2013 IEEE 10th International Conference on Ubiquitous Intelligence & Computing and 2013 IEEE 10th International Conference on Autonomic & Trusted Computing

Agents based Monitoring of Heterogeneous Cloud Infrastructures

Rocco Aversa, Luca Tasquier and Salvatore Venticinque
Department of Industrial and Information Engineering
Second University of Naples
Aversa, Italy
Email: {rocco.aversa, luca.tasquier, salvatore.venticinque}@unina2.it

***Abstract*—Monitoring of resources is one among the major challenges that the virtualization brings with it within the Cloud environment since the user applications are often distributed on several nodes whose location is unknown a priori and can dynamically change. Consumers need to monitor their resources for checking that service levels are continuously compliant with the agreed SLA and for detecting under-utilization or overloading conditions of their resources. Furthermore the monitoring of service levels becomes critical because of the conflicts of interest that might occur between provider and customer in case of outage. In this paper a framework that supports the monitoring of Cloud infrastructure is presented. It provides to the user the possibility to check the state of his/her resources, even if they have been acquired from heterogeneous vendors. The proposed environment will offer high elasticity and extensibility by the provisioning of an high level of customization of the performance indexes and metrics.**

I. Introduction

All Cloud services must be metered and monitored for cost control, chargebacks and provisioning.
Consumers need to monitor their resources for checking that service levels are continuously compliant with the agreed Service Level Agreement (SLA), and for detecting under-utilization or overloading conditions in order to eventually reconfigure the computing infrastructure also via new negotiations. Furthermore, in the Cloud context, the monitoring of service levels becomes critical because of the conflicts of interest that might occur between provider and customer in case of outage. Being the Cloud conceived as a market utility it is necessary to grant that vendor and consumer meet the conditions of the achieved agreement, which can have financial or legal consequences.
The monitoring of the resources is one among the major challenges that the virtualization brings with it within the Cloud environment. In order to ensure scalability and dependability, the user applications are often distributed on several computational resources, such as Virtual Machines, storages and so on. For this reason, the customer is able to retrieve information about the Cloud infrastructure only by acquiring monitoring services provided by the same vendor that is offering the Cloud resources, thus being forced to trust the Cloud provider about the detected performance indexes. Whereas in private Cloud all is available to the user, in a public Cloud customers can only access the virtual resources, meanwhile the provider manages hardware and hypervisor. This means that monitoring information are accessible only to providers for administration purposes, and eventually shared to the customers. Thus Cloud customers cannot check the compliance of the SLA trusting the monitoring service of their own same provider. In fact the Cloud provider has a conflicting interest ensuring the guarantees on service levels it provides. Furthermore in many cases the monitoring services offered by the vendors are located on a higher level of abstraction (platform or application level) and don't provide to the user an overview about the state of the acquired resources.
In this paper we present a framework that supports the monitoring of Cloud infrastructure, providing to the user the possibility to check the state of his/her resources, even if acquired from different vendors. The proposed environment will offer high elasticity and extensibility providing an high level of customization of the performance indexes and metrics.
The paper is organized as follows: related work is presented in Section II; in Section III the architecture of the proposed monitoring framework is presented, while in Section IV the developed environment is described; in Section V conclusion is due.

II. Related work

Monitoring of the Cloud infrastructures is a so relevant challenge that a new concept has been recently introduced within the contest of the service models: the *Monitoring as a Service* (MaaS) [1]. Having a monitoring service is an opportunity for both provider and user. First of all, the MaaS facilitates the monitoring by offering several functionalities and thus avoiding the development of ad-hoc tools. Besides, through the implementation of the "pay-as-you-go" model, the possibility to choose the monitored parameters and the detail's level of the measurements is given to the customer by taking into account his/her needings, as well as his/her available funds. Furthermore this service encourages the providers to invest in the monitoring field in order to provide improvements of both QoS and performances.

Most of the Cloud vendors provide tools to monitor the offered services. Other companies, even if not offering Cloud services, provide application that addesses the monitoring of distributed resources. Some of these solutions are "*agent-based*": this kind of monitoring system is based on the installation of a set of agents on the monitored infrastructure; these ones compute the measurements on the specific resource and send the data to a centralized collector. By contrary, the "*agent-less*" monitoring solutions are based on programs already installed on the monitored resources, without adding any software on them; all the monitoring computations, including

the measurements retrieval, are performed by a centralized server that is in charge of monitoring the environment [2].
Even if the agent-less solution has low maintenance costs and deployment time, the agent-based monitoring systems provide more tolerance to network and resource failure by mantaining the computed measures in case of service outage. Moreover a customized agent provide deeper and more specialized measurements with respect to the general purpose protocols used by the agent-less protocols, like SNMP [3].

For what is concerning the monitoring of distributed infrastructure, several commercial solutions exist. Infrastructure-level resource monitoring [4] aims at the measurement and reporting of system parameters related to real or virtual infrastructure services offered to the user (e.g. CPU, RAM, or data storage parameters). Traditional monitoring technologies for single machines or Clusters are restricted to locality and homogeneity of monitored objects and, therefore, cannot be applied in the Cloud in an appropriate manner [5]. Many available tools need to run on the physical machine and use facilities of the hypervisor. In a Private Cloud they can be used for management purpose, but in a Public cloud they are controlled by the Cloud vendor. At the state of the art there are many tools which provide Cloud monitoring facilities, like Rackspace Cloud Monitoring [6], CA Nimsoft Monitor [7], Monitis [8], Opnet [9], RevealCloud [10]. All of them are proprietary solutions and do not aim at defining a standard for monitoring interoperability.

As resource usage and the service workload can change very frequently, a continuous and updated answer must be provided to the user to efficiently monitoring the environment (e.g., a change in the answer must be detected as soon as possible while minimizing the communication cost). Furthermore the Cloud elasticity allows the possibility to change dynamically over time the system configuration, and so the distributed monitoring must adapt itself quickly to the new requirements. That is why we propose an agents based monitoring service. In [11] authors claim that an approach based on software agents is a natural way to tackle the monitoring tasks in the aforementioned distributed environments. Agents move and distribute themselves to perform their monitoring tasks.

Even though the proposed framework has been conceived as completely independent and usable by other architectures that tackle the Cloud interoperability issue, the presented work takes place downstream of another framework that addresses the provisioning and management of heterogeneous Cloud resources, the Cloud Agency. *Cloud Agency* [12], [13] is a multi-agent based system that is responsible for the Cloud resource provisioning and brokering. It has been designed and developed within the activities of the mOSAIC project [14]. Cloud Agency is enabled to interact with the Cloud providers and the customers in order to negotiate and broker the needed resources. Moreover, it provides interfaces and API to expose management services.

III. The Cloud Monitoring Framework

The *Cloud Monitoring Framework* allows the user to monitor his/her Cloud infrastructure by using an hybrid approach to perform the measurements and takes into account all the possible resources and configurations. An overview about the architecture is depicted in Fig. 1.

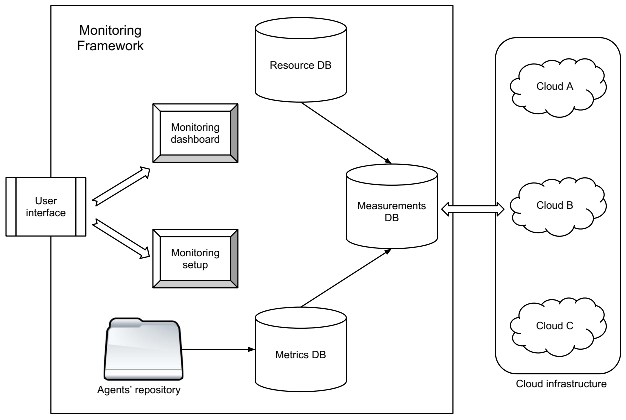


Fig. 1. Cloud monitoring framework

The *Resource DB* lists the resources that are part of the Cloud infrastructure. The resource description follows the hierarchy proposed by the Open Cloud Computing Interface (OCCI) [15]. OCCI is the Open Grid Forum proposal of a standard for the Cloud infrastructures level (IaaS). The main infrastructure types defined within OCCI infrastructure are *Compute* (information processing resources), *Network* (interconnection resources, belonging to L2 networking) and *Storage* (information recording resources) [16]. These top resource types are specialized by the following Link sub-types: *NetworkInterface*, that connects a Compute instance to a Network instance, and *StorageLink* that connects a Compute instance to a Storage instance. Each type is characterized by a number of attributes, which can be extended by using special data structures defined in the standard, called *Mixins*.
All the information about the Cloud configuration come from an upstream framework that is in charge of managing the resources, such as the Cloud Agency. The main information stored in this database are:

- *ID*: an unique identifier within the user's infrastructure;
- *type*: the resource type;
- *operating system*: in case of a Virtual Machine, this field stores the operating system characteristics;
- *provider*: the vendor that is providing the resource;
- *links*: in case of particular links among the resources of the Cloud infrastructure (e.g. between a Virtual Machine and a storage), these ones are listed by using this DB field.

The *Metrics DB* is in charge of saving information about possible customizable metrics collected on the target resources. The approach used to perform the measurements is agent-based, thus in the *Agents' repository* are stored the agents' implementations that wrap a particular set of measurements by using different technologies. The highly extensibility of the framework is due to a plug-in approach: if someone wants to add new measurements for a resource type, or wants to improve existing ones, he/she has only to develop the module that implements the measures by overriding and using an agent's common interface (Fig. 2). The agent is able to

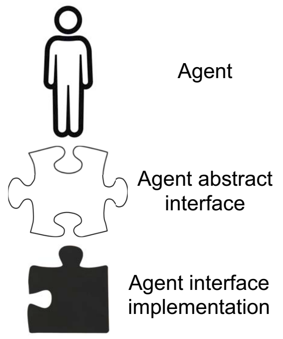


Fig. 2. Plug-in approach for agents' extension

communicate with the module by calling the functionalities exposed by the abstract interface, without taking care about the particular set of measures that it's performing; at the same time the new module developer can implement the new measurements without being aware about the agent technology. The Metrics DB keeps the connection among the resource types, the sets of available measures and the agents in charge of performing these ones. Besides this, the database contains, for each couple of resource type/provider, the list of the available proprietary or third-party tools that are able to monitor the specific resource.
The *Measurements DB* stores all the received information about the performed measures on the Cloud infrastructure. By querying it, it is possible to get an historical overview of the Cloud state and to map the results to higher level QoS parameters, also setting up reconfiguration policies. The *User interface* interacts with the customer in order to load the resources' configuration, set up the monitoring infrastructure and view the monitoring results. After retrieved the list of the resources and stored it in the Resource DB, it is possible to have two different scenarios. By joining the information in the Resource DB and in the Metrics DB, the Monitoring setup module shows to the customer resources' list together with the main other data, including the proprietary or customizable tools that is possible to use in order to take under control that part of the infrastructure. If the user selects a proprietary or third-party solution to monitor the resource, the scenario in Fig. 3 is implemented. The Monitoring setup module relates all the information about the resource and provides to the user the link of the chosen external tool that is in charge of checking the resource; in this case the Monitoring framework ends its work about the selected resource, delegating the monitoring efforts to the external tool.
In the second possible scenario (Fig. 4) the user selects, for a given resource, a set of metrics among the ones provided by the framework, according to his/her own needing. After choosing the metrics, the Monitoring setup module creates the related agent by retrieving it within the Agents' repository and customizes it to meet the user metrics' parameters. After this step, the meter agent migrates on the resource in order to perform the required measurements, thus providing all the benefits and aforesaid advantages of the agent-based approach. The sent measures are received by a collector that stores these ones in the Measurements DB. At this point the user can control the monitored resources by using the Monitoring

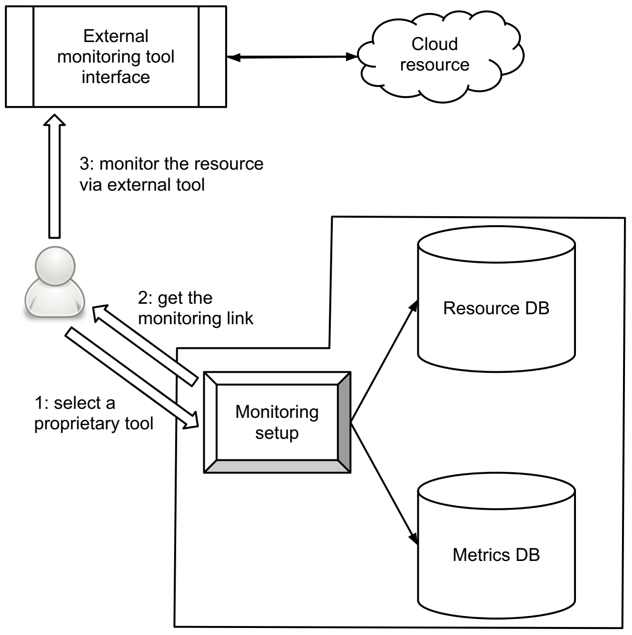

External monitoring tool interface
Cloud resource
3: monitor the resource via external tool
2: get the monitoring link
Resource DB
1: select a proprietary tool
Monitoring setup
Metrics DB

Fig. 3. First monitoring scenario: the user selects a proprietary/third-party tool to monitor a resource

dashboard. An important feature provided by the Monitoring

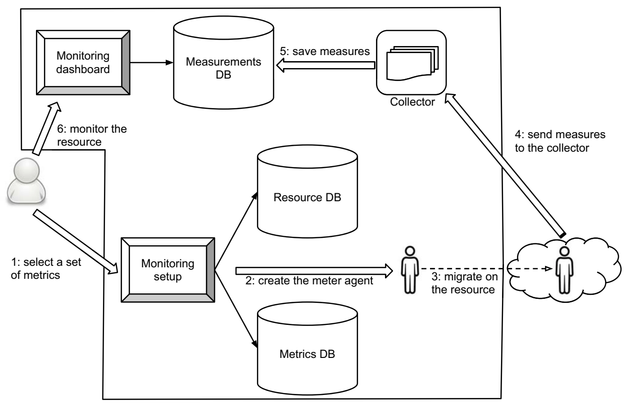


Fig. 4. Second monitoring scenario: the user selects customized metrics to monitor a resource

setup module is the usage of the information about possible links among the resources. In fact, if no metrics are provided to monitor directly a particular resource, metrics related to another resource linked to it can be exploited. To better explain this feature, let us suppose that the user infrastructure is composed by two storages (Storage_1 and Storage_2) and one Virtual Machine. Storage_2 is linked to the Virtual Machine, that uses the storage as additional disk space; let us also suppose that the Monitoring framework has not an agent that performs directly measurements on a storage resource, but among the customizable metrics exists the possibility to perform measurements on the disks attached to a Virtual Machine. For this reason, the Monitoring setup module shows only proprietary or third-party solutions in order to take under control Storage_1. Unlike what happened for Storage_1, by exploiting the information about the link between the Virtual Machine and Storage_2 the module is able to show also customizable metrics for Storage_2, which will be executed by using an agent that migrates on the Virtual Machine and performs measures on the attached disk. These measurements will be treated by the framework as measures performed on Storage_2.

A. Mapping of QoS parameters

A relevant issues about the monitoring of Cloud infrastructures concerns the mapping of low level measures, which can be obtained by installing probes distributed across the computing infrastructure, to high level parameters, which describe service levels. These QoS parameters require specific customization and elaboration on both the agent performing the measurements and the collector in the Monitoring framework.

An example of the QoS parameters that could be measured by the presented framework is the *Downtime* of a resource. In Service Level Agreements (SLAs), it is common to mention a percentage value that is calculated by dividing the sum of all downtimes time spans by the total time of a reference time span. 0% downtime means that the resource was available all the time.
In order to elaborate this metric, it becomes crucial to estimate the value of a downtime time span on a resource. With aim the agents can be properly programmed to behave as an heartbeat for the resource on which they are performing the measures. A solution to address the calculation of this QoS parameter is described as follow. The parameters taken into account to perform the measure are:

- P_m = measurements' period: it acts as sampling period for the measure;
- T_r = real time when the measures are sent. At the start of the measurements the value is the one read at the end of agent's migration;
- $T_{foreseen}$ = foreseen time in which the measurements have to be performed;
- ε = time that takes into account any delays related to the measurements' actions;
- T_d = downtime.

Together with this parameters, in the proposed algorithm other commands are used: *write()* to make persistent some data, *read()* in order to read some persistent data, *wait()* to block the agent's execution waiting for a given timeout. In order to recognize if an outage of the resource occurs, the agent executes Algorithm 1. The first *if-then-else* branch of the algorithm is performed each time the agent restarts its execution. If it has just ended the migration, it makes persistent the needed parameters, otherwise it restores the previously saved ones. After that it waits until the P_m timeout expires and performs the measurement operations. It is important to notice that if a failure of the resource occurs, at the restart of the agent the timeout for performing another measurement is reset: this makes, in case of resource outage, the read system time greater than the foreseen one.
At the collector side it is possible to compute the percentage of

```
if $migration\_ended()$ then
    $write(P_m)$;
    $write(\varepsilon)$;
    $T_r = read\_system\_time()$;
    $write(T_r)$;
end
else
    $read(P_m)$;
    $read(\varepsilon)$;
    $read(T_r)$;
end
while $true$ do
    $wait(P_m)$;
    $T_{foreseen} = T_r + P_m + \varepsilon$;
    $T_r = read\_system\_time()$;
    if $T_r <= T_{foreseen}$ then
        $T_d = 0$;
    end
    else
        $T_d = T_r - (T_{foreseen} + P_m)$;
    end
    $write(T_r)$;
end
```

Algorithm 1: Local downtime calculation

downtime by using the received downtime estimations coming from the involved agent and using Equation 1:

$$Downtime\% = \frac{\sum_{i=1}^{N} T_{d_i}}{T_a - T_i} \times 100 \quad (1)$$

where T_i is the time when the measure starts, T_a is the time when the downtime computation is required and N is the number of the received measures.
Since many times the SLAs express this QoS parameter in terms of Availability (A), it is possible to compute it by using Equation 2:

$$A\% = 100 - Downtime\% \quad (2)$$

The mapping of QoS parameters requires direct and indirect evaluations, combining information retrieved both at sender and receiver side. For this reason, the computation of other parameters is still an ongoing work; in this paper we stop the discussion about this topic here, presenting in the next section an implementation of the Monitoring framework.

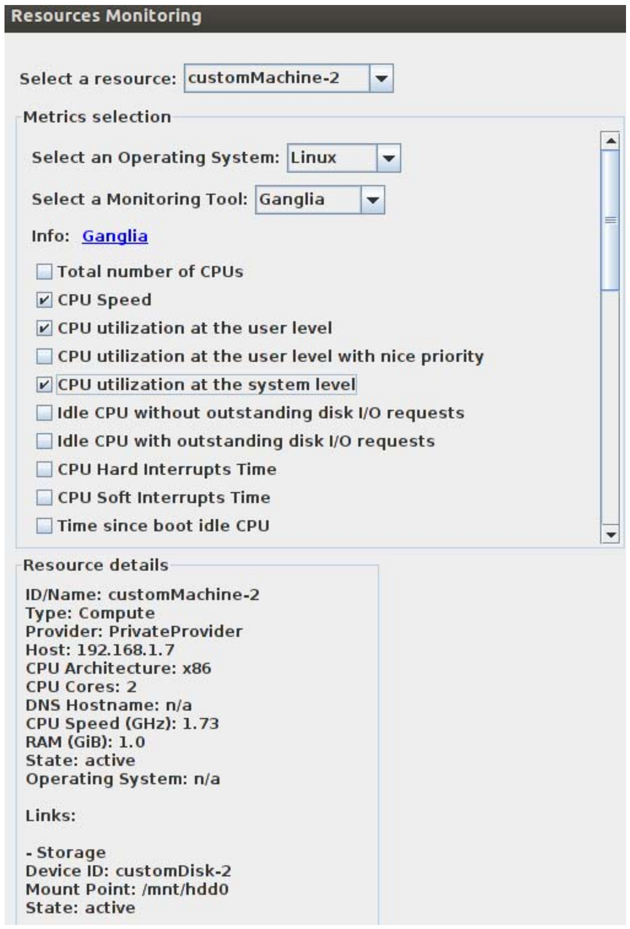
IV. The Monitoring environment

In order to implement the aforesaid framework, the first effort was spent on the usage of standards for the Cloud interoperability, in order to be compliant with them and maximize the compatibility with the upstream framework for resource brokering and managing. The implementation of the framework is consistent with the OCCI standard [17]: in fact all the Cloud infrastructure is described in terms of OCCI resources and attributes. As input of the Monitoring framework, the resources' description has been wrapped in XML format. The Monitoring setup module read this file and load the Resource DB, according with what it is described in Section III.
In order to understand the environment features, let us suppose

that the user infrastructure has only one storage (*amazonEBS-stor2*). If the customer wants to monitor *amazonEBS-stor2*, only proprietary or third-party tools are provided. By relating the information about the resource type and the provider, the Monitoring setup module offers the possibility to invoke the external tool CloudWatch [18]. A link to the proper web page appears by choosing this option. Let us consider now that the user infrastructure is composed by two compute resources (*customMachine-1* and *customMachine-2*) and one storage (*customDisk-2*); moreover *customMachine-2* and *customDisk-2* are linked together via a StorageLink. Supposing that for a storage there are not agents to monitor directly the resource, the Monitoring setup module suggests the customer to monitor the storage by using the Virtual Machine linked to it (*customMachine1*). In the proposed implementation, two meter agents have been developed. The former uses *Host-sFlow* [19] to perform the measurements. Host-sFlow provides scalable, multi-vendor, multi-OS performance monitoring with minimal impact on the systems being monitored. It implements the *sFlow* protocol [20] that is a standard for monitoring distributed infrastructures. The second agent wraps the *Ganglia* daemon [21]: it is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. Ganglia can be used also for collecting data coming from other daemons implementing the sFlow protocol to read and transmit the measures. Both the measurement's modules implement the abstract interface exposed by the mobile agent environment in order to be exploited by the Monitoring framework. The interface exposes these operations:

- *setup(user_customizations)*: the module has to implement this operation in order to allow the setup of the metering daemon with the user customizations;
- *setup_ended()*: this operation is called by the module to notify the end of setup operations to the wrapping agent;
- *start()*: this functionality allows to start the measures by the agent;
- *started()*: it is used by the module to notify the successful start of the measurement operations;
- *stop()*: the module has to implement this operation in order to stop of the measurements;
- *stopped()*: this operation is called by the module in order to announce the proper stop of the measurements;
- *send_measures(measures)*: this functionality allows the module to send the performed measures to the collector.

As previously described, the developer that wants to add features to the environment inherits several methods from the framework, while he/she can customize the agent's behaviour by setting up his/her module's functionalities.
The software platform chosen to develop the agents that wraps the modules is *JADE* [22]. It is an open source Java agent platform, compliant with the FIPA standard [23], supporting migration and communication among agents.
Once the user completes the metering customization, a new JADE agent is created. This agent loads the selected measurement module and calls the *setup()* method in order to perform


Resources Monitoring

Select a resource: customMachine-2

Metrics selection

Select an Operating System: Linux

Select a Monitoring Tool: Ganglia

Info: Ganglia

- [ ] Total number of CPUs
- [✓] CPU Speed
- [✓] CPU utilization at the user level
- [ ] CPU utilization at the user level with nice priority
- [✓] CPU utilization at the system level
- [ ] Idle CPU without outstanding disk I/O requests
- [ ] Idle CPU with outstanding disk I/O requests
- [ ] CPU Hard Interrupts Time
- [ ] CPU Soft Interrupts Time
- [ ] Time since boot idle CPU

Resource details

ID/Name: customMachine-2
Type: Compute
Provider: PrivateProvider
Host: 192.168.1.7
CPU Architecture: x86
CPU Cores: 2
DNS Hostname: n/a
CPU Speed (GHz): 1.73
RAM (GiB): 1.0
State: active
Operating System: n/a

Links:

- Storage
Device ID: customDisk-2
Mount Point: /mnt/hdd0
State: active

Fig. 5. Resource monitoring via customizable agent

its initialization and customization. After that *setup_ended()* is used by the module to notify the agent about the end of the setup operations. At this point the agent migrates on the resource and invokes the *start()* method in order to begin the measurements. This last invocation triggers on the module all the operations aimed at starting the measurements. After that, the module sends an acknowledge to the agent by using the *started()* method. This advice is forwarded by the agent to the collector that stores this information in the Measurements DB. Whenever the module wants to send the measures to the collector, it has only to invoke the *send_measures()* method. For example, in order to develop a module that implements the Availability estimation of a resource described in Section III-A, it is possible to setup P_m and ε parameters by overriding the *setup()* method, while the Algorithm 1 has to be implemented in the module's body, starting it by redefining *start()* method. All the protocol for beginning the measurements is described in Fig. 6.

After the metering setup, the user can take under control the selected resources by using the Monitoring dashboard that acts on the collector in order to retrieve and depict the asked resource parameters. The collector is able to query the Measurements DB to get the data about the selected resource and the chosen time span. In fact, the Monitoring

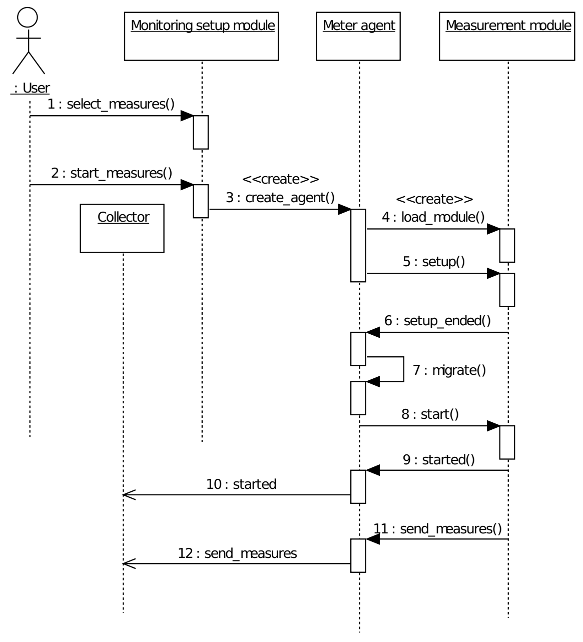


Fig. 6. Meter agent setup

dashboard gives the possibility to show and analyze historical data according to a user defined period. It also allows to get a real time figure about the selected metric on a parameter. In the latter case, the collector sends to the Monitoring dashboard the computed data each time it receives the updated measurements by the involved agent. The collector is also in charge of performing interpolations among the related data in order to retrieve QoS parameters, such as the availability and so on.

V. Conclusion

Cloud resources need monitoring facilities in order to allow the service levels' checking by the customers and to detect overloading or under-utilization conditions on the user's infrastructure. In the context of an easier, trustworthy and performance losslessly monitoring, we presented a framework to address the monitoring of Cloud infrastructure, by giving to the user the possibility to check the state of the resources, even if acquired from different vendors. The proposed environment offers high elasticity and extensibility by giving an high level of customization of the performance indexes and metrics; the framework uses agents mobility to perform the measurements on the selected resources, while a centralized collector computes performance indexes and QoS parameters. A prototype implementation of this framework has been presented as well.

An implication of this work will be the elaboration and implementation of new algorithms that allow the monitoring of SLA parameters. Another future field of research will be the exploitation of the information about links among the resources in order to perform an user application level monitoring, instead of an infrastructure level one. In addition to this, the monitoring framework can be exploited to trigger some reconfiguration policies based on verified critical conditions of one or more resource parameters. Autonomic reconfiguration agents could raise up and automatically reconfigure the Cloud resources in order to balance the infrastructure and resolve the detected critical conditions.

Acknowledgment

The authors would like to thank Emiliano Cimmino for his technical support.

References

[1] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," 2012.

[2] A. Kumar *et al.*, "AGENTLESS DATA COLLECTION," 2011, wO Patent 2,011,041,464.

[3] V. B. Pargaonkar and K. M. Ramakrishnan, "AUTOMATIC SELECTION OF AGENT-BASED OR AGENTLESS MONITORING," 2011, wO Patent 2,011,034,827.

[4] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. M. Vaquero, K. Nagin, and B. Rochwerger, "Monitoring Service Clouds in the Future Internet," in *Future Internet Assembly*, 2010, pp. 115–126.

[5] V. C. 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), 2010 International Conference on*. IEEE, 2010, pp. 48–54.

[6] "Rackspace Cloud Monitoring." [Online]. Available: http://www.rackspace.com/cloud/monitoring

[7] "CA Nimsoft Monitor." [Online]. Available: http://www.ca.com/us/lpg/nimsoft.aspx

[8] "Monitis." [Online]. Available: http://www.monitis.com

[9] "Opnet." [Online]. Available: http://www.opnet.com

[10] "RevealCloud." [Online]. Available: http://copperegg.com

[11] S. Ilarri, E. Mena, and A. Illarramendi, "Using cooperative mobile agents to monitor distributed and dynamic environments," *Information Sciences*, vol. 178, no. 9, pp. 2105–2127, 2008.

[12] S. Venticinque, L. Tasquier, and B. Di Martino, "Agents based cloud computing interface for resource provisioning and management," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 249–256.

[13] S. Venticinque, "User-centric infrastructure as a service by Cloud Agency," *MULTIAGENT AND GRID SYSTEMS*, vol. 9, pp. 157–159, 2013. [Online]. Available: http://dx.medra.org/10.3233/MGS-130204

[14] B. Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Máhr, and M. Loichate, "Building a mosaic of clouds," in *Euro-Par 2010 Parallel Processing Workshops*. Springer, 2011, pp. 571–578.

[15] T. Metsch, A. Edmonds, R. Nyrén, and A. Papaspyrou, "Open Cloud Computing Interface–Core," in *Open Grid Forum, OCCI-WG, Specification Document*, 2010.

[16] T. Metsch, A. Edmonds *et al.*, "Open Cloud Computing Interface–Infrastructure," in *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*, 2010.

[17] S. Venticinque, A. Amato, and B. Di Martino, *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*. PRT: SciTePress – Science and Technology Publications, 2012, ch. An OCCI compliant interface for IAAS provisioning and monitoring, pp. 163–166.

[18] "Amazon CloudWatch." [Online]. Available: http://aws.amazon.com/cloudwatch

[19] "Host sFlow." [Online]. Available: http://host-sflow.sourceforge.net

[20] "sFlow." [Online]. Available: http://www.sflow.org

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

[22] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE–A FIPA-compliant agent framework," in *Proceedings of PAAM*, vol. 99, no. 97-108. London, 1999, p. 33.

[23] "FIPA." [Online]. Available: http://www.fipa.org