Efficient Configuration of Monitoring Slices for Cloud Platform Administrators

Márcio Barbosa de Carvalho*, Rafael Pereira Esteves*, Guilherme da Cunha Rodrigues*,
Clarissa Cassales Marquezan†, Lisandro Zambenedetti Granville*, Liane Margarida Rockenbach Tarouco*
*Institute of Informatics – Federal University of Rio Grande do Sul – Porto Alegre, Brazil
Email: {mbcarvalho, rpesteves, gcrodrigues, granville}@inf.ufrgs.br, liane@penta.ufrgs.br
†Paluno, University of Duisburg-Essen, Germany
Email: clarissa.marquezan@paluno.uni-due.de

***Abstract*—Monitoring is an important issue in cloud environments because it assures that acquired cloud slices attend the user's expectations. However, these environments are multi-tenant and dynamic, requiring automation techniques to offload cloud administrators. In a previous work, we proposed FlexACMS: a framework to automate monitoring configuration related to cloud slices using multiple monitoring solutions. In this work, we enhanced FlexACMS to allow dynamic and automatic attribution of monitoring configuration tasks to servers without administrator intervention, which was not available in previous version. FlexACMS also considers the monitoring server load when attributing configuration tasks, which allows load balancing between monitoring servers. The evaluation showed that enhancements reduced FlexACMS response time up to 60% in comparison to previous version. The scalability evaluation of enhanced version demonstrated the feasibility of our approach in large scale cloud environments.**

***Index Terms*—Cloud computing, monitoring configuration.**

I. Introduction

Infrastructure as a Service (IaaS) [1] is a cloud model that offers *cloud slices* to cloud users. These cloud slices are indeed virtualized computational resources (*e.g.*, compute, storage, network) that need to be closely monitored to guarantee the quality of service expected by IaaS cloud users. In addition to cloud slices, Infrastructure Providers (InPs) also offer Monitoring as a Service (MaaS) [2]. In this case, cloud users can also subscribe to personalized monitoring services associated with their cloud slices. InPs also need monitoring results to enable efficient physical resource utilization; to avoid wasting expensive resources that are over provisioned. The cloud monitoring therefore directly impacts the InP revenue: by keeping cloud user's subscription guarantying their expectations, earning extra revenue provided by MaaS, and enabling efficient resource utilization of the cloud infrastructure.

When a cloud user requests a new cloud slice, the monitoring support for this new cloud slice must be also configured. We define the set of monitoring support (*e.g.*, monitored metrics and required configuration) associated with one cloud slice as a *monitoring slice* [3]. Current monitoring solutions cannot satisfy all cloud platform administrator requirements because these solutions cannot support all the possible monitoring requirements that different cloud slices might have. This means that cloud slices need to be monitored by a set of monitoring solutions [4] instead of a single unified solution. In addition, some monitoring solutions required by cloud slices are not integrated with cloud platforms. As a consequence, cloud platform administrators need to manually configure solutions to monitor metrics of monitoring slices or develop specialized and individual scripts for this configuration.

In a previous work [3] we proposed a framework called Flexible Automated Cloud Monitoring Slices (FlexACMS) to address the problem of automatically setting up monitoring slices. Despite the increase in the automation this work introduced, it remained a task of the cloud platform administrator to manually configure which set of metrics from a monitoring slice would be attributed, *i.e.* mapped, to which monitoring server. For instance, CPU metrics of monitoring slice#1 are configured to be responsibility of monitoring server Nagios#A, while CPU metrics of monitoring slice#2 are responsibility of Nagios#B. As consequence, there was no automatic mechanism to better distribute the configuration load among the monitoring servers during the configuration task, *i.e.*, the process of creating the monitoring slices.

In this paper, we extended the previous version of the FlexACMS framework solving the two aforementioned drawbacks and enhanced its automation and performance on configuring monitoring slices. The three main contributions of this paper are: *(i)* the dynamic and automatic attribution of configuration tasks to monitoring servers in a Message-Queuing fashion; *(ii)* load balancing among monitoring servers when attributing configuration tasks; and *(iii)* exploring the intrinsic parallelism in building monitoring slices.

We provide an analysis of the overhead introduced by the FlexACMS in terms of response time during the process of automatically configuring the support for new cloud users of IaaS and monitoring services (*i.e.*, creation of a cloud slice and its required monitoring slice). In the evaluated IaaS scenario, OpenStack [5] was used as cloud platform to create cloud slices; while Nagios [6] and MRTG [7] are the monitoring solutions associated with the monitoring slice to be configured by FlexACMS. The results show that the enhanced version has little impact on the response time during the automatic configuration of IaaS and monitoring services of InPs. We also evaluated the scalability of FlexACMS in terms of response time when varying the number of: cloud slices in place,

monitoring slices to be created, and metrics per monitoring slice. This scalability evaluation demonstrated the feasibility of our approach for large scale cloud environments.

This paper is organized as follows. In Section 2, the state-of-the-art on cloud computing monitoring is reviewed. In Section 3, we review the previous version of FlexACMS framework and its limitations. In Section 4, we detail the new features and enhancements proposed for the FlexACMS framework. In Section 5, we comment the results of both functional and scalability evaluations. In Section 6, we discuss our conclusions and future work.

II. Related Work

A variety of monitoring solutions are available for gathering updated status of cloud slices and applications deployed in these cloud slices. For instance, PCMONS [8] is a monitoring system for IaaS cloud models that uses a layered architecture to handle heterogeneity in the infrastructure, which provides an abstraction that allows the monitoring integration of different cloud platforms. Global Monitoring systEm (GmonE) [4] is also a monitoring solution for IaaS cloud models which provides a complete approach to cloud monitoring including metrics to users and administrators. However, these solutions need to be improved to have the ability of monitor different type of resources (*e.g.*, web servers, databases, applications).

InPs and cloud platforms also offer monitoring solutions to their costumers, such as Amazon CloudWatch [9] and Ceilometer [10]. Amazon CloudWatch is the monitoring service for Amazon Web Services (AWS) [11] that offers predefined basic metrics and allows user-defined metrics that are charged apart in a MaaS manner. Ceilometer aims to be the monitoring infrastructure for OpenStack platform to collect and share monitoring data with external consumers. However, these solutions are specific to these InP services and platforms, which hamper service/platform choice and migration.

In addition, traditional monitoring solutions can also be employed in cloud computing monitoring, such as Nagios [6] and MRTG [7]. These solutions are usually employed in heterogeneous environments, which are similar to cloud computing. For instance, Nagios is used for monitoring of different type of resources (*e.g.*, web servers, databases, applications) that are also deployed in cloud computing. Furthermore, the administrator's know-how in these solutions facilitates their use and extensibility. However, traditional monitoring solutions are not integrated with cloud platforms to be configured automatically, which requires manual configuration, scripting techniques, or discovery process. The latter is an expensive approach, especially in cloud computing environments where new cloud slices arrive and leave in a dynamic way.

There are solutions that monitor all layers of a cloud application stack, such as Runtime Model for Cloud Monitoring (RMCM) [12] and GmonE [4]. Other examples of cloud monitoring solutions include: the mOSAIC framework [13] that allows the development of cloud monitoring systems through the mOSAIC API, and the RESTFul Cloud Management System (RESTFul CMS) [14] that includes a cloud monitoring system based on RESTful Web services.

Aceto *et al.* [15] provide a wide study about available cloud monitoring solutions. Beyond, they also analyzed the cloud monitoring properties and requirements that monitoring solutions need to provide. Through their study we can conclude that there is not a single solution that addresses all monitoring properties and requirements. Thus, we believe that integrating multiple monitoring solutions cannot be ignored as an approach to build cloud monitoring systems.

There are also solutions like Puppet [16] and Chef [17] that could be used to automate configuration in general. Their clients are installed on server/hosts that retrieve configuration from a centralized configuration database. In cloud monitoring configuration the inverse behavior is expected, when a cloud slice (host) is created, the monitoring servers should retrieve new configurations. Thus, these solutions must be adapted to be appropriated for cloud monitoring configuration.

III. Review on FlexACMS

In this section, we review the key concepts and elements of FlexACMS framework introduced in our previous work [3]. Then, we discuss the limitations that motivated the enhancements that we propose now in this paper. The most important concept introduced by our framework is the *Monitoring Slice*. It reflects all monitoring information about a cloud slice, which is composed by both metrics related to a cloud slice and by the corresponding monitoring configuration to perform the collection. Multiple monitoring solutions might be used to collect these metrics [4]. Examples of monitoring solutions that can be combined in the collection of information from a cloud slice are: Nagios [6], MRTG [7], and Ceilometer [10].

The previous version of the FlexACMS framework created the support for automating the process of configuring the monitoring slice. The elements of the previous version of framework are: *Gatherers*, *FlexACMS Core* and *Configurators*. The *Gatherers* are responsible for collecting information about cloud slices hosted in cloud platforms and for sending this information to the Framework Core element through the REST Web service. The *FlexACMS Core* is responsible for processing the information about cloud slices received from gatherers. It stores the received cloud platform information to enable the detection of changes (*e.g.*, cloud slice creation). Based on the detection of these changes, the *FlexACMS Core* triggers the *Configurators* to build the corresponding monitoring slice for new detected cloud slice. Internally the *FlexACMS Core* was composed of three modules: REST Web Service, that receives the information from *Gatherers*; Change detection, responsible for comparing the stored information of created monitoring slices with the information about the existent cloud slices; and the Configuration Executor, that was designed to trigger the *Configurator*, passing as parameters the information needed to set up the monitoring slice. Finally, the *Configurator* is responsible for conducting the process of setting up the monitoring slices by configuring the monitoring servers that will collect the information associated with such slice.

Previous FlexACMS version freed cloud administrators of configuring each one of the monitoring servers themselves to support the collection of information about each new cloud slice. However, the cloud administrator still had to statically configure the set of metrics to be monitored by each monitoring server, *e.g.*, CPU metrics of monitoring slice of type #1 will be monitored by server Nagios#A, and CPU metrics of monitoring slice type #2 will be monitored by Nagios #B.

In the previous version of FlexACMS, the attribution of configuration tasks was done statically, as discussed above. We define the term attribution of the configuration tasks to denote this process of defining which monitoring server will be responsible for receiving request for setting up the monitoring slice support for a given type of monitoring slice. Previous FlexACMS version also did not take into account that the load of monitoring servers can change dynamically. Thus, the manual and static attribution of configuration tasks could lead to unbalanced distribution of the configuration load among the monitoring servers. In turn, this leads to potential loses of performance in the monitoring support.

IV. Enhanced FlexACMS Framework

In this paper, we propose the enhanced FlexACMS framework that is able to tackle problems related to attribution of configuration tasks to monitoring servers. To tackle the aforementioned problems and leverage the automation of FlexACMS, we introduced two major features:

- Dynamic and automatic attribution of configuration tasks in a Message-Queueing fashion: enables the automatic mapping between type of metrics to be monitored in a monitoring slice and the type of monitoring server to consume the request for configuration of the metric. This means that there is not a static specific monitoring server that will consume a request for configuration of a monitoring slice. Now any monitoring server can register to receive requests of a certain type based on the attributes configured by the cloud administrator.
- Load balancing during the configuration of the monitoring slices: enables the monitoring servers to volunteer to receive a task (i.e., setting up the monitoring slice) when they have capacity for doing so. The decision of registering can be taken based on different criteria. In this paper, we consider the load of the monitoring server.

Our previous FlexACMS architecture [3] was single-threaded, thus the configuration of monitoring slices was handled as a large task and all required steps were performed one after another. This approach did not consume much computational resources, but it did not also explore parallelism in performing these tasks and the parallel architecture of current computers. We thus extended FlexACMS to break the task of configure monitoring slices into several small tasks.

A. New architecture

The enhanced FlexACMS framework has significant structural changes in its core elements. The new architecture is illustrated in Figure 1, where all gray elements denote the changes introduced in the new version of the framework. Along this section, we detail the queue and workers components introduced in enhanced FlexACMS to enable the new features. Other components such as Gatherers, REST Web Service, and Configurators were briefly presented in Section III, and they were not modified in the enhanced version.

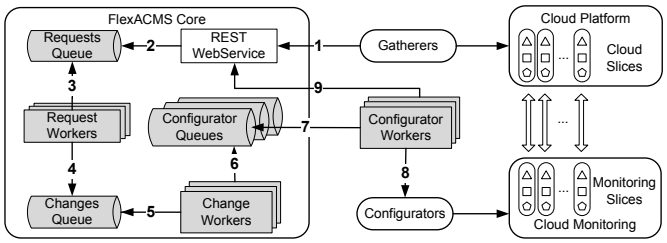


Fig. 1. FlexACMS architecture

FlexACMS Core receives information about cloud slices from gatherers through the REST Web Service (Figure 1, step 1). The FlexACMS Core performs tasks over the received information until triggering appropriate Configurators (steps 2-6). These tasks are processed by queue/workers to explore the intrinsic parallelism of these tasks. Firstly, the received information is queued in the *Requests Queue* (step 2) to be processed by a *Request Worker* (step 3). Request Workers are responsible for detecting changes between received information and previously stored information about the cloud slices deployed in the cloud platform. Each change detected is queued in the *Changes Queue* (step 4) to be processed by a *Change Worker* (step 5). Change Workers are responsible for evaluate whether detected change satisfies all Configurator's interests and conditions (see Table I), which are rules predefined by the cloud administrator. This means that these rules are checked to determine whether a new monitoring slice should be created for the new cloud slice. When all these rules are satisfied, the Change Worker queues a configurator call in the appropriated *Configurator Queue* (step 6) indicated by the cloud administrator. The configurator call has an identification and the command that executes the Configurator within all parameters necessary to build the monitoring configuration.

Configurator Queues represent pools of monitoring servers that share some characteristics. Table I shows these characteristics, which we call configurator attributes, and they include the queue where the configurator calls will be placed. In this example, we have two Configurator Queues: nagios_basic and nagios_platinum, which represents the pool of Nagios servers for Basic or Platinum MaaS subscriptions, respectively. The appropriated queue is selected according to the MaaS slice subscription stored in cloud slice attribute `@slice.MaaS`.

Configurator Workers are responsible for retrieving configurator calls from appropriated Configurator Queues (Figures 1 and 2, step 7). The Configurator Worker executes the configurator (step 8) and stores its execution status and output to send to FlexACMS Core through the REST Web service (step 9). This information is used by cloud administrators for future analysis or debugging. Configurator Workers are aware

TABLE I
EXAMPLE OF CONFIGURATORS ATTRIBUTES

Configurator	Attr.	Value
Name: nagios_host_basic Queue: nagios_basic	Int. Cond.	New Slice @slice.MaaS =~ /basic/
Name: nagios_cpu_basic Queue: nagios_basic	Int. Cond. Cond.	New Resource @resource.identifier =~ /CPU/ @slice.MaaS =~ /basic/
Name: nagios_host_plat Queue: nagios_platinum	Int. Cond.	New Slice @slice.MaaS =~ /platinum/
Name: nagios_cpu_plat Queue: nagios_platinum	Int. Cond. Cond.	New Resource @resource.identifier =~ /CPU/ @slice.MaaS =~ /platinum/

of the monitoring server load. Thus, they can decide stop retrieving configurator calls while the server load is unacceptable (Figure 2, phase 10). When the server load remains to an acceptable value, the Configurator Worker decides to consume configurator calls again (phase 11).

Configurator Queues and Configurator Workers enabled the two major features of enhanced FlexACMS. Firstly, the dynamic and automatic attribution of configuration tasks is achieved when the configurator calls are attributed to pools of monitoring servers (Configurator Queues) that are selected accordingly to predefined rules. In the previous version, the configurator tasks were statically attributed to a monitoring server. Secondly, load balancing is achieved when Configurator Workers concurrently consume the same Configurator Queue. Figure 2 presents the concurrency for configurator calls of two Nagios servers (Nagios#A and Nagios#B). This approach also enables that servers stop receiving monitoring tasks based on its load (Figure 2, phase 10) because Configurator Workers are aware of the server load.

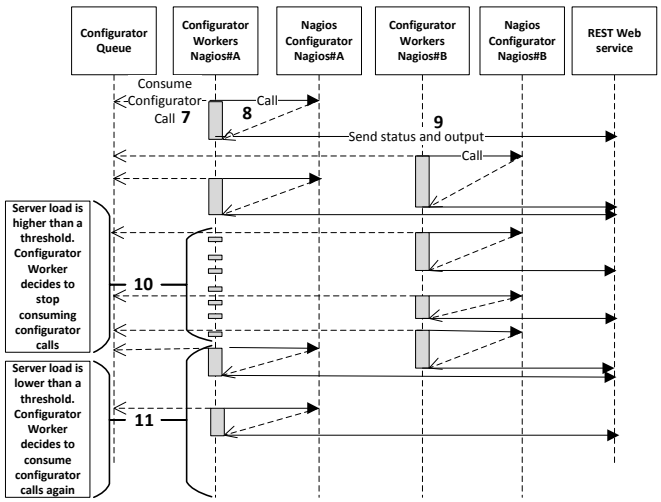


Fig. 2. Sequence Diagram for enhanced FlexACMS

B. Implementation details

We developed a gatherer to collect information about cloud slices for OpenStack [5] platform. The OpenStack gatherer uses several web services from OpenStack API [18] to retrieve information about cloud slices hosted in the platform. The response is organized according to the REST Web service definition. The OpenStack gatherer was developed in Python.

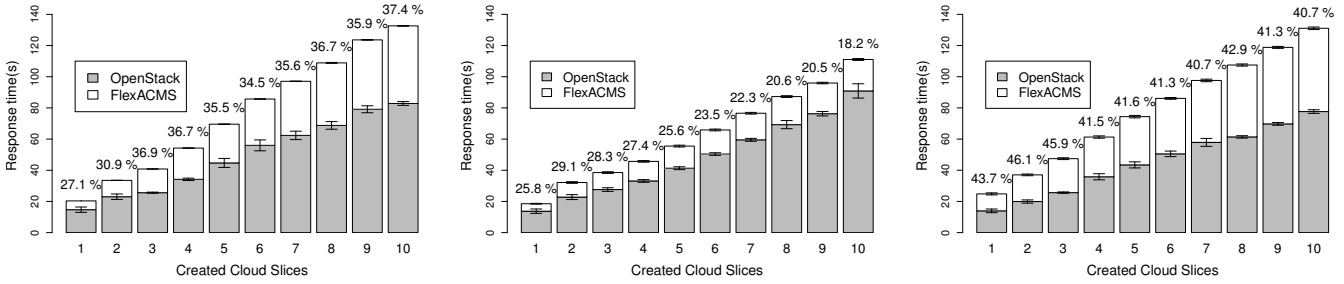
The FlexACMS Core was developed using the Ruby on Rails (RoR) framework which facilitates the development of REST Web services. RoR also facilitated the development of web-based interface used by administrators to interact with the FlexACMS. MySQL 5.5 is used as database to store cloud platform information. As the monitoring is performed by monitoring solutions and it is not performed by FlexACMS, we believe that a relational database is appropriated to this context. The queues are based on a RoR library called Resque that uses Redis, a key/value database, as backend to store the information related to queues and workers. Resque also allows the execution of workers based on Ruby code, which we used to develop Request Workers and Change Workers. Resque allows administrators to adjust the number of workers that will consume each queue. Based on this, the administrators can adjust the number of workers according to the capacity of the server hosting the FlexACMS framework (i.e, the framework server) and their response time requirements.

We developed the Configurator Worker using Perl, which consumes configurator calls within the queues directly from Redis. To be aware of the server load, the Configurator Worker reads the file `/proc/loadavg` that stores the average load of the server. We also developed a bash script to automate the initialization of configurator workers. This bash script accepts as arguments the number of workers that must be initialized, the queue name that they will consume, and the maximum load threshold that is acceptable in that server. By this manner, the administrators can also adjust the number of workers and the server load acceptable according to the monitoring server capacity and response time requirements.

Finally, we developed configurators for both Nagios and MRTG using Perl scripts. Nagios uses distinct configurations for host and service/resource status. We then define a configurator to reflect the creation of new slices (hosts) in Nagios, and a configurator for each monitored resource. We also developed a configurator for executing scripts that configure MRTG: `cfgmaker` and `indexmaker`, used to create configuration and index page required by MRTG graphs, respectively.

V. EVALUATION

The FlexACMS evaluation discussed in this section is twofold. First, we discuss the functional issues of employing FlexACMS in an IaaS environment with real cloud platforms and monitoring solutions. The goal is to identify the overhead introduced by FlexACMS when automatically configuring the monitoring slices required for cloud slices of a new IaaS service request from a cloud user. Second, we evaluate the scalability issues of the FlexACMS in order to determine the feasibility of our solution in large cloud environments. The numbers depicted in the graphics are the mean values observed in the experiments. All the experiments were repeated 30 times with confidence intervals of 95%. The details of these evaluations are described in the following subsections.



(a) Previous FlexACMS version (2 metrics) (b) Enhanced FlexACMS version (2 metrics) (c) Enhanced FlexACMS version (52 metrics)

Fig. 3. Comparative: serial vs. parallel approach

A. *Functional Evaluation*

We use the functional evaluation to analyze two issues. First we identify the overhead introduced by FlexACMS in automatically configure monitoring slices for cloud slices. The goal is to quantify the increase on the response time by adding the automatic configuration of the monitoring support for a cloud slice. Second, we compare the gains that enhanced architecture depicted on Figure 1 has over the previous architecture [3].

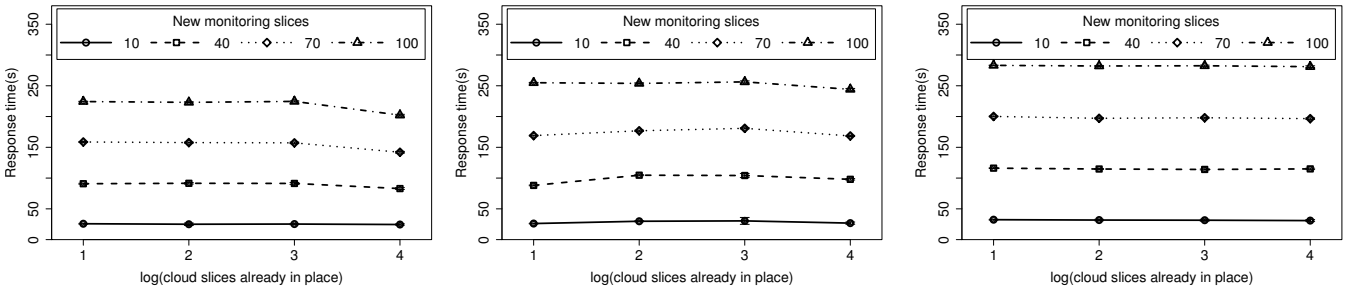
The evaluation scenario is composed of two servers: an OpenStack, and a monitoring server. The former has two Intel(R) Xeon(R) CPU E5430@2.66GHz with 4 cores per processor, 16GB of RAM, running Ubuntu Server 12.04 LTS. The latter has one Intel(R) Core(TM) i5 CPU 650@3.20GHz, 4GB of RAM, running Ubuntu Server 12.10. Both are connected to a switch by 1 Gbps links. The OpenStack server hosts all components of OpenStack infrastructure and an OpenStack gatherer. The monitoring server hosts the FlexACMS Core, Nagios and MRTG solutions, their corresponding configurators, and runs 10 configurator workers simultaneously.

We observe the time required to create a cloud slice using the OpenStack platform (OpenStack time) and the corresponding time to inform FlexACMS and to configure the monitoring slices associated to new cloud slices (FlexACMS time). The monitoring slices in the experiment reflects two metrics: host status and CPU usage. Thus, two configurators were defined: one for the host monitoring configuration in Nagios, and the second to create configuration to monitor CPU usage.

The Figure 3a shows the evaluation results of response time of the previous architecture. OpenStack requires around to 73-63% of the experiment time to create cloud slices. The previous version of FlexACMS required around to 37-27% of experiment time to configure the corresponding monitoring slices. The Figure 3b shows the response time results of the enhanced architecture. In this experiment, OpenStack requires around to 82-71% of the experiment time to create cloud slices, while the enhanced FlexACMS requires around to 29-18% of the experiment time to configure the corresponding monitoring slices. The enhanced FlexACMS reduces in circa of 10% the impact of the automatic monitoring slice configuration in the whole experiment time: OpenStack plus FlexACMS times.

In Figure 3b there is a clear trend showing that the increase in the number of cloud slices does not create a significant increase in the response time of the enhanced FlexACMS, when configuring few metrics per monitoring slice. For instance, when we analyze the bars for the creation of 10 monitoring slices, we observed that OpenStack requires around 81.8% of the experiment time, and FlexACMS requires 18.2% of the experiment time. In addition, when we compare the experiment time for creation of 9 monitoring slices (*i.e.*, cloud slices), we observe again that the response time of enhanced FlexACMS corresponds to 20.5%, almost the same value if compared with the case of 10 monitoring slices. This happens because the enhanced FlexACMS is built to better distribute the configuration load by exploring the parallelism and thus reducing the overall time of configuration of multiple monitoring slices. These experiments show that enhanced FlexACMS architecture fits well when a highly number of monitoring slices need to be created with few number of monitoring metrics per slice.

However, the monitoring slices were composed of only two metrics: host status and CPU usage. We then observed FlexACMS configuring more complex monitoring slices, which are composed of: a host status and 50 metrics monitored by Nagios, and a network usage graphic built by MRTG. In this case, each monitoring slice has 52 metrics and requires 52 configurator calls to be configured. The Figure 3c shows that FlexACMS requires more time to configure more complex monitoring slices. In this scenario, OpenStack requires around to 60-55% of the experiment time to create cloud slices against 45-40% required by FlexACMS to configure the corresponding monitoring slices. Indeed, looking to the bars related with the 10 created cloud slices in Figures 3b and 3c, the FlexACMS time to configure more complex monitoring slices was 2.66 times (50.4s in Figure 3c) higher than the response time to configure simpler monitoring slices (20.1s in Figure 3b). Thus, we can conclude that the number of metrics of a monitoring slice affects the FlexACMS time. Although, the response time and the monitoring slice size did not increase in the same rate (respectively, 2.66 times vs. 26 times). The scalability evaluation on Section V-B will further analyze a growing number of metrics per monitoring slice.



(a) 5 metrics per monitoring slice (b) 25 metrics per monitoring slice (c) 50 metrics per monitoring slice

Fig. 4. Scalability results

B. Scalability Evaluation

In this scalability evaluation, we are interested in further analyzing how the response time is affected when a higher number of metrics is used in a single monitoring slice. The evaluation scenario was composed of two servers: a monitoring server, and a FlexACMS server. Both servers have one Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz with 4GB of RAM, running Gentoo Linux and Ubuntu Server 12.10, respectively. Both servers are connected to a switch with links of 1 Gbps. The monitoring server runs 10 configurator workers simultaneously, and hosts Nagios configurators. The FlexACMS server hosts the FlexACMS Core, the gatherer that generates artificial input, and runs 10 workers simultaneously that are able to consume Requests queue and Changes queue. The implementation details, except the gatherer, are described in Section IV-B. The gatherer was developed using Perl to generate information about cloud slices with 5 resources and 20 associated information. These values are similar to the values achieved when we used the real OpenStack gatherer.

We varied the number of metrics in each monitoring slice in 5, 25, and 50 metrics per monitoring slice, which represents 1, 5, and 10 metrics per resource, respectively. We believe that 50 metrics per monitoring slice (10 metrics per resource) is a high number of metrics. Another scalability issue that we are interested is the capacity of the enhanced FlexACMS to hold the response time when a large number of cloud slices are already in place. We varied the number of cloud slices in 10, 100, 1000, 10000 cloud slices already in place. As far as we know there are not statistics of real large IaaS scenarios. We believe that an InP that hosts 10000 cloud slices is large. We also varied the number of monitoring slices to be built during a burst in 10, 40, 70, and 100 new monitoring slices per burst. We believe that the creation of 100 new cloud slices in a single gatherer request is a high number of new cloud slices. Specially, if we consider that gatherer requests to FlexACMS will be into a polling process of few minutes.

The first observation when analyzing the response time evaluation results shown in Figures 4a, 4b, and 4c is related to the number of cloud slices that are already in place. We can observe this following the lines along the x-axis in each figure. The x-axis uses a logarithmic scale to allow a proper comparison. We vary the number of cloud slices from 10 to 10000 cloud slices, *i.e.*, from 10^1 to 10^4, and we can observe that lines are flat, and are almost constant along the x-axis. This means that FlexACMS scales when a growing number of cloud slices are in place. The second observation is related to the number of monitoring slices that need to be built in a burst. We can observe this comparing the distance between adjacent lines into the same figure. We varied the number in 10, 40, 70, and 100 required monitoring slices. Thus, we varied by a fixed number (30 monitoring slices), and thus the response time also must vary by a fixed value between lines in the same figure. We can observe that the lines keep a similar distance when increase the number of monitoring slices.

However, it is difficult to measure that distances are similar just looking to the figures. Therefore, we calculated the throughput for each scenario to assure that FlexACMS scales when the number of monitoring slices increases. To calculate the throughput we need the number of configured metrics for each line, which can be calculated multiplying the number of new slices by the number of metrics per slice. For instance, 100 new slices $*$ 50 metrics per slice $=$ 5000 metrics to be configured. We just need to divide the response time by the number of metrics configured to achieve the throughput. The lines on each figure has similar throughput that are around to 2.5, 10, and 17 metrics configured per second in Figures 4a, 4b, and 4c, respectively. We observed that in the same figure the observations have similar throughput, thus, we can conclude that FlexACMS scales when the number of monitoring slices that need to be built increases.

Finally, we observe the effect of growing number of metrics in each monitoring slice by comparing the lines of same pattern between neighbor figures. We varied the number of metrics per slice in 5 metrics, 25 metrics, and 50 metrics. We can observe, for instance, comparing the lines of Figures 4a and 4b which present results for 5 metrics and 25 metrics, respectively, *i.e.* monitoring slices 5 times greater. The response time in Figure 4b is slight higher than in Figure 4a, and the difference is much smaller than 5 times. Similar conclusions can be done analyzing the difference between

Figures 4a and 4c, which present results for 5 and 50 metrics, respectively, *i.e.* monitoring slices are 10 times greater, and between Figures 4b and 4c, which present results for 25 and 50 metrics, respectively, *i.e.*, where monitoring slices are 2 times greater. We observed, thus, that FlexACMS scales when a growing number of metrics are used on monitoring slices.

The enhanced FlexACMS architecture can consume more computational resources that our previous single-threaded architecture. However, the resource consumption can be tuned by the administrator adjusting the number of workers. Our experiments were performed in commodity computers with 2 cores and 2 threads per core; however, we conclude that achieved response times are appropriate. Beyond, despite our evaluated scenarios were based on single machine as FlexACMS server, its architecture uses components employed in traditional environments, such as web server, database, and queue system. Thus, there are solutions to leverage the throughput of these components, such as load balancers. Cloud administrators can built more complex monitoring environments employing load balancers to increase FlexACMS throughput if necessary.

VI. Conclusion

In this paper, we enhanced FlexACMS which configures monitoring slices automatically when cloud slices are created. The enhancement introduced two features that leverages the automation achieved by FlexACMS. The first feature is the dynamic and automatic attribution of configuration tasks to pools of monitoring servers. The second feature is the load balancing when attributing configuration tasks among the monitoring servers members of a pool. This feature is achieved because monitoring servers volunteer to receive configuration tasks and can decide to stop receiving tasks while they are overloaded. Both features required architectural modifications into the FlexACMS Core, such as the use of queue/workers. This new approach allowed FlexACMS to break the whole task of configuring monitoring slices into small tasks performed in parallel improving its response time.

We conducted a comparative evaluation between previous and enhanced FlexACMS. We measured the experiment time as the time to create cloud slices in OpenStack and the additional time required to configure the corresponding monitoring slices in FlexACMS. The previous version configured monitoring slices in 34,7% of the experiment time. The enhanced version configured the monitoring slices in 24,1% of the experiment time. Thus, the enhanced version reduces in circa of 10% its influence in the experiment time. However, when looking to FlexACMS time to configure 10 monitoring slices, the enhanced version reduces the response time from 49,76s to 20.1s, which represents up to 60% of response time reduction. In this case, FlexACMS explores the parallelism available when needs to configure 10 monitoring slices, which demonstrate the benefit of queue/workers approach.

We also conducted a scalability evaluation, which focuses on how the FlexACMS response time is affected by different aspects. We varied the number of cloud slices already in place, the number of monitoring slices to be built, and the number of metrics in each monitoring slice. We concluded that the response time was not affected by the number of cloud slices already in place. We concluded that the response time is affected by the number of metrics in each monitoring slice and by the number of monitoring slices that must be built, but the response time does not increase in the same rate that the number of metrics and monitoring slices increases. These observations show that the enhanced FlexACMS scales in regards to response time, which demonstrate the feasibility of our approach in large cloud computing environments.

After solve the problem of create monitoring slices automatically, we are able to address other related cloud monitoring issues, as the reconfiguration and destruction of monitoring slices, and adaptive allocation strategies in placing new monitoring slices. We also plan to employ FlexACMS into an InP provider to improve its capabilities and offer it as an open source project. Furthermore, we want to evaluate the feasibility of the framework ideas in Platform as a Service (PaaS) and Software as a Service (SaaS) cloud models.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008.

[2] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," *Computers, IEEE Transactions on*, vol. 62, no. 9, pp. 1705–1720, 2013.

[3] M. Carvalho, R. Esteves, G. Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A Cloud Monitoring Framework for Self-Configured Monitoring Slices Based on Multiple Tools," in *9th Intl. Conference on Network and Service Management 2013 (CNSM 2013) - Poster Session*, Oct 2013.

[4] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: A complete approach to cloud monitoring ," *Future Generation Computer Systems*, pp. –, 2013.

[5] OpenStack Community, "OpenStack," 2014, available at: http://www.openstack.org/ accessed in: Feb. 2014.

[6] Nagios Enterprises, "Nagios," 2014, available at: http://www.nagios.org/ accessed in: Feb. 2014.

[7] Tobias Oetiker, "MRTG," 2014, available at: http://oss.oetiker.ch/mrtg/ accessed in: Feb. 2014.

[8] S. De Chaves, R. Uriarte, and C. Westphall, "Toward an Architecture for Monitoring Private Clouds," *Communications Magazine, IEEE*, vol. 49, no. 12, pp. 130 –137, Dec. 2011.

[9] Amazon, "Amazon CloudWatch," 2014, available at: http://aws.amazon.com/en/cloudwatch/ accessed in: Feb. 2014.

[10] OpenStack Community, "Ceilometer," 2014, available at: https://wiki.openstack.org/wiki/Ceilometer accessed in: Feb. 2014.

[11] Amazon, "Amazon Web Services," 2014, available at: http://aws.amazon.com/ accessed in: Feb. 2014.

[12] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *IEEE 3rd International Conference on Cloud Computing*, Jul. 2010, pp. 313 –320.

[13] M. Rak, S. Venticinque, T. Mahr, G. Echevarria, and G. Esnal, "Cloud Application Monitoring: The mOSAIC Approach," in *IEEE CloudCom 2011*, Dec. 2011, pp. 758 –763.

[14] H. Han, S. Kim, H. Jung, H. Yeom, C. Yoon, J. Park, and Y. Lee, "A RESTful Approach to the Management of Cloud Infrastructure," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, Sep. 2009, pp. 139 –142.

[15] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093 – 2115, 2013.

[16] Puppet Labs, "Puppet," 2014, available at: http://puppetlabs.com/ accessed in: Feb. 2014.

[17] Opscode, "Chef," 2014, available at: http://www.opscode.com/chef/ accessed in: Feb. 2014.

[18] OpenStack Community, "OpenStack API," 2014, available at: http://api.openstack.org/ accessed in: Feb. 2014.