Assessment of Private Cloud Infrastructure Monitoring Tools
A Comparison of Ceilometer and Monasca

Mario A. Gomez-Rodriguez, Victor J. Sosa-Sosa and Jose L. Gonzalez-Compean
CINVESTAV Tamaulipas, Km. 5.5 carretera Cd. Victoria-Soto La Marina, Cd. Victoria, Tamaulipas, Mexico

Keywords: Cloud Computing, Cloud Monitoring, Resource Management, Ceilometer, Monasca.

Abstract: Cloud monitoring tools are a popular solution for both cloud users and administrators of private cloud infrastructures. These tools provide information that can be useful for effective and efficient resource consumption management, supporting the decision-making process in scenarios where administrators must react rapidly to saturation and failure events. However, the estimation of the impact on host systems in a private cloud is not a trivial issue for administrators, specially when monitoring measurements are required in reduced periods of times. This paper presents a performance comparison between two free and well supported cloud monitoring tools called Ceilometer and Monasca, deployed on a private cloud infrastructure. This comparison is mainly focused on evaluating these tools ability to obtain monitoring information in short time intervals for early detection of resource constraints. The impact of resource consumption on the performance of host systems produced by both tools was analyzed and the evaluation revealed that Monasca produced a better performance than Ceilometer for evaluated scenarios and that, according to the learned lessons in this comparison, Monasca represents a suitable option for being integrated into an adaptive cloud resource management system.

1 INTRODUCTION

Cloud computing has been widely adoped by organizations and corporations for delivering services (software, platform, infraestructure) over the internet because of the improvement of energy efficiency, hardware and software resources utilization, elasticity, performance isolation, flexibility and on-demand service schema (Aceto et al., 2013). In cloud model, applications are executed in virtual machines (VMs) that shares the resources of the physical machines (host systems) (Brinkmann et al., 2013); as a result, the management of cloud data centers managing large amount of virtual and physical machines becomes a complex task for administrators. The increment in the volumne of traffic is another important issue in the management of cloud data center. It includes topics such as the identification and resolution of conflicts related to availability, reliability and quality of service. In order to face up this challenge, precise and fine-grained monitoring activities are necessary (Aceto et al., 2013). The complexity of the monitoring activities are commonly managed by a monitoring system with funtional and non-functional components. The functional components of a monitoring system capture the state of functionality and consumption of cloud resources (Smit et al., 2013), whereas non-funcitonals include components to deploy the monitoring system in a distributed, fault-tolerant and scalable manner for monitoring a large amount of resources (Aceto et al., 2013; Fatema et al., 2014). Such tools help system administrators to obtain information such as load generated by users and the performance of the cloud computing resources (Fatema et al., 2014).

Cloud monitoring involves the collection of different metrics (e.g. CPU utilization, memory utilization, etc.), usually from both, the virtual platform (VMs, hypervisor, virtual networks, etc.) and the physical infrastructure (hosts, network, hard drives, etc.) and which are subsequently stored on a database system in order for IT operators to deal with it and make decisions that are transformed into actions.

Monitoring techniques are indispensable in order to manage large-scale cloud resources, since without an approprite monitoring some issues such as usage-based billing and elastic scaling are impossible (Fatema et al., 2014).

Cloud monitoring systems are key for improving decision-making processes in scenarios where administrators must react rapidly to saturation and failure events. The extraction of monitoring information in reduced periods of time, applying a precise and fine grained configuration is a non-trivial key

Gomez-Rodriguez, M., Sosa-Sosa, V. and Gonzalez-Compean, J.
Assessment of Private Cloud Infrastructure Monitoring Tools - A Comparison of Ceilometer and Monasca.
DOI: 10.5220/0006484503710381
In *Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017)*, pages 371-381
ISBN: 978-989-758-255-4

for early detection of resource constraint that administrators should deal with. This paper presents a performance comparison, carried out in a private cloud infrastructure, between two free and well supported cloud monitoring tools called Ceilometer and Monasca. This comparison is mainly focused on evaluating these tools ability to obtain monitoring information in short time intervals for early detection of resource constraints. As main contributions, this comparison reveals that Ceilometer is a suitable and easy to configure tool to manage the virtual layer in a private OpenStack cloud infrastructure, but when fine grained monitoring is required its Collector process does not have an efficient use of RAM in the host system, degrading its performance meaningfully. Unlike Ceilometer, Monasca is easier to configure for managing the physical layer in an OpenStack cloud infrastructure, showing that its Persister process has a better use of RAM in the host system. Moreover, in a fine grained monitoring scenario, Monasca was able to provide timely monitoring information, becoming a suitable option for being integrated into an adaptive cloud resource management system.

The rest of the paper is organized as follows: Section 2 shows the research on cloud monitoring, covering general purpose monitoring tools adapted to cloud environments and cloud specific monitoring tools. The characteristics of the cloud monitoring tools evaluated in this paper are presented in Section 3 whereas Section 4 shows a qualitative analysis of Ceilometer and Monasca cloud monitoring tools, highlighting their main features. Section 5 describes the methodology used to conduct the performance assessment of the previously described monitoring tools as well as the analysis of the results. Finally the paper ends giving the conclusions in Section 6.

2 RELATED WORK

The tools for monitoring of resources have been developed ad hoc for different computing models (e.g. servers, clusters, grid and cloud). For instance, Nagios (Nagios Enterprises, 2017; Barth, 2008) is an open-source monitoring tool for distributed systems and networks which has been adapted to cloud environments. It uses a plug-in-based model to extract information about different performance metrics (Perez-Espinoza et al., 2015b) from resources of physical and virtual machines. Nagios follows a centralized approach that produces a single point of failure and a bottleneck on the server storing the metrics, which could affect the reliability and response time in which the administrators of data centers receive monitoring information. Although Nagios can be extended using DNX in order to be distributed and to support large deployments of infrastructures (Calero and Aguado, 2015), DNX suffers from high-availability problems as "if the master server goes down, everything is down" (Nagios Enterprises, 2015). Other monitoring systems commonly used in cluster environments, where the changes in the infrastructure are not as dynamic as they are in cloud scenarios (Aceto et al., 2013) are Ganglia (Massie et al., 2004) and Collectd (Cowie, 2012).

PCMONS (Chaves et al., 2011) was designed specifically for monitoring the resources of private cloud infrastructures. The main drawback of PCMONS is its centralized architecture since the monitored data are stored and processed by a unique server, impacting directly to the scalability and fault tolerance of the system and representing a single point of failure (Perez-Espinoza et al., 2015a). Another proposal is FlexACMS (de Carvalho et al., 2013), a modular monitoring solution designed for private clouds based on general purpose monitoring solutions such as Nagios and MRTG (Oetiker, 2017). This type of tool supports the creation of monitoring slices, which are composed of a set of monitoring metrics and associated configurations used to monitor cloud slices on cloud platforms; however, fault tolerance and scalability issues are not addressed in their proposal (Perez-Espinoza et al., 2015b). Konig et al. (König et al., 2012) implemented a P2P monitoring framework to provide reliable and elastic monitoring services for gathering information across all cloud layers. Povedano-Molina et al. (Povedano-Molina et al., 2013) proposed a distributed cloud monitoring architecture called DARGOS which provides measures of the physical and virtual resources using push/pull approaches. It is composed of two main components, the Node Monitoring Agent (NMA), for collecting the resources statistics, and the Node Supervisor Agent (NSA), as a subscriber to monitoring information being published by NMA. DARGOS mainly copes with extensibility, adaptability, and intrusiveness (Aceto et al., 2013). MonPaaS (Calero and Aguado, 2015) is another cloud monitoring tool which uses OpenStack as cloud stack and the Nagios tool to monitor the cloud. It can be used by cloud providers and consumers. Also, it was relesead as open-source under a GPL license. The problem with MonPaaS is that its last version was implemented in a very old OpenStack version (Folsom) and it is not well documented. Its deployment requires that users has a great understanding of both, the source code of the monitoring tool and the OpenStack architecture and their APIs.

Ceilometer (Foundation, 2017c) is a free and well

supported cloud monitoring tool native of the OpenStack ecosystem, which represents the main option for administrators to deploy private/public cloud infraestrucutre (Foundation, 2017b). Monasca, which stands for Monitoring-at-Scale (LP, 2017a), is also included in the native OpenStack ecosystem but can be deployed in stand-alone manner. This tool deploys agents on the monitored cloud to extract monitoring data from cloud resources and delivers this information to Monasca service by using push operations. This service has gained popularity among OpenStack users as it offers a multi-tenant management environments and alarms for events and metrics, which is not completely offered by tools previously described.

Ceilometer and Monasca are becoming the prevalent monitoring tools for OpenStack users and have been used as support for big data proposals (Zareian et al., 2016) and network virtualization (Gardikis et al., 2016). Although Ceilometer and Monasca are prevalent tools among the OpenStack users and are useful as monitoring solution supporting resource management proposals, there are no available assessment comparison studies between these tools for the decision-making processes of administrations of private clouds. This paper presents useful insights and learned lessons for decision makers to take performance considerations of both tools into account when choosing the most suitable tool for specific monitoring requirements.

3 A COMPARATIVE PERFORMANCE STUDY OF CLOUD MONITORING TOOLS

In this section, we describe in more detail the characteristics of the cloud monitoring tools evaluated in this paper. We also discuss the advantages and drawbacks of both tools to establish a big picture about monitoring features of both tools, which is described as qualitative assessment focused on decision makers and private cloud administrators in Section 4.

3.1 Ceilometer

OpenStack enables the creation of private and public IaaS clouds and consists of several key services (Foundation, 2016). These services include the Telemetry service which contains the Ceilometer component to provide a data collection service across all OpenStack core components (Foundation, 2017c). The data collected by Ceilometer can be used to provide customer billing and resource tracking. Such data can be sent to different targets, for example, to MongoDB (NoSQL) or Gnocchi (time series) database. It provides four main components (Foundation, 2017c):

1. Polling agent: program to poll OpenStack services and build meters.
2. Notification agent: program to listen to notifications on message queues, convert them into events and samples and apply pipeline actions.
3. Collector: program to gather and record (e.g. in a database) event and metering data created by notification and polling agents.
4. API: service to query and view data recorded by collector.

Ceilometer provides two methods which can be used together to collect data (Foundation, 2017c):

1. Bus listener agent: takes events generated on the notification bus and transform them into Ceilometer samples. This method is supported by the ceilometer-notification agent which monitors the messages queues for notifications.
2. Polling agents: poll some API or other tool to collect data at a regular interval. The agents can be configured to poll the local hypervisor (compute-agent) or remote (central-agent) APIs (public REST APIs or host-level SNMP/IPMI daemons). The last method is less preferred due to the load it can generate on the API services.

The data collected by agents are manipulated and published in what they call pipelines. A pipeline is a set of transformers that modify the data points and transform them. Ceilometer provides different transformers which can be used to manipulate data (e. g. combine historical data or use the temporal context) in the pipeline. Such functionality is carried out by the notification agent.

The processed event and metering data captured by notification and polling agents are gathered by a program called Collector and, after the data validation (signature check) it writes the samples to the specified target: database (MongoDB or gnocchi), file or http.

In case the collected data was stored in a database (e.g. MongoDB), such stored data can be accessed using a REST API rather than by accessing the underlying database directly (until OpenStack Mitaka version). If Gnocchi is used, the data must be accessed using the Gnocchi API.

3.2 Monasca

Monasca is an OpenStack project that provides an open-source multi-tenant, highly scalable, performant, fault-tolerant monitoring-as-a-service solution.

Table 1: Cloud monitoring level comparison: Ceilometer and Monasca.

Infrastructure monitoring level	Ceilometer	Monasca
Virtual	Only configure the meters in the pipeline configuration file of each compute node that will be monitored using the Compute-agent.	Only configure the libvirt.yaml plugin on each compute node that will be monitored with the Monasca-agent. This is a bit tricky plugin since we had to modify the python source code of such agent's plugin to make it work. The problem was related with the authentication of the Nova and Neutron clients. This plugin will allow us to obtain different per-instance metrics such as cpu.utilization_perc, io.read_bytes, net.in_packets_sec, mem.used_mb, etc.
Physical	Configure an SNMP server on each Compute node that will be monitored. Configure the Central-agent to poll the remote host-level SNMP daemons running on compute nodes. Configure the hardware meters to be polled via SNMP in the pipeline configuration file of the node that is running the Central-agent (typically the controller). This is the less prefered method since it can impose more load on the nodes and, if the Central-agent goes down, the hardware meters of all compute nodes get lost.	Only configure the plugins you want to enable (e.g. cpu.yaml, disk.yaml, memory.yaml network.yaml) on each compute node we want to monitor with the monasca-agent.. Each plugin can provide different metrics such as cpu.yaml: cpu.percent, cpu.idle_perc, etc.

Table 2: Hardware characteristics of cloud hosts.

Host	Cores	RAM (GB)	Disk(s) size	Freq. (Ghz)
controller	6	31	465.8G,1.8T	2
compute6	12	62	3.2T	2.5
compute7	6	23	465.8G,931.5G,931.5G	3.07
compute8	16	62	931.5G,931.5G,931.5G,931.5G	2.6
compute9	12	62	930.4G,27.3T	2.2
compute10	24	251	2.7T	2.2
compute11	24	125	2.7T,2.7T	2.2
compute12	24	125	2.7T,2.7T	2.2

Metrics can be published to the Monasca API, stored and queried. Monasca builds an extensible platform for advanced monitoring services that can be used by both operators and tenants to gain operational insight and visibilty, ensuring availabilty and stability (LP, 2017a).

The members of the Monasca team are primarily composed of companies, organizations and individuals involved in development and deployment of OpenStack. Some of the major companies involved with developing and/or deploying Monasca are (LP, 2017a): Hewlett Packard, Enterprise Time Warner Cable (TWC), Fujitsu, Cisco, Cray, Rackspace, SAP NEC.

The Monasca API is the gateway for all interaction with Monasca. In a typical scenario metrics are collected by the Monasca Agent running on a system and sent to the Monasca API. The API then publishes the metrics to the Kafka queue. From here the Monasca Persister consumes metrics and writes them to the Metrics database. The Monasca Threshold Engine also consumes the metrics and uses them to eval-

uate alarms (LP, 2017b). At this point the metrics are in the system and can be queried using the Monasca API, either directly or through one of other components, such as the Horizon plugin or the Monasca CLI (LP, 2017b). Also there exists a configuration database used for storing information such as alarm definitions and notification methods. This database can be either MySQL or PostgreSQL (LP, 2017b).

Finally, Monasca is composed of the following main components (Foundation, 2017a):

- Monitoring Agent (monasca-agent): a Python based monitoring agent that consists of several sub-components and supports system metrics, such as cpu utilization and available memory, Nagios plugins, statsd and many built-in checks for services such as MySQL, RabbitMQ, etc.
- Monitoring API (monasca-api): a RESTful API for monitoring that is primarily focused on the following concepts and areas: metrics, statistics, alarms and notification methods. It has both Java and Python implementations avaialble.
- Persister (monasca-persister): a program which consumes metrics and alarm state transitions from the MessageQ and stores them in the Metrics and Alarms database. It has both Java and Python implementations available.
- Message Queue: a third-party component that primarily receives published metrics from the Monitoring API and alarm state transition messages from the Threshold Engine that are consumed by other components, such as the Persister and Notification Engine. Currently, a Kafka based MessageQ is supported.
- Metrics and Alarms Database: a third-party component that primarily stores metrics and the alarm state history.
- Config Database: a third-party component that stores a lot of the configuration and other information in the system. Currently, MySQL is supported.
- Monitoring Client (python-monascaclient): a Python command line client and library that communicates and controls the Monitoring API. The Monitoring Client also has a Python library, "monascaclient" similar to the other OpenStack clients, that can be used to quickly build additional capabilities. The Monitoring Client library is used by the Monitoring UI, Ceilometer publisher, and other components.
- Monitoring UI: A Horizon dashboard for visualizing the overall health and status of an OpenStack cloud.

4 A QUALITATIVE COMPARISON OF CEILOMETER AND MONASCA

As we described in the previous sections, Ceilometer and Monasca are two different monitoring tools designed to monitor an OpenStack cloud. We decided to compare such tools since OpenStack is one of the most used cloud stack in both literature and open-source community.

In Table 1 we can see a comparison between the virtual and physical infrastructure monitoring levels of both Ceilometer and Monasca. On one hand we can see that it is easy to configure the virtual infrastructure monitoring on Ceilometer, whilst it has some drawbacks to monitor the physical infrastructure. To obtain the hardware metrics of the physical nodes it has to run a centralized server (central-agent) on one node to poll the SNMP servers running on compute nodes. Such approach has the drawback that, if the central-agent goes down, the hardware meters of all compute nodes get lost, in addition to imposing an extra load on the physical nodes. On the other hand, unlike Ceilometer, in Monasca it is easy to monitor the physical infrastructure. Only it is necessary to configure the plugins to enable the different metrics it provides. However, problems arise when we had to enable the virtual infrastructure monitoring via the libvirt.yaml plugin. Such plugin allows to obtain different per-instance metrics. We had to modify the python source code of such agent's plugin to make it works. The problem was related with the authentication of the Nova and Neutron clients.

A summary of the most important components of Ceilometer and Monasca are depicted in the mind map shown in Figure 1. It also shows some relevant properties that can be varied to perform different tests, for example to vary the number of workers or processes in charge of collecting metrics in the Collector or Persistert agent in Ceilometers or Monasca respectively.

5 PERFORMANCE EVALUATION AND RESULTS

In this section, we describe the methodology used to conduct the performance assessment of Ceilomenter and Monasca tools in a private cloud as well as the analysis of the results of this evaluation.

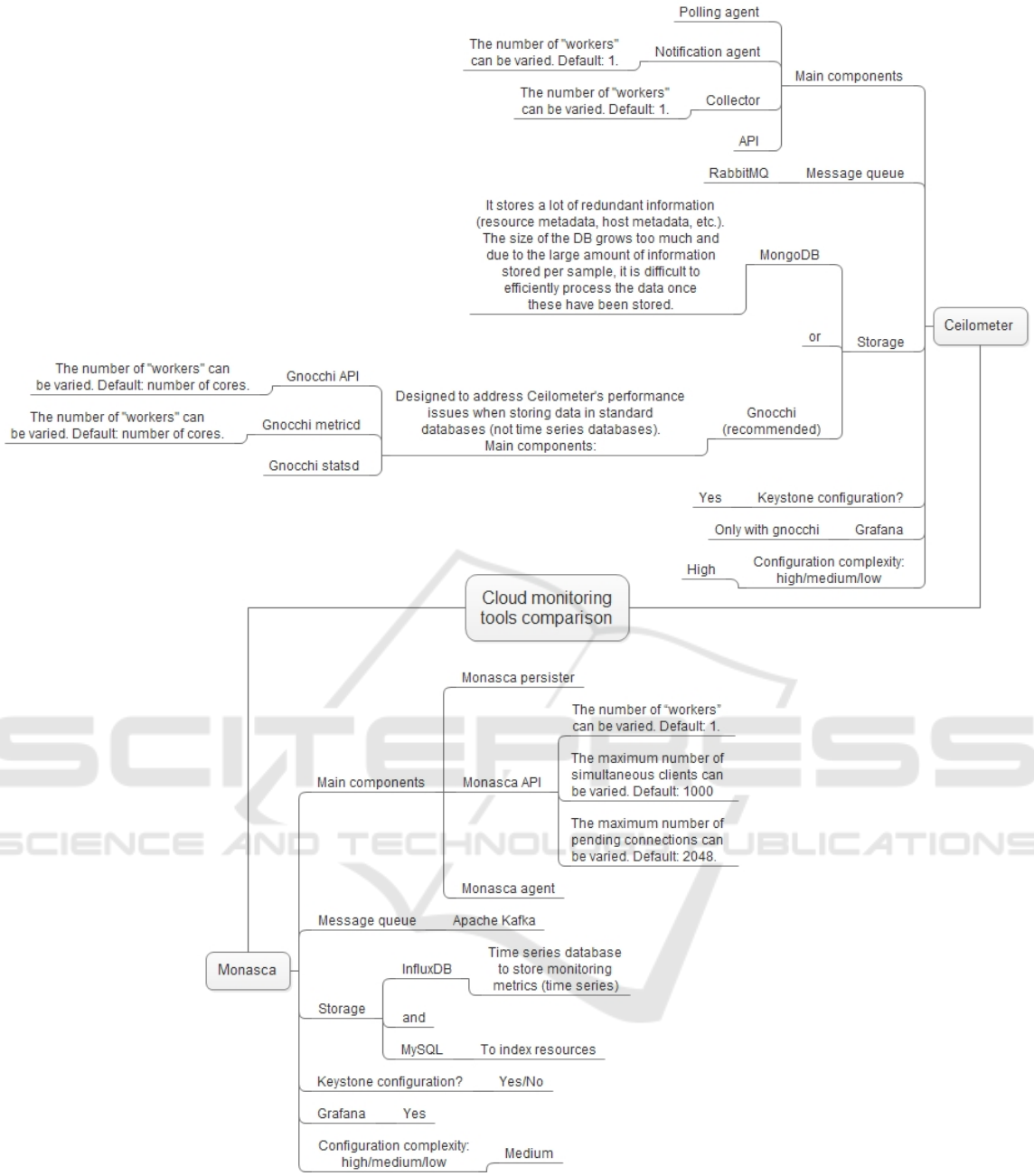

Figure 1: Ceilometer and Monasca mind map.

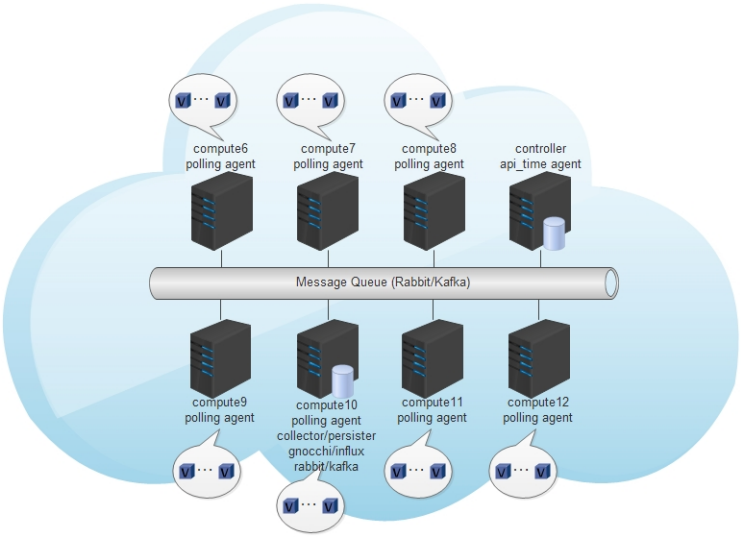

Figure 2: Cloud testbed.

5.1 Evaluation Methodology

A set of experiments were executed in a real cloud computing environment and the cloud deployed for this evaluation included eight nodes. Seven for Compute and one Controller (see the characteristics of each node in Table 2). Ceilometer and Monasca were used to monitor the virtual and physical infrastructure. Figure 2 shows that Compute6 to Compute12 run the polling agents. In the Compute10 node is also located the Ceilometer Collector and Monasca Persister processes, whose main tasks are to obtain (through message queues) the monitoring information extracted by polling agents. This information is stored in the Gnocchi database (Ceilometer configuration) and the Influx database (Monasca configuration). The message queues used by Ceilometer Collector and Monasca Persister are Rabbit and Kafka respectively.

Depending on the configuration of the monitoring system, some times, the most recent measurements may have not been processed and stored when high frequency monitoring was activated. In this scenarios, measures could not be extracted in a timely manner with the respective API of the tool, implying that the current stored data were obsolete, providing an inconsistent landscape of the cloud state. In order to find which of the two monitoring tools were able to obtain the most recent measurements (almost real time), we measured the delay between the time (UTC) of the Controller node and the timestamp of the last measurement of one metric (CPU % usage of a virtual machine) stored in the database of the monitoring system. All this using the respective API of each system.

The total number of tests performed with both monitoring tools were 50 (25 for each monitoring tool), each one executed during 1 hour, with different polling interval and different number of workers/processes of the program in charge of taking the data from the message queue to send them to the database. Table 3 shows the different configurations executed on the cloud. It is worthy to mention that in this evaluation the monitoring information that Ceilometer and Monasca were requiring to polling agents installed in the private cloud has to be with CPU, RAM, Hard Disk and Network consumption of every node in the cloud (including VMs metrics from the hypervisors). In this evaluation, for every request of monitoring information, Ceilometer extracted a total of 62 metrics and Monasca 92 metrics from the different plugins included in their respective polling agents in our private cloud scenario.

Table 3: Test configurations.

Monit. tool	# Workers	Polling intervals	# Configs.
Ceilometer	1, 2, 4, 8, 16	10, 30, 60, 90, 120	25
Monasca	1, 2, 4, 8, 16	10, 30, 60, 90, 120	25
Total			50

5.1.1 Metrics

In order to conduct our evaluation, we collected two basic performance metrics:

- % RAM: This metric allows us to determine the RAM consumption impact when Ceilometer Collector and Monasca Persister processes use different configurations and vary the time interval for extracting monitoring information.
- Delay: This metric represents the delay produced by Ceilometer and Monasca in the extraction of monitoring data.

5.2 Analysis of Results

During the execution of the previously described tests, we analyzed the physical resource consumption of both monitoring systems and we found that Ceilometer has memory problems with the agent (Collector) in charge of sending the monitored data to the database, whilst Monasca didn't present problems with its analogue agent (Persister). However, Monasca presented some limitations when the time interval configured in polling agents was less than 10 seconds, its polling agents suddenly crashes. This behavior was caused by delays in the execution of plugins used by the polling agents, since some plugins are still in execution measuring metrics when the agents tries to poll again according to their interval rate. For this reason, we use in this evaluation time intervals of 10 or more seconds.

We started our evaluation running Ceilometer as our monitoring tool. As we can see in Figure 3 (being the horizontal axis the time and the vertical axis the RAM %), using a fine-grained polling interval of 10 seconds and 1 Collector worker causes that the RAM usage increases very fast (Figure 3a), whilst using longer polling intervals (Figures 3b-d) the usage of RAM decreases. We can note that using the 10 polling frequency provokes that the RAM usage increases 1 percent every 0.28 hours wich means the monitoring system is consuming more than 2.5 GB of RAM per hour, running the host system out of memory in almost 28 hrs. This situation implied that for a test scenario as described in Figure 2, it was necessary to start again the monitoring process every day, limiting the continuous monitoring activity to 1 day periods. On the other hand, the RAM consumption of Monasca was negligible (almost 0) when using a similar number of processes (number of workers in Ceilometer) in its Persister agent, reason why these graphs were not included in the paper. The intuition behind this behavior is that Monasca Persister is freeing the memory used after completing every request for monitoring information and Ceilometer Collector is keeping this information in memory.

Figure 4 shows the timestamp delay comparison of the following 10 configurations of both monitoring tools: Collector workers or Persister processes: 1; polling intervals: 10, 30, 60, 90 and 120 seconds. Polling intervals define the time interval that will be used by polling agents to get monitoring information from a specific resource, e.g. CPU, RAM, HD or Network. The delay was calculated as the difference between the current timestamp of the controller node and the timestamp of the last measurement of one metric (e.g., CPU % usage in a VM) stored in the database of the monitoring system. We can see in Figure 4a that Ceilometer's delay has a linear growth and the smaller the polling interval (fine-grained monitoring) the greater the timestamp delay, e.g., in this case the 10s configuration obtained the greater delay along the time series whilst the 120s configuration obtained the smaller delay. Monasca on the other hand presented a stabilized behavior. In Figure 4b it can be seen that the delay is almost the same along the time series, generating only an approximate increase of 5 seconds over the polling interval, e.g., the 10s configuration obtained a delay of about 15 seconds along the time series whilst the 120s configuration obtained a delay of about 125 seconds, confirming what was previously said. It is worthy to mention that every time Controller requests for a specific metric (e.g., CPU % usage), Ceilometer or Monasca responses with a time series of that metric (list of all measures with its corresponding timestamps taken since the monitoring process started). This means that the time series is short at the the initial phase of monitoring activity and is getting larger with time.

Delay of the rest of executed configurations of both monitoring tools is shown in Figure 5. In the case of Ceilometer (Figures 4a and 5a) we can see that the worst configurations were those with the smaller polling interval (10s), whereas the bests configuratons were those with the greater polling interval (120s); and the two better configurations were those with four and sixteen workers and 120s polling interval (depicted in Figure 5a as 4_w-p_120s and 16_w-p_120s respectively) with delay of about 500s along the time series. This means that with the best Ceilometer's configuration we obtained an approx. delay of 8 minutes during 1 hour. Monasca on the other hand obtained a better performance (see Figure 5b), following the same behavior of the first 5 configurations previously analyzed (Figure 4b) in which the delay only increased 5 seconds approximately on the polling interval along the time series during 1 hour. This shows that Monasca can provide a more precise landscape of the cloud since it had a better performance (smaller delay) in all the executed tests.

Currently the entire time series is retrieved to get the last measurement of the metric. Since the moni-

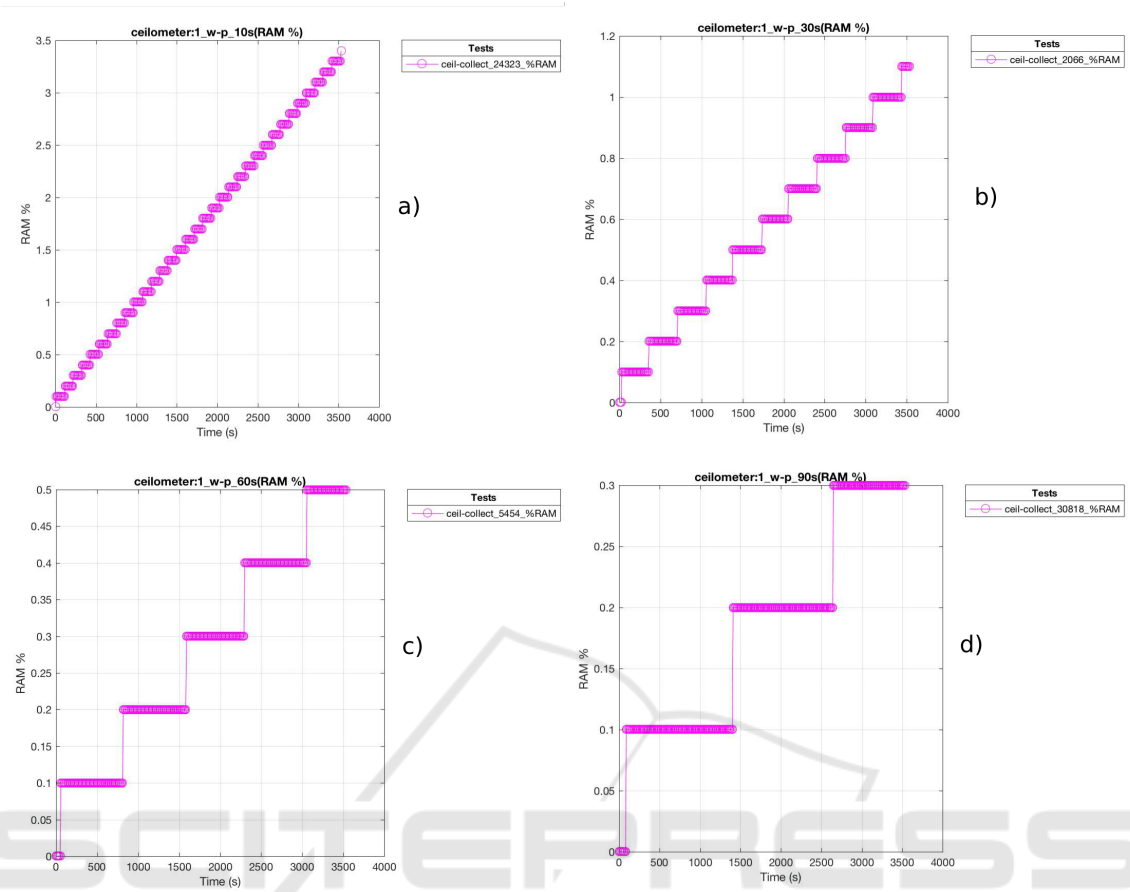

Figure 3: One collector worker in Ceilometer execution. Polling intervals: a) 10s, b) 30s, c) 60s and d) 90s.

Table 4: Experiments summary.

Cloud stack	Monitor	Feature measured	Resource measured	Agent name	Memory consumption	Delay's growth
OpenStack	Ceilometer	Timestamp delay	RAM	Collector	High	Linear
	Monasca			Persister	Low	Stable

toring time of the experiments only extends to 1 hour, such a behaviour is not a problem when we get the time series. However, if we decided to run more experiments with a longer monitoring time it would be necessary to get only a subset of the last measurements. Taking into account the previous observation, a more extended set of experiments increasing the running time of the monitoring tools can be performed to confirm our findings.

Finally, Table 4 shows a summary of the previously described experiments and results. As we can see, an OpenStack cloud was constructed to run the set of experiments, measuring the RAM consumption of the collectors programs of the monitoring tools (Ceilometer Collector and Monasca Persister) and the timestamp delay of the last monitored measurement; the former allows us to detect possible bottlenecks during the data collection stage and the latter to identify which monitor enables a timely resource monitoring.

6 CONCLUSIONS

This paper provides a relevant information about results obtained from a qualitative and quantitative evaluation of two monitoring tools (Ceilometer and Monasca), which are popular among the administrators of cloud deployed by using OpenStack. The qualitative comparison of both tools shows a big picture of the main functionality characteristics, whereas the quantitative evaluation provides useful insights about how the collector agents impact resource consumption in host systems in a private OpenStack cloud en-

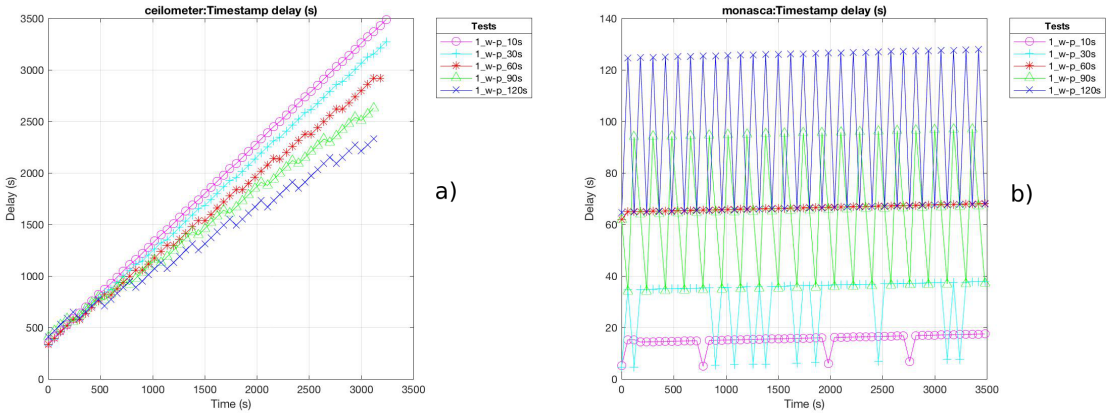

Figure 4: Delay in extraction of monitoring data. Collector workers/persister processes: 1. Polling intervals: 10s, 30s, 60s, 90s, 120s. a) Ceilometer and b) Monasca.

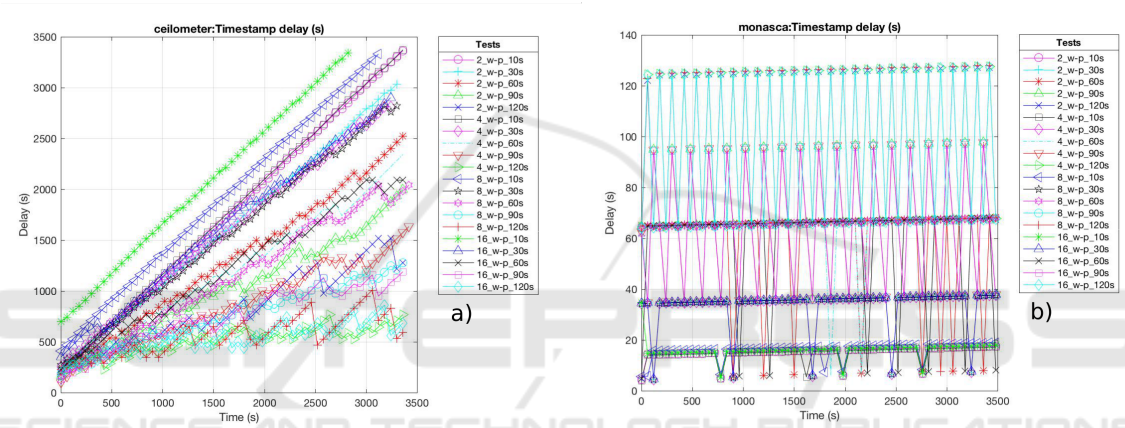

Figure 5: Delay in extraction of monitoring data. Collector workers/persister processes: 2, 4, 8, 16; polling intervals: 10s, 30s, 60s, 90s, 120s. a) Ceilometer and b) Monasca.

vironment when high frequency sensing of node resources is required. As expected, the consumption of resources (especially memory) in the host server/node meaningfully increases when sensing is carried out in short periods of time. The obtained results of this evaluation make Monasca as an easier to use and suitable tool for monitoring private cloud infrastructure than Ceilometer when high frequency sensing is required. Ceilometer exhibits memory problems causing delays when obtaining monitoring data; as a result, the metrics stored on Ceilometer database were out-of-date. This is a relevant finding for decision makers to take into account when choosing a monitoring tool for a private cloud infrastructure. This is also relevant for an ongoing work in which an adaptive cloud resource management system is being developed and that will include Monasca as data monitoring provider based on learnt lessons in this study. The adaptive system will apply data analysis and prediction algorithms that allow the cloud manager platform to assign and reallocate resources in a dynamically way, without user intervention.

ACKNOWLEDGEMENTS

This work was partially supported by the sectoral fund of research, technological development and innovation in space activities of the Mexican National Council of Science and Technology (CONACYT) and the Mexican Space Agency (AEM), project No.262891.

REFERENCES

Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Barth, W. (2008). *Nagios: System and network monitoring*. No Starch Press.

Brinkmann, A., Fiehe, C., Litvina, A., LÃijck, I., Nagel, L., Narayanan, K., Ostermair, F., and Thronicke,

W. (2013). Scalable monitoring system for clouds. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 351–356.

Calero, J. M. A. and Aguado, J. G. (2015). Monpaas: An adaptive monitoring platformas a service for cloud computing infrastructures and services. *IEEE Transactions on Services Computing*, 8(1):65–78.

Chaves, S. A. D., Uriarte, R. B., and Westphall, C. B. (2011). Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, 49(12):130–137.

Cowie, B. (2012). Building a better network monitoring system. *Bachelor Report Degree, Computing and Mathematical Sciences, University of Waikato, Hamilton, New Zealand*.

de Carvalho, M. B., Esteves, R. P., da Cunha Rodrigues, G., Granville, L. Z., and Tarouco, L. M. R. (2013). A cloud monitoring framework for self-configured monitoring slices based on multiple tools. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 180–184.

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.

Foundation, O. (2016). Openstack installation guide for red hat enterprise linux and centos, http://docs.openstack.org/mitaka/install-guide-rdo/.

Foundation, O. (2017a). Monasca, https://wiki.openstack.org/wiki/monasca.

Foundation, O. (2017b). Openstack open source cloud computing software, https://www.openstack.org/.

Foundation, O. (2017c). Welcome to the ceilometer developer documentation!, http://docs.openstack.org/developer/ceilometer/.

Gardikis, G., Koutras, I., Mavroudis, G., Costicoglou, S., Xilouris, G., Sakkas, C., and Kourtis, A. (2016). An integrating framework for efficient nfv monitoring. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 1–5.

König, B., Calero, J. A., and Kirschnick, J. (2012). Elastic monitoring framework for cloud infrastructures. *IET Communications*, 6(10):1306–1315.

LP, H.-P. E. D. (2017a). Monasca: about, http://monasca.io/root/about/.

LP, H.-P. E. D. (2017b). Monasca: architecture, http://monasca.io/root/architecture/.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840.

Nagios Enterprises, L. (2015). Nagios xi - using dnx for load-balancing, https://assets.nagios.com/downloads/nagiosxi/docs/using-dnx-for-load-balancing-with-nagios-xi.pdf.

Nagios Enterprises, L. (2017). Nagios, http://www.nagios.org/.

Oetiker, T. (2017). Mrtg: Multi router traffic grapher., http://oss.oetiker.ch/mrtg/.

Perez-Espinoza, J. A., Sosa-Sosa, V. J., and Gonzalez, J. L. (2015a). Distribution and load balancing strategies in private cloud monitoring. In *2015 12th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 1–6.

Perez-Espinoza, J. A., Sosa-Sosa, V. J., Gonzalez, J. L., and Tello-Leal, E. (2015b). A distributed architecture for monitoring private clouds. In *2015 26th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 186–190.

Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., and Foschini, L. (2013). Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *Future Generation Computer Systems*, 29(8):2041 – 2056. Including Special sections: Advanced Cloud Monitoring Systems & The fourth {IEEE} International Conference on e-Science 2011 - e-Science Applications and Tools & Cluster, Grid, and Cloud Computing.

Smit, M., Simmons, B., and Litoiu, M. (2013). Distributed, application-level monitoring for heterogeneous clouds using stream processing. *Future Generation Computer Systems*, 29(8):2103–2114.

Zareian, S., Fokaefs, M., Khazaei, H., Litoiu, M., and Zhang, X. (2016). A big data framework for cloud monitoring. In *Proceedings of the 2Nd International Workshop on BIG Data Software Engineering*, BIGDSE '16, pages 58–64, New York, NY, USA. ACM.