The structure of the test environment is visualized in Figure 1.
After the deployment of OpenStack, the Round Robin scheduler is plugged into Nova. By default, Nova works with a filter and weighting scheduler that first filters the available hosts and then selects the best hypervisor to host the new instances based on the amount of free memory, average CPU usage, etc. For implementing the Round Robin scheduler, the same principle is used, namely, Round Robin is written as filter with a row of the different hypervisors (in this case, the 3 hypervisors in Figure 1). When a new instance needs to be created, only one hypervisor will be chosen by the Round Robin filter and thus the default Nova weighter will let that hypervisor host the new instance (because there is but one hypervisor that came through the filter).
Finally, a cloud application called FaaFo (First App Application For Openstack) [7] is deployed with an OpenStack stack. FaaFo is a cloud application that calculates fractals with the possibility to split the work over different worker nodes, all orchestrated by a controller node. A stack, written in the Heat Orchestration Template (HOT) format [8], is used for the creation of security groups, alarms, servers, scaling groups, etc. Such a stack is then monitored by Ceilometer and Aodh, the Telemetry services of OpenStack, and when the CPU-usage is too high or low, Heat will scale out or in the whole cloud application or in this case the number of worker nodes.
The combination of these three steps allows us to create a cloud environment, plug in a new allocation strategy and generate some workload to test the placement of new instances and the distribution of resources over the different hypervisors.

V. Monitoring tool

There are many possibilities to monitor the hosts of an OpenStack cloud. The first possibility is Rally [9], a tool that automates and unifies a multi-node OpenStack deployment, verifies the cloud, benchmarks the deployment and profiles it. In this context, Rally is used for executing benchmarking tests and for the possibility to verify the cloud or in other words, to check of the cloud is working properly under simulated workload.
The second possibility is a custom developed nodejs-agent. This agent, developed using Node.js, uses different Node packages to construct a web service that will save the current resource usage such as CPU-usage, free memory and number of hosted instances in a MongoDB and also returns these results to the caller.
Other monitor tools exists, such as Grafana and DataDog. The OpenStack community is also busy with a project called Monasca that will be able to monitor the whole OpenStack cloud.
For the evaluation of resource allocation schemes on OpenStack, we used the first and second possibility because it is a free option with less and easy configuration in contrast to DataDog, a paying service, or Grafana that needs many and difficult configuration. Also the possibility to adapt this application or expand it with more graphs is interesting for other evaluations.

VI. Evaluation of schemes

With the test environment described in Section IV and the monitoring tools Rally and the nodejs-agent described in Section V, the PoC with the Round Robin scheduler can be evaluated.

First of all, a Rally test creates 10 times a standard instance with two iterations at the time (concurrency), simulating that two users are creating instances at the same time. While the test was being executed, the nodejs-agent was monitoring the three hypervisors and show the results as visible in Figure 2 after the completion of the Rally test. In this case, the instances are equally divided between the three hypervisors but the CPU-usage and the memory usage of jerico-03, the OpenStack controller, is significantly greater than the two OpenStack-compute nodes.
After the Rally test, the FaaFo application is deployed with a stack on the cloud. By default, the deployment consists of

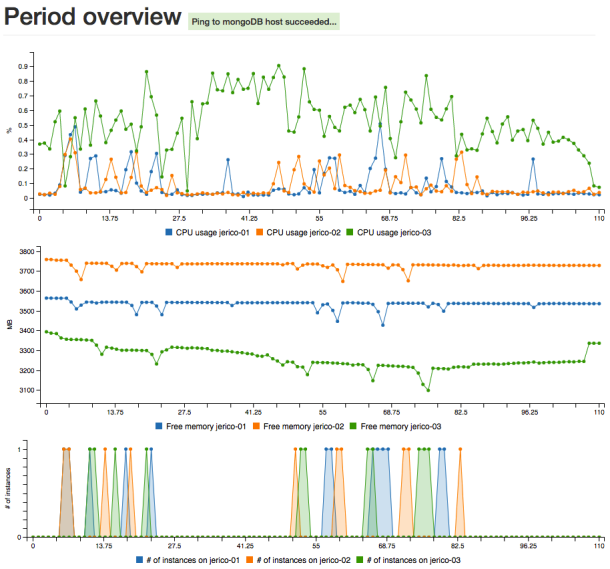

Fig. 2: Results of the Rally test in the nodejs web application

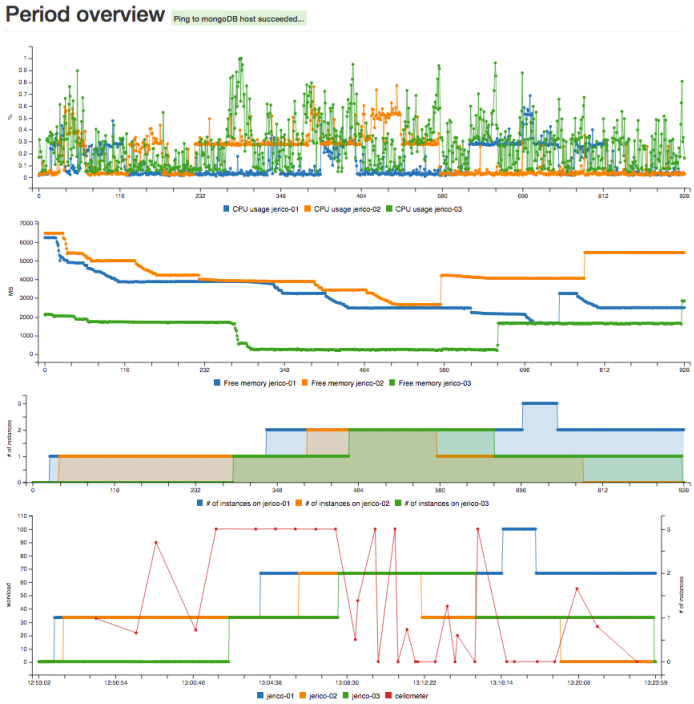

Fig. 3: Results of the FaaFo test in the nodejs web application

the required security groups, alarms, scaling policies, one controller node and one worker node. When the deployment is complete and FaaFo is running, a task to create three 5555 x 5555 fractals is started on the FaaFo controller. The only worker node will start to calculate these fractals with a high CPU usage as a result. Thanks to the configured alarms in the stack, Ceilometer and Aodh will monitor the worker node and trigger an alarm when the CPU usage is too high. Heat, on his turn, will then start a new worker node that can help the other node to calculate the fractals. This will go on till everything is calculated and when the CPU usage starts decreasing, each unnecessary worker node will be deprovisioned.

From the Rally test, the following can be concluded: the creation and deletion of different instances in this order is no problem for a Round Robin scheduler. The whole process of the FaaFo test, from deploying the stack, calculate the fractals and in- and out-scaling the worker nodes is also monitored with the nodejs agent and visualized in Figure 3. Here, we can conclude that in the beginning, new instances are equally divided between the hypervisors but near the end, after deprovisioning of some instances, this equal distribution of instances faded away. Also the available memory on the OpenStack controller is very low. In a worst case scenario, there is a possibility that when Heat is consequently scaling-up and scaling-down, all instances will be hosted on one hypervisor. The reason for this is that Round Robin acts without any context, meaning that it doesn't know the capabilities of the hypervisors.

In both tests, there is one remarkable similarity, namely the CPU and memory usage of the OpenStack controller is much greater than the other nodes. A possible explanation for this are the many tasks an OpenStack controller has to do, such as scheduling, logging, authentication, etc. A possible fix to lower the CPU and memory usage of this OpenStack controller is to disable the hypervisor on it so it only needs to manage OpenStack and not OpenStack in combination with some instances.

VII. Conclusion

The target of this thesis was to provide an easy and quick method to evaluate new resource allocation schemes on a real cloud testbed. With OpenStack, a new allocation strategy can easily be plugged into Nova, whether or not as a filter. A whole new approach of writing schedulers for Nova is possible but requires further research. Rally and the FaaFo application will then test the new strategy while the nodejs-agent monitors everything so it can be easy evaluated afterwards.

References

[1] R. N. Calheiros, R. Rajiv, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit formodeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software - Practice and Experience*, vol. 41, no. 7, pp. 23 – 50, 2011.

[2] OpenStack, "OpenStack," 2017. [Online]. Available: https://www.openstack.org/

[3] P.-j. Maenhaut, B. Volckaert, V. Ongenae, and F. D. Turck, "RPiaaS : A Raspberry Pi Testbed for Validation of Cloud Resource Management Strategies," *proceedings of the 2017 IEEE International Conference on Computer Communications (INFOCOM 2017), Atlanta, GA, USA*, 2017.

[4] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. X. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing," *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, pp. 179–186, 2015.

[5] C. H. Benet, R. Nasim, K. A. Noghani, and A. Kassler, "OpenStackEmu - A Cloud Testbed Combining Network Emulation with OpenStack and SDN," *The 14th Annual IEEE Consumer Communications & Networking Conference*, no. January, 2017.

[6] DevStack, "Multi-Node Lab," 2017. [Online]. Available: https://docs.openstack.org/developer/devstack/guides/multinode-lab.html

[7] OpenStack, "FaaFo - LibCloud," 2017. [Online]. Available: https://developer.openstack.org/firstapp-libcloud/getting{_}started.html

[8] OpenStack, "Heat Orchestration Template (HOT) specification," 2017. [Online]. Available: https://docs.openstack.org/developer/heat/template{_}guide/hot{_}spec.html{\#}hot-spec

[9] OpenStack, "Rally," 2017. [Online]. Available: http://rally.readthedocs.io/en/latest/index.html