Role-based Templates for Cloud Monitoring

The An Binh Nguyen, Melanie Siebenhaar, Ronny Hans, Ralf Steinmetz
Multimedia Communications Lab (KOM)
Technische Universität Darmstadt
Rundeturmstr. 10, 64283 Darmstadt, Hessen, Germany
{The.An.Binh.Nguyen, Melanie.Siebenhaar, Ronny.Hans, Ralf.Steinmetz}@kom.tu-darmstadt.de

***Abstract*—Cloud computing has evolved in the recent years to a well established computing paradigm. With this evolution, the complexity and requirements for monitoring cloud-based services have also increased. Without a doubt, monitoring for cloud computing is a crucial task which has been addressed in a number of research works. However, monitoring for cloud computing is often designed to be carried out by cloud providers. Monitoring by cloud providers on the one hand offers the flexibility and full control required for monitoring; on the other hand, the trustworthiness of the cloud provider is often questioned. In this work, we present a generic approach that can harmonize both of the aforementioned issues. Our solution abstracts from the complexity by using role-based templates for monitoring in combination with autonomous agents; thus, this approach can be used by both, cloud consumers and cloud providers. With a proof of concept prototype, we show that our approach can be adapted for large scale cloud monitoring scenarios. Furthermore, we discuss the possibility that our monitoring solution can be extended to be applicable for different domains.**

I. Introduction

Cloud computing is one of the most widely used computing models nowadays. It provides many advantages not only for cloud consumers, but also for cloud providers [1]. On the one hand, cloud consumers enjoy the pay-as-you-go and on-demand services [2]; on the other hand, cloud providers can maintain their services as well as leverage their infrastructure more efficiently. As a consequence, cloud monitoring is important for both, cloud providers and consumers [3]. While cloud providers require monitoring to assess the performance of their provided services and to maintain their resources, cloud consumers are concerned if the performance of the cloud services adheres to the agreement negotiated with cloud providers. Based on the taxonomy of cloud computing, there are three main service categories, i.e., Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) [4]. Services from one category can be further combined with services from other categories to create a new cloud service, i.e. cloud consumers of one cloud service can become a cloud provider for other types of service [1]. Hence, the roles of the stakeholders in cloud computing are flexible and dynamic. Thus, the complexity and flexibility in the service model of cloud computing makes monitoring in cloud computing more difficult compared to other computing models. Moreover, cloud services can also be deployed across different cloud providers which makes aggregation of monitoring data a demanding task. As a consequence, holistic monitoring is one of the most challenging tasks for cloud computing [3].

Since the roles of cloud stakeholders can be dynamic, i.e, cloud consumers can be at the same time cloud providers for others, we need to take the roles of the actors into consideration for monitoring. Even for the same users, their different roles can lead to different monitoring requirements. Besides, cloud consumers and cloud providers have a different expertise and different capabilities for monitoring tasks. Cloud consumers as the real end users do not possess the required skills for monitoring as cloud service developers/providers. Furthermore, real end users do not have the same monitoring concerns as a developer or cloud provider. For instance, an end user only requires to be notified if the cloud service is available or not, while a service developer wants to know more about resource related information of the infrastructure. All in all, the different roles of the stakeholders in cloud computing can affect monitoring tasks and need to be tackled carefully [5].

Very often, monitoring tasks are carried out by cloud providers because cloud providers have all the capabilities and expertise required for monitoring. But the validity of the monitoring data provided by cloud providers can be questioned, since cloud providers might want to hide information in order to avoid paying penalties for cloud consumers. On the contrary, cloud consumers that want to monitor the service to verify the promised performance often lack the capability and resources (e.g, computing capacity, human capacity) needed for monitoring. A solution for the mentioned issues is a trusted third party who can take care of monitoring activities and act as a broker for both cloud providers and cloud consumers [6].

Cloud computing is the evolution of distributed computing, therefore, many legacy approaches for monitoring distributed systems can also be applied for cloud computing. In addition to the characteristics inherited from distributed computing, cloud computing can also be scaled up and down dynamically, which makes adaptivity an important aspect for cloud monitoring. Autonomous agents are often used to solve many problems in distributed systems. Even though agents are an old technology, they can still fulfill many requirements for monitoring in cloud computing [7], [8]. An agent is referred to as an entity (in general software entity) that can act autonomously within a system given appropriate intelligence. Agents can be moved from one container to another container, this feature is called the mobility feature. While the autonomous behavior of agents can support adaptivity and automatic task execution, the mobility feature supports dynamic and flexible deployment.

In this paper, we introduce a user-oriented approach to allow holistic monitoring for cloud computing:

- We design a role-based template to extend model-based monitoring.
- We make use of agent mobility [9] in combination

with our proposed role-based templates to enhance flexibility and adaptability of monitoring activities.

- We propose to use third-party monitoring as a service as the main facilitator for the previous two propositions. With a trusted third-party, we can also realize independent monitoring.

The rest of this paper is structured as follows: In Section II, we review and discuss the related work. Section III discusses our proposed role-based templates for cloud monitoring and their specification. Section IV discusses the incorporation of mobile agents in our approach to allow for holistic monitoring. In Section V, we describe the results of the conducted experiments with our proof-of-concept prototype. Finally, the conclusion and outlook for future work will be provided in Section VI.

II. Related Work

Monitoring distributed systems has been one of the most challenging problems since decades. Accordingly, there have been several prominent monitoring solutions for distributed systems which are still in use in practice today, e.g., Nagios [10], Ganglia [11]. However, these systems are designed to target distributed servers; as a result they focus on monitoring infrastructure and lack support for services of higher layers in cloud computing, i.e., SaaS. In addition to the existing monitoring solutions for distributed systems, there have been many research works that address different issues of cloud monitoring.

The services model of cloud computing requires the bridging between low-level and high-level monitoring metrics. In general, there are different performance metrics, some of them can be measured directly, some of them have to be derived from other metrics. LoM2HiS [12] is such an approach that attempts to map the monitoring metrics from the infrastructure level to a higher level. The monitoring is carried out by monitoring agents on the corresponding cloud infrastructure. Only the hardware and network resource metrics are collected and sent to a remote host monitor. The low-level monitoring data is forwarded through a message queue to a complex event processing (CEP) engine for further processing as well as mapping to higher layer monitoring metrics.

CASVID [13] deals specifically with SLA management and application monitoring for cloud computing. The framework uses a front end to allow users to deploy pre-configured cloud instances. Collection monitoring data is conducted using the SNMP protocol. The management node polls the monitoring data periodically from the SNMP agent running on each VM instance. To determine the most effective polling interval, the authors design an algorithm which looks for the most optimal interval from a list of possible intervals by comparing the net utility obtained from these.

Hoßbach et al. [14] present a monitoring architecture which is based on CEP and a message broker for data delivery. Monitoring data collected from different cloud layers by capable sensors is sent as stream. Streaming data can be aggregated and filtered easily by a CEP engine. A message broker is in charge of routing data streams from one component of the monitoring system to the other and is the central entity of the proposed architecture. By connecting all modules of the monitoring system and the cloud system to the message broker, the cloud system and the monitoring system are able to communicate with each other and make use of data streams produced by other modules. Thus, a high level of aggregation can be achieved.

MISURE [15] exploits different existing monitoring solutions and adapts them for cloud monitoring. MISURE is based on the publish/subscribe model to disseminate monitoring data. Monitoring data collected from different sources is published as stream. Data streams can be aggregated or split in different ways which will result in appropriate streams to be delivered to subscribers. The subscription can be done in three ways, i.e., a subscription through notification, a subscription through storage engine and a PaaS API intended for developers. The ability to exploit and plug existing monitoring solution into the system makes the MISURE approach extensible.

DARGOS [16] also follows the publish/subscribe model and focuses on reducing the amount of transmitted data generated by monitoring. DARGOS bases its data delivering system on DDS - Data Distribution Service. In DARGOS, the cloud hosts need to be deployed hierarchically and grouped together into zones. Each monitoring agent is in charge of receiving data from one zone, but a hypervisor agent can receive data from several zones, which supports multi-tenancy for cloud computing. The system also allows tenants to configure the monitoring view adaptively with regard to the requirements by subscribing to different topics.

Leitner et al. [17] use Aspect Oriented Programming to move execution code to a remote cloud host. Moving code remotely allows the system to adapt the execution of tasks dynamically. Moreover, the authors also propose a generic architecture for event based monitoring. Each component of the monitoring system can emit events which will go through a CEP engine and be correlated to a higher-level monitoring event. The designed architecture comprises three correlation levels, i.e., host level, resource level and metric level. Grouping events hierarchically is an important step in order to define metrics for holistic monitoring in cloud computing.

König et al. [18] introduce another cloud monitoring architecture which collects data from all layers of cloud computing. They divide their monitoring system into three layers. The data layer makes use of different adapters to collect monitoring data from different sources; the processing layer uses a meta language to define queries and policies which are executed by the operators to aggregate monitoring data; the distribution layer is in charge of deploying the monitoring components on the cloud infrastructure according to the definition from the monitoring policies.

Although SAaaS [19] with the main focus on security is designed for abnormal incident detection, its architecture is still suitable to monitor performance in cloud computing. The system makes use of autonomous agents to detect violations or possible threats. The threats of business flows can be represented using a special security modeling language. This formal representation allows the agents to react automatically on detected security breaches.

Shao et al. [5] present a model-based monitoring framework for cloud computing. The authors realize the importance of abstracting from the concerns such as types of servers or types of databases to reduce complexity for monitoring. As a result, an entity model is constructed which represents a running cloud. The entity model focuses on functional aspects of a cloud instance. Based on the profile model, cloud monitoring

can be orchestrated accordingly during run-time.
Comparable to our proposed approach are the works of [5] and [19]. In contrast to [5], [19], our user-oriented template model is able to represent monitoring requirements that are specific to a particular role. Both functional and non-functional requirements are supported. With role-based templates, we are able to reduce the complexity for personalized monitoring. Even though the monitoring roles abstract from individual users with similar interests in monitoring, they can still be adjusted to individual needs. The use of monitoring roles also allows the users to better control the collection of monitoring data, thus supports privacy protection. Last but not least, our architecture is designed to support adaptivity for monitoring during run-time. In the following sections, we will gradually present the design of the components that constitute our monitoring system.

III. Role-based templates for monitoring

A monitoring task can vary a lot based on the context and the demands of a user. With regard to cloud computing, it gets even more complicated. A cloud consumer of one service can be a cloud provider of another service. Even when using the same service, different roles will lead to different concerns, e.g., an end user using a cloud storage instance is concerned whether the storage instance is available, while a service developer might want to get more information from the instance such as network overload from data access. Given the fact, that a cloud actor can take on different roles at the same time, the importance of distinguishing roles for cloud monitoring is emphasized.

A general service establishment process involves the leasing of infrastructure resources from a provider, the creation of the service on the basis of the leased resources and the usage of the service. Based on the taxonomy of cloud computing with three main service models (IaaS, PaaS, SaaS) [4] and the service establishment process, we identified three main roles for cloud monitoring, i.e., end user, developer and infrastructure operator. A similar role classification scheme can also be found in [5]. An end user is the service consumer who usually has no expertise about maintaining the service. An end user simply wants to use the service to fulfill her goal. A developer is referred to as the creator of a service. A developer is usually an expert with the deep knowledge about the cloud services stack and the underlying resources required to run the services. An infrastructure operator is the owner of the infrastructure, whose main concern is to keep the infrastructure available and compliant to the agreements and demands negotiated with the cloud consumers. Further role classification schemes are possible if we consider cloud computing from other perspectives. One other role classification is to further distinguish between private users and enterprise users. While a private user often uses public cloud services, an enterprise user can set up and build hybrid cloud infrastructures. Hence, the monitoring concerns and responsibilities for these two roles are different. Depending on the scenarios, we can create monitoring roles that capture common requirements for particular contexts. Hence, a roles database can be conceivable. Another advantage of using monitoring roles can be noticed in monitoring data transmission. Given different concerns, different roles will require different data. An end user without expertise about the infrastructure is not interested in low-level monitoring parameters, rather he is more concerned to get notified in case his demands for the service are not met. In this case, it is sufficient for the user to receive only a short notification, without sending the whole monitoring information required for service developers. In another example, a service developer needs to get detailed information about the underlying infrastructure so that he can react to failures. Thus, the amount of data that is transmitted in these two examples is extremely different. Therefore, it is beneficial to determine, which data to send and how much data to send depending on different contexts. By doing so, we will be able to spare a considerable amount of data given millions of users and cloud instances. The separation of roles helps us to achieve this goal. By assigning suitable roles for each cloud service, users are able to determine which data and how much data to be sent individually for various scenarios. Hence, the amount of monitoring data to be transmitted can be reduced intelligently. A template depending on the respective role predefines the maximum amount of monitoring data, and can be further adjusted to suit the need of each single user.

Having the classification into roles that capture the specific requirements for different services, the complexity of cloud monitoring can be reduced considerably. Moreover, it is possible to further enhance the monitoring process with a kick-off template. A monitoring template is an abstract description which helps to model the monitoring requirements. A monitoring template represented in a modeling language with a standard syntax can be processed automatically. For different roles, we can predefine different templates which serve specifically for their needs. Users are allowed to configure the template based on their demands. Multiple templates can be nested together which allows to cope with more complex use-cases. All in all, role-based monitoring templates can be reused easily across different domains and cloud providers.

Inspired by the SLA template model from SLA@SOI [20], we designed an abstract model for monitoring templates. The abstract monitoring templates capture the most important aspects which are required for monitoring. The abstraction can be represented using different modeling languages which allows for exchange and task automation. The abstract model for the monitoring templates is shown in Figure 1. To support model re-usability, the monitoring templates which have been created before are stored in a database together with multiple meta-information. The meta-information altogether characterize the context of the respective monitoring scenario. The users depending on their corresponding roles can query the database to look for templates with similar contexts based on the meta-information. If similar templates can be found, these templates will be returned back and can be used directly by the user or can be adapted respectively. If no result can be returned, a default template will be recommended to the user according to his role.

A template comprises three main components, i.e., cloud endpoint, monitoring variable and monitoring policy (cf. Figure 1). Cloud endpoint contains monitoring layers of the cloud services (IaaS, PaaS or SaaS for further recommendations regarding monitoring parameters) and deployment details of the monitoring units. Since cloud instances can be located at many places, remote deployment of monitoring units is an important aspect. Hence, under deployment details, the user needs to provide information how to connect to the

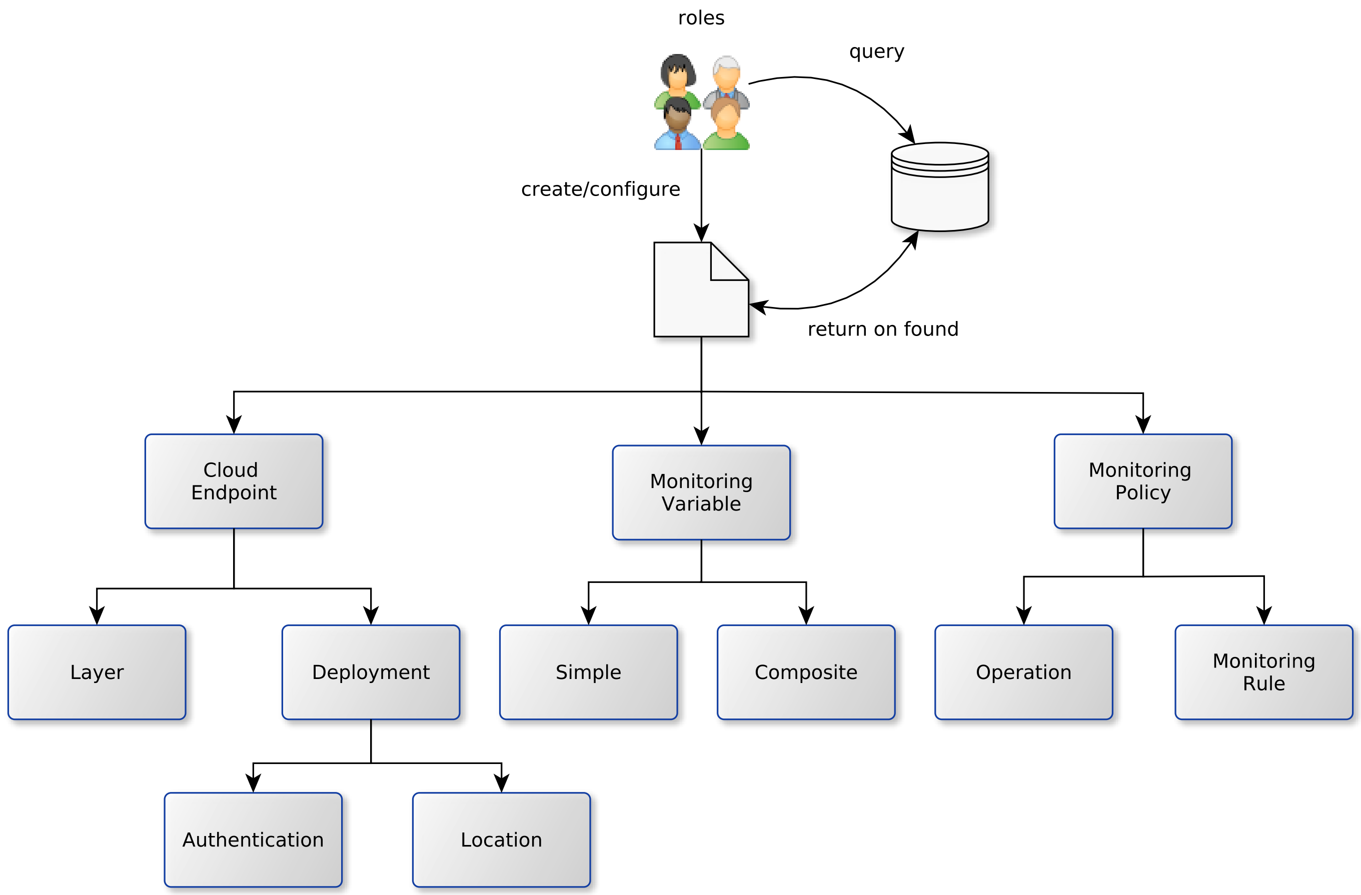

Fig. 1. Abstraction model of role-based templates for cloud monitoring

cloud services. This information is captured by the parameters defined in the location component of the template. Location parameters can be, for example, a URL address or a SSH address. To make sure that the monitoring units get official access rights to the cloud service, the access information of the user such as password or private key have to be specified under authentication parameter. Furthermore, in authentication detail, the user can also specify to which extend the monitoring entity is allowed to access his cloud instance with regard to the granted role. Monitoring variables are a crucial part of the monitoring template. The user has to specify what he wants to measure. We classify the monitoring variables into two main types, i.e., simple variables and composite variables. Simple variables are the monitoring parameters which can be measured directly. Low-level resource metrics and operating system statistics are examples of this type of variable. The composite metrics represent a combination of multiple simple metrics. An example for a composite metric is availability which can be calculated by uptime and downtime. The composite metrics tend to be high-level metrics which are often of concern for the end user. The monitoring variables dictate which metrics to measure. To complete the monitoring template, we still need to know how to measure the monitoring data and what to do with the obtained data. This information is defined under monitoring policies. Accordingly, there are two sub-components under monitoring policy, i.e., monitoring rules and operation. The monitoring rules are the expressions that indicate how to measure the monitoring variables and what are the conditions that the rules need to be aware of. For instance, the CPU utilization of an instance needs to be measured every 30 seconds and checked if it exceeds 99% (this might indicate an overload that requires proper reaction). Last but not least, it is important to define which operations to execute after collecting monitoring data. A simple but very common operation is to send the monitoring report to a dedicated destination. Further automatic reaction is possible, such as automatic scale up or scale down of the infrastructure. The operation definition in the template can also be used as an interface to integrate the monitoring system into other management processes such as billing or analytic systems.

IV. Agent-Based Monitoring Architecture

Agents are an old but powerful technology which is still used a lot in scientific researches. Agents can be understood as a software entity which can act autonomously based on the intelligence it is given. Therefore, agents are generally useful for distributed systems and particularly for monitoring. We combine our proposed model for role-based templates with agents for cloud monitoring. We define and use three types of agents in our monitoring architecture: activator agent, sensor agent and collector agent. The functions of these agents will be described together with the monitoring workflow. We divide

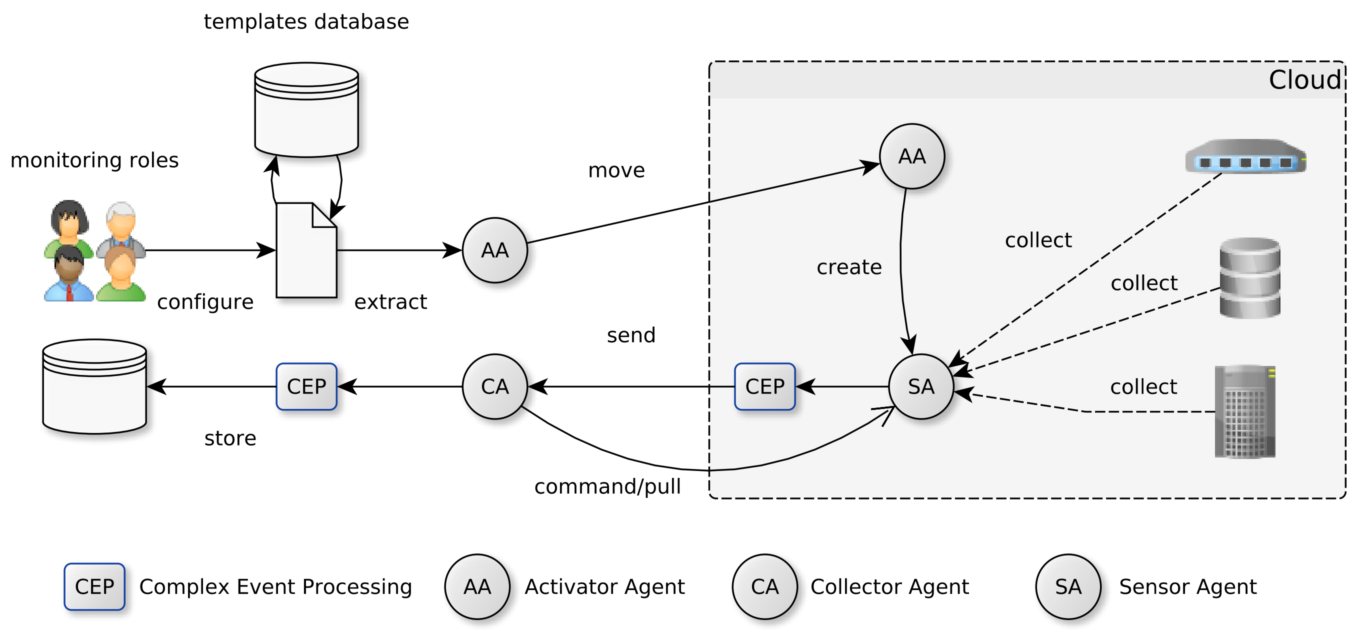

Fig. 2. Monitoring architecture by incorporating monitoring roles, monitoring templates and various types of agents

the monitoring workflow into two main steps and describe these steps respectively in the following subsections: monitor deployment and monitor data collection. The overall design of our monitoring approach can be found in Figure 2.

A. Monitor deployment

Our monitoring architecture is designed following a trusted third party approach. In our architecture, a trusted third party will take care of the monitoring for a cloud actor regardless if he is an end user, a developer or cloud operator. Nevertheless, the same workflow can also be executed by the cloud providers themselves. In order to deploy the monitoring solution for the respective cloud services, a user configures a monitoring template as described in Section III. The configuration and generation of a monitoring template are done through a gateway provided by a trusted third party. With the chosen roles and the configured monitoring template, the trusted third party now obtains enough information to deploy the actual monitoring service. An activator agent will be created at this point for the deployment task. The activator agent can extract the cloud endpoint from the template. With the endpoint information, the activator agent moves itself to the cloud instances to be monitored. In the cloud endpoint definition, the activator agent can also extract the appropriate authentication data in order to obtain the required access rights to execute the actual deployment on the cloud instance. The access rights also allow the activator agent to check for the dependencies and requirements to activate the monitoring activities on the cloud. After validating that all the desired conditions are met, the activator agent will create the corresponding sensor agent and remove itself from the monitored cloud instances to spare resources. The use of a separate activator agent as mobile agent to carry the parameters required for monitoring and the creation of the sensor agent for the actual monitoring task directly on the cloud will further reduce the network traffic. This is due to the fact that in most cases, the activator agent only has to contain simple behavior (less code/data) for the activation, while the sensor agent often requires more complex behaviors (more code/data). Since a cloud can contain thousands of instances, moving complete sensor agents to all these instances will obviously generate more traffic compared to moving only activator agents. In addition, sensor agents may fail during their monitoring activities, e.g., due to an outage of the underlying infrastructure. Hence, appropriate approaches for robust monitor placement are required that are explored in our ongoing work in [21], [22].

B. Monitor data collection

After having been created on the cloud systems, the sensor agent will carry out the real monitoring tasks based on the monitoring policies defined in the monitoring template. The activator agent from the previous step already checked the dependencies required for the monitoring activities; thus, upon creation, the sensor agent is capable of collecting monitoring data according to the template and executing appropriate actions. The data collected by sensor agents can be low-level monitoring data such as CPU and memory usage or high-level monitoring data obtained by aggregating the low-level data. For example, availability is calculated using mean time to failure and mean time to repair. Both types of data can be collected or derived by the sensor agent. To facilitate the collection of low level-monitoring data, the sensor agent can be equipped and extended with further tools and libraries which enable to access the required information. The definition of monitoring variables in the template allows the sensor agent to calculate high-level monitoring metrics by combining the collected low-level metrics.

Monitoring data collected from the cloud instances can be evaluated directly on the cloud or be disseminated to other remote components in the monitoring system for further processing. These are specified in the operation entry located under the monitoring policy entry of the template. In case that the monitoring data needs to be disseminated remotely, the operation will contain the destinations of the receivers. The corresponding receivers in our monitoring architecture are also realized through agents, i.e., collector agents. Collector agents are a special type of agents which are able to communicate directly with sensor agents. The communication between two agents is bidirectional. The collector agents can be started based on the preferences of the user defined in the monitoring template. Hence, collector agents can be run on the infrastructure of the trusted third party or on a dedicated server of the user. A service developer often owns other servers besides cloud instances. In addition to the task of receiving monitoring data from a sensor agent, the ability to communicate bidirectionally with a sensor agent also allows the user to send commands to the sensor agent through the collector agent. In consequence, with appropriate intelligence, the sensor agent can adapt itself upon receiving commands. Therefore, such set-up will allow our monitoring system to be adaptive during the monitoring process.

We also employ complex event processing (CEP) in our monitoring architecture since a CEP engine can filter and aggregate monitoring data efficiently. We deploy CEP engines at two different levels. A CEP engine is started together with one sensor agent in order for a sensor agent to filter, correlate and evaluate the monitoring data directly on the cloud. Similarly, a CEP engine is started together with a collector agent. The collector agent is intended to receive monitoring data from multiple sensor agents. Hence, with the CEP engine in conjunction with a collector agent, we can correlate information from multiple cloud instances that make up the underlying infrastructure for the cloud services.

The position of CEP at two levels has two main advantages. At the cloud instances, this allows the sensor agent to filter monitoring data which reduces the network traffic caused by monitoring data transmission. The sensor agent with the CEP engine is also able to correlate raw low-level data to high-level user oriented monitoring metrics. For the collector agent, the CEP engine allows to correlate data from multiple cloud instances belonging to the user. Thus, holistic monitoring across providers can be achieved.

V. Evaluation

Based on the previous design of our monitoring approach, we implemented a prototype as a proof of concept for evaluation. The architecture of our prototype is shown in Figure 2. In the evaluation, we analyzed the overhead caused by the monitoring system. We used the agent development framework JADE[1] to implement the different types of agents of our concept. An agent can be extended with further libraries for more functionalities. Furthermore, we used the JADE Inter-Platform Mobility (jipms)[2] to move agents from one container to another remote container.

The server as a trusted third party which allows a user to configure monitoring templates is realized using Apache

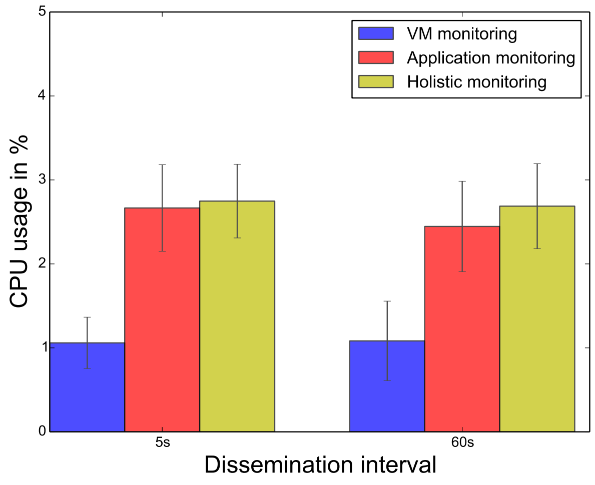

Fig. 3. CPU overhead measured by deploying a sensor agent on a client

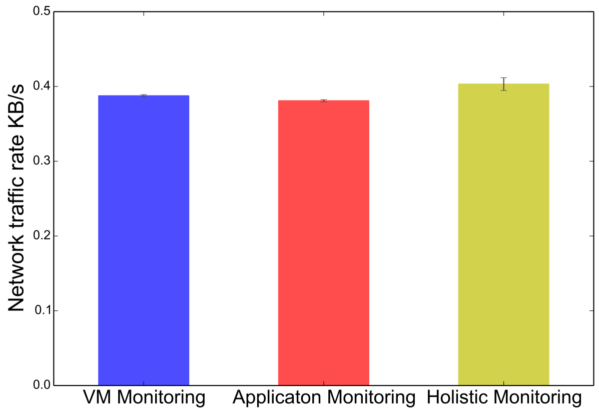

Fig. 4. Data transmission rate measured by a monitored instance in three sample scenarios

Tomcat Server with a communication gateway to the JADE platform. A user configures a monitoring template by choosing his role. Depending on the role, different forms and different parameters will be presented to the user. These can then be configured by users. Depending on the scenario and demands of a user, the collector agents can be created in advance if the monitoring is carried out by the trusted third party; or the user can specify a destination of his own to run collector agents as well. By specifying a destination for collector agents, the user can control the final aggregation and storage of monitoring data. The information extracted by the submitted form will be transferred through a gateway to a JADE main container. The activator agent is created at this point with the monitoring parameters. Next, the activator agent will move itself to the designated cloud instance. The sensor agent will be created there as a result with the required dependencies to collect monitoring data. We use Sigar[3] to collect monitoring data from the cloud instances. In our design, we also emphasized the necessity of a CEP engine at two different collection levels.

[1]http://jade.tilab.com/

[2]http://sourceforge.net/projects/jipms/

[3]http://www.hyperic.com/products/sigar

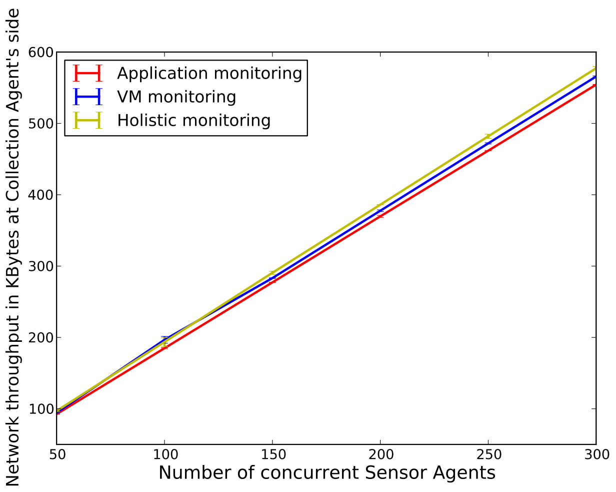

Fig. 5. Data throughput measured at the monitoring server in three sample scenarios

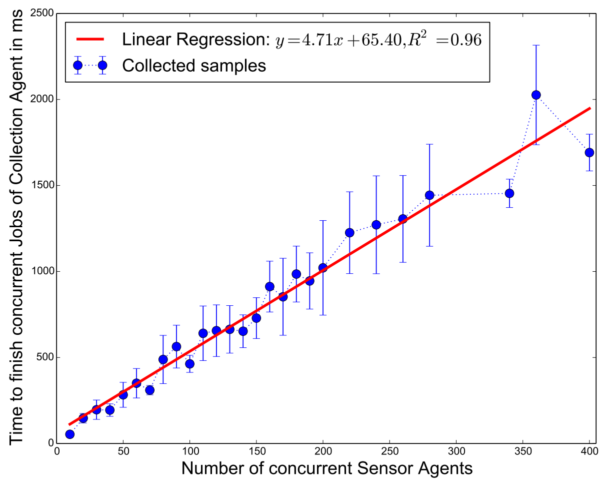

Fig. 6. Execution time measured at the monitoring server

Therefore, each sensor agent and collector agent will start an instance of a CEP engine upon their creation. We use Esper[4], a Java based CEP engine. After having been collected, the monitoring data will be stored in a NoSQL database by the collector agent. We choose MongoDB[5] as the solution for a database. MongoDB provides several benefits which address some requirements of our concept. First, we transform the monitoring data into JSON format, which is officially supported by MongoDB; second, MongoDB is a document-oriented database and does not require a user to specify a database schema in advance. As a consequence, MongoDB allows users to change and store different data structures during the monitoring process.

The evaluation scenarios are carried out using two Amazon EC2 micro instances[6] with the minimum configuration. Both instances run Ubuntu Server 12.04.1 LTS with an Intel(R) CPU E5-2650, 2Ghz, 8GB space, 0.613 GB memory. On the first instance, we deployed the server for the user to configure the monitoring template; the second instance was intended to be monitored. All in all, we emulated three monitoring scenarios, i.e., VM monitoring, application monitoring and holistic monitoring. VM monitoring emulates the collection of raw monitoring data while application monitoring emulates the monitoring tasks that aim to collect data related to the processes generated by a particular application. The holistic monitoring scenario emulates the collection of both, raw resource information and application processes related metrics. We collected CPU utilization, free memory, used memory and system uptime for the VM monitoring scenario. In the application monitoring scenario, we collected used memory and the uptime of the corresponding processes. The data collection in holistic monitoring contained all of the aforementioned metrics. Furthermore, we used two different data transmission intervals in the evaluation: one short interval with 5 seconds and one longer interval with 60 seconds. Each test scenario was repeated 20 times.

On the monitored instance, we measured the CPU overhead and network traffic generated by the monitoring system. The results are visualized in Figures 3 and 4. It can be observed from Figure 3, that the CPU overhead is negligible and smaller than 3% in all runs. There is almost no difference between the two dissemination intervals of 5 seconds and 60 seconds. This indicates that the CPU overhead generated by the system is not caused by the data transmission, but rather by the data collection process and by other dependencies such as the CEP instance. One noticeable fact is that the CPU overhead in the application monitoring scenario is very close to that of the holistic monitoring scenario but almost double compared to the VM monitoring scenario. This is to be expected because application monitoring and holistic monitoring need to look for a process and its corresponding monitoring metrics while VM monitoring only needs to read the monitoring data directly from the file system.

Besides CPU overhead, we also analyzed the amount of network traffic generated by the dissemination process at the monitored instance. For this test, we only measured the traffic rate for the 5 seconds interval scenario, since the amount of traffic will be obviously higher compared to the 60 seconds interval. In all three scenarios, the traffic rate at the monitored cloud instance is only around 0.4 KB/s and the standard deviation of the traffic rate values is smaller than 0.001 KB/s. Hence, the traffic rate for one day will be around 35 MB. All in all, the overhead on the monitored instances caused by the sensors and the corresponding dependencies for monitoring activities is small.

On the other cloud instance which represents a trusted third party, we are interested to assess the overhead caused by the collector agent when it has to handle more than one sensor agent. We emulated this scenario by starting multiple sensor agents and let them send monitoring data concurrently to the collector agent. We increased the number of sensor agents gradually to the limit that EC2 micro instance can still support. For the evaluation, we measured two types of overhead, i.e., the traffic throughput that one collector agent has to handle and the execution time that this collector agent needs for processing monitoring data. The execution time of

[4]http://esper.codehaus.org/
[5]http://www.mongodb.org/
[6]http://aws.amazon.com/de/ec2/instance-types/

the collector agent is measured from the point in time where the collector agent receives the monitoring data until it finishes writing the data into the database. The results are visualized in Figure 5 and 6. The highest throughput which can be observed from the test is caused by 300 sensor agents in the holistic monitoring scenario (300 sensor agents are the limit which an Amazon EC2 micro instance can still hold up to). Even in this case, the measured throughput is very small: 577.28 KB. Furthermore, with the increased number of sensor agents, the traffic throughput only increases linearly. The results measured for the execution time of the collector agent in Figure 6 clearly show that the collector agent is able to handle a great number of monitoring data in linear time. The execution time data sampled from the test cases can be fitted by the linear line $y = 4.71 * x + 65.40$ with $R^2 = 0.96$. This shows that the execution time of the collector agent only increases linearly with the growth of the number of sensor agents. This indicates that the collector agent can deal with the demand for scalability in cloud monitoring. In summary, one single collector agent can deal with the scalability in cloud computing; if we increase the number of collector agents, the overhead of each collector agent will be further reduced.

VI. Conclusion and Future Work

Since monitoring has been one of the major concerns for cloud computing in the recent years, there have been many research works that address different approaches for various aspects and requirements of cloud monitoring. The work at hand proposes a role-based template approach in conjunction with agents technology. Combining roles and templates allows us to abstract from the monitoring tasks compared to model based monitoring. Furthermore, the monitoring templates are reusable for the same roles under different scenarios. We also designed an abstraction model for the monitoring templates. The abstraction model of the templates can be represented by other formal modeling languages and standard syntax, which allows the automatic processing of the templates. A monitoring template created based on the demands of different roles helps a user to get started with the monitoring more easily. Even under the same role, templates can still be configured and adapted individually for different users with different demands. This ensures the flexibility of our approach. In addition, the monitoring roles also give us the ability to extend the template for security monitoring by specifying additional parameters such as access restrictions for different components during the monitoring process.

In our work, the core idea of role-based monitoring templates is further combined with agent technology. Agents with their autonomous behaviors and intelligence increase the flexibility in monitoring and permit dynamic configuration during runtime. With a monitoring template, an agent is given the intelligence to execute monitoring tasks autonomously and adaptively. The monitoring templates also allow the monitoring system to be further extended and coupled with other important components of cloud computing systems such as: analysis, planning and automatic reaction on violation of constraints. We combined role-based monitoring templates and agent technology with complex event processing to a holistic monitoring system for cloud computing that can also serve as a basis for further cross-layer extensions.

For future work, we will extend and improve our monitoring approach concerning the following aspects. We plan to employ machine learning techniques for the sensor agents to allow the sensor agents not only to be adaptive but also to be predictive towards monitoring tasks. Besides, we will work on the integration of our monitoring approach with other management processes of cloud computing, e.g., the monitoring template can be extracted from the result of the SLA negotiation process between a cloud consumer and a cloud provider. Finally, automatic actions can also be triggered depending on the detected events. Therefore, a complete MAPE (Monitoring, Analysis, Planning and Execution) cycle will be considered in our future work.

References

[1] A. Fox, R. Griffith, and A. Joseph, "Above the Clouds: A Berkeley View of Cloud Computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing (draft)," *NIST Special Publication*, vol. 800, 2011.

[3] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud Monitoring: A Survey," *Computer Networks*, vol. 57, 2013.

[4] B. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *INC, IMS and IDC, NCM, Fifth International Joint Conference*, 2009.

[5] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *Cloud Computing (CLOUD), IEEE 3rd International Conference*, 2010.

[6] L. Sun, J. Singh, and O. Hussain, "Service Level Agreement (SLA) Assurance for Cloud Services: a Survey from a Transactional Risk Perspective," in *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, 2012.

[7] J. Ahn, "Lightweight Fault-Tolerance Mechanism for Distributed Mobile Agent-Based Monitoring," in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, 2009.

[8] R. Aversa, L. Tasquier, and S. Venticinque, "Agents Based Monitoring of Heterogeneous Cloud Infrastructures," in *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, 2013.

[9] A. Liotta, G. Pavlou, and G. Knight, "Exploiting Agent Mobility for Large-scale Network Monitoring," *Network, IEEE*, 2002.

[10] "Nagios." [Online]. Available: http://www.nagios.org/

[11] "Ganglia." [Online]. Available: http://ganglia.sourceforge.net/

[12] V. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low Level Metrics to High Level SLAs - LoM2HiS Framework : Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments," in *High Performance Computing and Simulation (HPCS) International Conference*, 2010.

[13] V. Emeakaroha, T. Ferreto, M. a. S. Netto, I. Brandic, and C. a. F. De Rose, "CASViD: Application Level Monitoring for SLA Violation Detection in Clouds," in *Computer Software and Applications Conference (COMPSAC), IEEE 36th Annual*, 2012.

[14] B. Hoßbach, B. Freisleben, and B. Seeger, "Reaktives Cloud Monitoring mit Complex Event Processing," *Datenbank-Spektrum*, vol. 12, 2012.

[15] M. Smit, B. Simmons, and M. Litoiu, "Distributed, Application-Level Monitoring for Heterogeneous Clouds using Stream Processing," *Future Generation Computer Systems*, vol. 29, 2013.

[16] J. Povedano-Molina, J. Lopez-Vega, J. Lopez-Soler, A. Corradi, and L. Foschini, "DARGOS: A highly Adaptable and Scalable Monitoring Architecture for Multi-tenant Clouds," *Future Generation Computer Systems*, vol. 29, 2013.

[17] P. Leitner and C. Inzinger, "Application-Level Performance Monitoring of Cloud Services based on the Complex Event Processing Paradigm," in *Service-Oriented Computing and Applications (SOCA), 5th IEEE International Conference*, 2012.

[18] B. König, J. Alcaraz Calero, and J. Kirschnick, "Elastic Monitoring Framework for Cloud Infrastructures," *IET Communications*, vol. 6, 2012.

[19] F. Doelitzscher, C. Reich, M. Knahl, A. Passfall, and N. Clarke, "An Agent based Business aware Incident Detection System for Cloud Environments," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, 2012.

[20] "SLA & SLA Template Model (Abstract Syntax)," http://wiki.sla-at-soi.eu/@api/deki/pages/1970/pdf [last access 15.07.2014].

[21] M. Siebenhaar, U. Lampe, D. Schuller, and R. Steinmetz, "Robust Cloud Monitor Placement for Availability Verification," in *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER)*, 2014.

[22] M. Siebenhaar, D. Schuller, O. Wenge, and R. Steinmetz, "Heuristic Approaches for Robust Cloud Monitor Placement (accepted for publication)," in *Proceedings of the 12th International Conference on Service Oriented Computing (ICSOC)*, 2014.