A Service Oriented Monitoring Framework for soft real-time applications

Gregory Katsaros, George Kousiouris, Spyridon Gogouvitis, Dimosthenis Kyriazis, Theodora Varvarigou
School of Electrical and Computer Engineering, National Technical University of Athens
9 Heroon Polytechniou Str. 15773, Zografou, Athens, Greece
gregkats@mail.ntua.gr, gkousiou@mail.ntua.gr, spyrosg@mail.ntua.gr, dkyr@telecom.ntua.gr, dora@telecom.ntua.gr

Abstract
The advancements in distributed computing have driven the emergence of Service Oriented infrastructures that allow for on-demand provision of ICT assets. Taking into consideration the complexity of distributed environments, significant challenges exist in providing and managing the offered on-demand resources with the required level of Quality of Service (QoS), especially for real-time interactive multimedia applications. Monitoring mechanisms are a fundamental part in service-based platforms that support real-time QoS guarantees by providing coherent and consistent real-time attributes at various levels of the infrastructure (application, network, storage, processing). In this paper we present an architectural design and implementation of a complete monitoring framework for measuring QoS at both application and infrastructure levels targeting trigger events for runtime adaptability of resource provisioning estimation and decision making. We also demonstrate the operation of the implemented mechanism and evaluate its effectiveness using an application scenario, namely Film Postproduction.

Keywords: monitoring, service oriented architecture, Service Level Agreement, Quality of Service, real-time systems

1. INTRODUCTION
Research in the fields of Grid Computing, Service Oriented Architectures (SOA) as well as Virtualization technologies has driven the emergence of Cloud [1] service models such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). Furthermore, the appearance of different business roles according to this classification, potentially with differing interests, introduces new challenges with regard to the tools and mechanisms put in place in order to enable the efficient provisioning of services. Security, Quality of Service (QoS) assurance and efficiency are just a few issues that the providers are trying to tackle and integrate within the new products and services that they offer. In general, the SaaS provider offers an application as a service over a distributed environment to the end users. The PaaS Provider offers a service platform supporting storage and giving access onto various Cloud infrastructures, while the IaaS Provider offers resources on demand depending on each deployed instance. To this end, applications must be adapted in order to be executed on a service oriented and distributed infrastructure, describing it in a machine understandable way and enabling it in order to be executed on virtualized environments. Moreover, it involves maintaining a seamless flow of information from the bottom layers to the latter and an expected Quality of Service (QoS) level in cooperation with the IaaS Provider. In order to achieve this goal, an efficient, scalable and versatile monitoring mechanism is required in order to transmit the values of critical monitored parameters. The latter can be used for a variety of reasons like usage in performance estimation, monitoring of adherence to the terms agreed in the Service Level Agreement (SLA), statistical information to the end user etc. Given the nature of future internet applications (mainly referring to multimedia and interactive applications), monitoring mechanisms are considered to be a fundamental part in any system seeking to provide “real-time” services. Before discussing the challenges for monitoring mechanisms with regard to real-time systems, let us clarify the term “real-time”. Traditionally, “real-time” refers to hard real-time systems, where even a single violation of the desired timed behavior is not acceptable, for example because it leads to total failure, possibly causing loss of human lives. However, there is also a wide range of applications that also have stringent timing and performance requirements, but for which some violations of the timing constraints are acceptable provided these are well understood and carefully managed, as they lead to degradation in the provided QoS level. These are “soft real-time” applications and include a broad range of interactive and collaborative tools and environments, including concurrent design and visualization in the engineering sector, media production in the creative industries, and multi-user virtual environments in education.
and gaming. In this paper, we focus on interactive soft real-time applications where one or more users interact with the application and with each other. Our goal is to present a Service Oriented framework that enables PaaS providers to collect QoS parameters and effectively guarantee what is offered to the end-users. The latter has been highlighted as one of the main challenges from a group of experts discussing on the future of Cloud computing [2] but also from a PaaS Provider, namely Windows Azure [3].

From the aforementioned description of a state-of-the-art IT infrastructure, we can identify the challenges as well as the requirements of such mechanisms. While the majority of the contemporary high-end industrial applications are in principle component workflows that are being executed in different nodes, the monitoring mechanism must be able to operate in the context of this service-based environment. A sophisticated solution should be able to follow these components throughout the infrastructure in real-time with the workflow execution and forward the respective monitored information to the platform level. In addition, it must bridge the gap between the different layers of providers and the client, thus expanding across layer-specific boundaries. Furthermore, it should be able to adapt to the component requirements with regard to the granularity of the information collection. This depends on the nature / constraints of each part of the workflow and the type of QoS parameters that are needed to be monitored (and probably evaluated and treated). Another necessary feature is the combination of information from both the application (high level monitored parameters) and the infrastructure (low level monitored parameters) layer. What is more, the storage of these data must be performed in an efficient way so that they can be reused by other components of the PaaS layer such as performance estimation mechanisms.

In this paper, we present an architectural approach and implementation of a monitoring mechanism that addresses the challenges and meets the requirements described above. As depicted in the following figure (Figure 1) we propose a two-layer approach: a Monitoring Service that resides in the platform layer (PaaS) and an instance of the mechanism that is deployed in the virtualized environment (Monitoring Instance) along with the application components in order to collect monitoring information and relay it to the monitoring framework instance. This two-layer approach is of major importance since it minimizes communication delays between the virtualized environment and the platform services given that the information is collected and aggregated on the virtualized level. With regard to performance, the architectural design and implementation enables only one client (the Monitoring Service) to “query” the Index Service and according to the results presented in [4] – Figure 7 (“MDS4 Index service Response time Performance. Smaller values are better”) it achieves minimum response times (less than 1 sec), as also discussed in greater detail in Section V of our paper.

**Figure 1: Monitoring Framework Overview**

The remainder of the paper is structured as follows: section II presents the related work in the field of application execution monitoring and the solutions provided by other researcher and scientists. In section III we will present the architecture of the proposed monitoring mechanism while in section IV we will describe the baselines of the implementation. Finally, we will evaluate that solution through a selected use case (section V) and close with the conclusions and the future steps that are planned for the specific implementation.

**II. RELATED WORK**

Monitoring of distributed systems has been a topic of research interest and development for many years. Solutions like Ganglia [5] and MonALISA (MONitoring Agents using a Large Integrated Services Architecture) [6], offer distributed monitoring mechanisms for cluster and Grid infrastructures. Ganglia is based on a hierarchical design and relies on a multicast listen/announce protocol to monitor state within clusters. MonALISA is a service system, relying on Jini [13] and Web Services technologies. Both solutions focus on the scalability of the system and they are using common technologies such as XML, Java threads and WSDL/SOAP.

In the same context, GridICE [7] provides a monitoring Tool for Grid systems that promotes the adoption of standard Grid Information Service interfaces, protocols and data models. It provides different aggregations of monitoring data regarding the specific needs of different user categories with different abstraction level of a Grid (Virtual Organization level, Grid Operation Center level, Site Administration level and End-User level). In addition, Nagios [8] is a monitoring system that was designed to run checks on hosts and services using several external plugins and return status information to administrative contacts. While it includes valuable features and capabilities, it is destined for Local Area Networks.
(LAN) and cannot manage dynamic resources.

From the other hand, GrAMoS [9] in a monitoring service that runs over the Globus Toolkit 4 [12] middleware, monitors resources and detects agreement violations. The implementation is based on the WS-Agreement specification while it also interacts with Ganglia in order to retrieve resource information.

Another monitoring solution within Grid environment is the Relational Grid Monitoring Architecture (R-GMA) [11]. The R-GMA system is consisted of three components: the Consumers, the Producers and a Registry. The Producers publish their information to the Registry while the Consumer can query the latter in order to retrieve the information available. This general architecture has been designed for monitoring within computational Grids but could be used for other applications that require querying distributed pools and streams of data.

The RESERVOIR project [15], proposed the Lattice Monitoring Framework [10] based on the new requirements appeared for the Service Cloud monitoring. Lattice is designed to infrastructure as well as service monitoring and distinguishes the roles of monitoring Producers and Consumers. It utilizes multiple probes for collecting information and defines a distribution framework for publishing the collected data.

The mechanism we introduce in this paper advances the field of research in monitoring mechanisms and addresses the requirements for monitoring of virtual resources, multiple workflow applications components, different sources of information and other. However, one of the challenges not addressed by the approaches presented so far refers to the performance constraints during the collection and aggregation of monitoring data. These constraints are of major importance for real-time systems and thus our goal was to architect and implement a two-layered monitoring mechanism that fulfils this requirement.

III. PROPOSED APPROACH

In this section we present the proposed Monitoring mechanism. Our two-layered approach applies to Service Oriented platforms that encompass a “virtualization” layer, including Virtual Machine Units (VMUs). Within these VMUs the components related with the application workflow execution and the monitoring instance of the presented framework are deployed, as shown in the diagram below (Figure 2). The distinction between the two layers (Virtual Environment and Physical Infrastructure) is also portrayed, along with the components and their interactions.

**Figure 2: Monitoring mechanism architecture**

The mechanism consists of six components:

**Monitoring Framework Service (MFS):** This component is part of the platform’s framework. It exposes appropriate interfaces for starting / stopping the operation and has access to the collected data from both the infrastructure level and the application level. It is the major component of the mechanism while it orchestrates the monitoring of all applications towards the virtual environment and has access to the aggregated information through the Monitoring Central Index. Also, it can serve different and concurrent applications that are being deployed in separated virtual environments.

**Monitoring Service Instance (MSI):** It is located within the Virtual Environment (VMU) and is specific to every application workflow deployment. It exposes an interface towards the MFS in order to initiate the application monitoring. During that enabling action, the latter provides the configuration parameters for the application workflow monitoring, such as the private IP addresses of the VMUs, the time granularity of each component Data Collector etc.

**Data Collector:** is deployed in the same VMU with each application component in order to collect data from the application execution and publish them into the local Monitoring Index. The implementation of this component is part of the application adaptation to the Service-enabled infrastructure and is performed by the Application Developer. The Data Collector is automatically executed with a time interval that is defined by the MSI during the registration process. The outcome of this execution is a set of parameters (name, value, unit etc) formatted in plain XML and fed into the local Monitoring Index.

**Monitoring Index Service:** serves the role of a local repository of the monitored parameters for each application component. It is also deployed in the same VMU with an application component in order for the respective Data Collector to provide the necessary high level information. While it keeps the application monitoring parameters, it publishes them in the same time to the Central Monitoring Index, outside the Virtual Environment. The values within the Monitoring Index Service are being refreshed every time that the Data Collector provides a new set of parameters values.
**Monitoring Central Index**: it is the global repository of the platform. The high level monitored parameters of all application components as well as the low level parameters deriving from the infrastructure are published here. The values of the parameters listed in this repository are refreshed with a different time interval based on the report granularity of each individual application component.

**Infrastructure Monitoring Service**: this component collects low level information regarding the execution of the VMUs on the physical hosts and publishes respective reports to the Monitoring Central Index. This is part of the IaaS provider.

In more detail the sequence of interactions are:

1. The MFS enables the Infrastructure Monitoring by invoking the respective service. The latter collects low level information (e.g. CPU speed, hard disk usage, bandwidth etc) from the physical hosts where the VMUs of the application will be deployed.
2. The MFS enables application monitoring by invoking the MSI within the Virtual Environment.
3. The MSI registers the Data Collector of each application component of the workflow to its own Monitoring Index Service.
4. The Data Collector publishes the high level monitored information of the application component to the Monitoring Index Service. The values of the parameters are refreshed within the Index with a granularity defined during the registration.
5. The Monitoring Index Service forwards the high level monitored information to the Monitoring Central Index. By that, the information of all components of the Virtual Environment are being published on the central repository at the platform level (5b). Accordingly, the Infrastructure Monitoring Service delivers the low level monitored parameters to the Central Index (5a).

**IV. IMPLEMENTATION**

The implementation of our service-oriented mechanism lies on different layers as depicted in Figure 3. The Platform Services (Monitoring Framework Service and Monitoring Central Index) are directly deployed on a Physical Host. From the other hand, the execution of the Application Workflow (including all application components) will be executed within a Virtual Environment of the IaaS provider. The virtualization layer is implemented through the Kernel-based Virtual Machine (KVM) [16] virtualization engine. It was selected due to the fact that it is an open source hypervisor, with great mindshare and good support. KVM is also supported by major industry players such as IBM, Cisco, Intel, AMD, RedHat and others. The Virtual Machine Unit that will be deployed over a Physical Host is depicted with the orange color on the figure below and it is the layer where the Application Components will be executed.

![Figure 3: Layered architecture](image)

Furthermore, the implementation of this service-oriented monitoring mechanism was based on the Globus Toolkit 4 (GT4). GT4 is an open source Grid middleware that provides the necessary functionality required to build and deploy fully operational web services and will be installed within the VMU of each Application Component. This toolkit provides an Application Programming Interface (API) to support and implement the Web Services Resource Framework (WSRF), WS-Addressing, WS-Security, WS-BaseNotification specifications and other. In addition, GT4 offers a specific framework for monitoring resources and services within distributed environments. The Monitor and Discovery System (MDS) [14] of GT4 provides multiple extensible interfaces (Aggregator Framework) for querying WSRF services for resource property information, subscription mechanism for collecting data via WS Notification or execute an external program for acquiring data.

To this end, we developed the Monitoring Framework Service, Monitoring Service Instance and Infrastructure Monitoring Service as state-full WSRF services that are deployed in the standalone container that GT4 offers.

**A. Monitoring Service (Framework and Instance)**

The Monitoring Framework Service is a WSRF Globus service implemented in Java and deployed on the platform layer. As presented in the class diagram (Figure 4), there are operations for enabling the application monitoring as well as stopping it using proper Endpoint Reference (EPR) of the respective Monitoring Service Instance. In addition, the getData() operation is being executed within a thread in order to retrieve information from the Monitoring Central Index and store them to a MySQL database for future usage (statistics, performance estimation etc).
From the other hand, the Monitoring Service Instance is actually a resource of the Monitoring Framework Service. By the term “resource” we mean that an instance of the Monitoring Service have been created on the platform layer, configured and deployed in a VMU. This resource in then considered a Monitoring Service Instance within the execution virtual environment of an application and is specific for each application. The underlying concept is that each application workflow has its own configuration regarding the application components and the time granularity of the respective Data Collector and of course are deployed and executed as defined in the agreed Service Level Agreement (SLA). In this context, a Monitoring Service Instance includes all the details of each component (private IP, name, ID, Data Collector time interval etc). As a result, the Monitoring Service class will implement the RegisterCollector and UnregisterCollector operations that are being utilized from the Monitoring Service Instance when invoked. The registration of each Data Collector is done using the Aggregator Framework provided by the MDS. Using this API we can define certain parameters for the individual Data Collector through a XML configuration file (Figure 5).

As you can see in the XML file above, the execution interval (PollIntervalMills) of a Data Collector is configurable and could varies regarding the certain needs of each application component.

B. Data Collector

The Data Collector is actually an executable script written in any script language (perl, shell script etc) and is a responsibility of the Application Developer. This is due to the fact that he/she is aware of the component operation and the way of retrieving the monitored high level parameters (and which of them are more important). This component will be deployed within the VMU in order to interact with the Application Component and collect the necessary information. The only requirement, is that the script must generate a valid XML text including the ID of the component, a timestamp of the generated report as well as the information for the parameters (Figure 6).

As you can see in the XML file above, the execution interval (PollIntervalMills) of a Data Collector is configurable and could varies regarding the certain needs of each application component.
Monitoring Local Index

Monitoring Central Index

Aggregator Framework

Upstream Registration

Monitoring Local Index

Aggregator Framework

Infrastructure Monitoring Service

Figure 7: Aggregation of information from two sources

In more detail, our mechanism exploits a twofold flow of information (Figure 7):

- The Data Collector publishes the high level monitored data through the Aggregator Framework to the local Index. In this case we making use of the Execution Source functionality provided by the Framework. The Local Index through an Upstream Registration publishes the data to the Monitoring Central Index.
- The Infrastructure Monitoring Service, as a Subscription Aggregator Source, publishes the low level monitored information directly to the Monitoring Central Index.

D. Infrastructure Monitoring Service

Finally, the Infrastructure Monitoring Service is WSRF service that collects low level parameters from the Physical Host regarding the execution and communication of the VMUs and publishes them to the Monitoring Central Index of the platform. The service generates reports including information for the performance of the virtual nodes (VMUs) as well as the links between them (Figure 8).

\[
\text{<Avg\_used\_Bandwidth> <CPU\_load> }
\]
\[
\text{<Value>xxx</Value> <Value>xxx</Value> }
\]
\[
\text{<Unit>kbps</Unit> <Unit>%</Unit> }
\]
\[
\text{<Avg\_Delay\_mssec> <Phys\_RAM> }
\]
\[
\text{<Value>yyy</Value> <Value>yyy</Value> }
\]
\[
\text{<Unit>mssec</Unit> <Unit>%</Unit> }
\]
\[
\text{<Avg\_Jitter\_mssec> <Used\_volatile\_storage> }
\]
\[
\text{<Value>zzz</Value> <Value>zzz</Value> }
\]
\[
\text{<Unit>mssec</Unit> <Unit>%</Unit> }
\]

Figure 8: Infrastructure monitoring report examples

It is important that the report will include consistent identifiers with the ones used from the Data Collector during the high level monitoring, in order to be able to aggregate the information afterwards.

V. EVALUATION

A. Application Scenario

In order to validate the functionality and the performance of the proposed monitoring mechanism we selected the case study of a collaborative and distributed color correction that is performed as part of film post-production. In our scenario a post-production house is contracted to perform color correction to some film shots while in the same time the film director and the producer will review that footage. In this context and by applying the scenario to our previous architecture we end up with the following diagram:

Figure 9: Color correction scenario architecture

With the grey boxes (Figure 9), we depict the involved actors: the post production colorist that uses the Color Correction Station, one viewer for the film director and one for the producer. The application workflow also consists of a video storage, a variable number of Image Processing Units (IPUs) depending upon the data rates, a Load Balancer (LB) responsible for delivery of the stream to the views and an Image Processing Controller (IPC) that keeps the color correction parameters derived from the colorist. For our case study we used four IPUs instances deployed in an equal number of VMUs. The IPC and the LB were deployed within VMUs as well, while the Video Storage resided on the physical infrastructure. Furthermore, all the application components (IPUs, LB and IPC) were accompanied with their own Data Collector and Monitoring Index Service.

The execution flow of the scenario starts when each IPU queries the image processing control (IPC) for the current set of color parameters and caches them locally in order to be able to operate as quickly as possible on any frame. The LB instructs each of the four IPUs to process frame one by one. The IPUs retrieve the original, uncompressed frame from the Video Storage and apply the color correction parameters from the local cache. From the other side the Viewers can...
connect with the LB, who plays the role of a broadcaster as well, and review the footage as the LB “pushes” the frames with a certain rate. During that process, any of the involved actors (colorist or viewers) can pause the execution in order for the colorist to apply different color parameters and then start the streaming again.

B. Validation / results

After having deployed the six application components of the workflow (IPU1-4, IPC, LB) as well as the Monitoring Service Instance within the virtual environment, we proceeded by enabling the mechanism from the platform layer through the Monitoring Framework Service. In Table 1 we present the high level monitoring information that were delivered to the Monitoring Central Index from all application components of the workflow throughout our mechanism. We present the monitoring parameters for a certain point of the execution (timestamp: 2010-04-29 19:40:15) as each Data Collector provided.

<table>
<thead>
<tr>
<th>timestamp</th>
<th>ComponentID</th>
<th>ApplicationID</th>
<th>param_name</th>
<th>param_value</th>
<th>param_type</th>
<th>param_unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-04-29 19:40:15 IPC PP01</td>
<td>p_droppedframes</td>
<td>1</td>
<td>int</td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:15 IPC PP01</td>
<td>p_droppedframes</td>
<td>0</td>
<td>int</td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:15 IPC PP01</td>
<td>p_droppedframes</td>
<td>2</td>
<td>int</td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:15 IPC PP01</td>
<td>p_droppedframes</td>
<td>0</td>
<td>int</td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:20 IPC PP01</td>
<td>p_droppedframes</td>
<td>1</td>
<td>int</td>
<td>count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Example of different monitoring granularity

Even the fact that the monitoring information was arriving at the Monitoring Central Index asynchronously and the application workflow included six different components, the monitoring mechanism operated stably and efficiently during the whole time of the application execution. While the minimum monitoring granularity of the components in the scenario was 1 sec (for the LB), the upstream registration refresh interval between the Monitoring Indexes (within the VMUs) and the Monitoring Central Index was also set to 1 sec. With this specific set up, the response time of the monitoring mechanism starting from the components in the VMU to the Central Index at the platform was close to 1 sec (0.8 – 1.2 sec). This measurement was realized by disconnecting the Viewers from the LB during execution and count the time that the first report with $b_{clients}=0$ takes to reach the Monitoring Central index.

In order to observe and measure the tolerance of the system in terms of performance and efficiency, we repeated the execution by configuring the granularity of the application components as follows: 1sec for LB, 2secs for IPUs and 5 sec for IPC.

Table 3: Reports missed in execution period of 30 minutes

By counting the monitoring reports delivered to the Monitoring Central Index at the platform, for a period of 30 minutes of execution we resulted in missing nine monitoring reports for the LB, one for IPU2 and one for IPU4 (Table 3). Those results indicate that when the monitoring mechanism is stressed with a granularity of 1 sec and multiple components fails to deliver the reports with an error of 0.5% of the expected reports. From the other hand, as the granularity of the application component monitoring is increasing, and even with values of 2 or 5 sec, the percentage of failed deliveries is 0%.

Table 1: Color correction scenario monitoring results

As mentioned in a previous section, the mechanism that we implemented is configurable regarding the time interval of each Data Collector operation. The granularity of monitoring for each application component varies depending on the type of the component, the type of QoS parameter that we monitor and the importance of a component within the whole application workflow. In our case, the LB component that retrieves the processed frames from the IPUs and broadcasts them to the Client, is the most important component to monitor. As a result, we have set the monitoring time interval for that component to 1 sec while for the IPUs and IPC to 5 sec. In the table below (Table 2), we have selected one monitoring parameter for the IPC and one for the LB and we present the sequence of the reporting within a period of 8 sec.

Table 3: Reports missed in execution period of 30 minutes

By counting the monitoring reports delivered to the Monitoring Central Index at the platform, for a period of 30 minutes of execution we resulted in missing nine monitoring reports for the LB, one for IPU2 and one for IPU4 (Table 3). Those results indicate that when the monitoring mechanism is stressed with a granularity of 1 sec and multiple components fails to deliver the reports with an error of 0.5% of the expected reports. From the other hand, as the granularity of the application component monitoring is increasing, and even with values of 2 or 5 sec, the percentage of failed deliveries is 0%.
Regarding the infrastructure monitoring, the respective service reported the low level parameters for the resource and network usage. In the Table 4 we present some selected measurements for the components LB and IPU1.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>ComponentID</th>
<th>VMU_ID</th>
<th>param_name</th>
<th>param_value</th>
<th>param_type</th>
<th>param_unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-04-29 19:40:16</td>
<td>PUT1ToLB1</td>
<td></td>
<td>Avg_total_Bandwidth</td>
<td>10</td>
<td>int</td>
<td>kbps</td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>PUT1ToLB1</td>
<td></td>
<td>Avg_Jitter_msec</td>
<td>0</td>
<td>int</td>
<td>msec</td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>PUT1ToLB1</td>
<td></td>
<td>Avg_Delay_msec</td>
<td>30</td>
<td>int</td>
<td>msec</td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>LB1 VMU01</td>
<td>CPU_load</td>
<td>10</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>LB1 VMU01</td>
<td>Phys_RAM</td>
<td>20</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>LB1 VMU01</td>
<td>used_volatile_storage</td>
<td>0</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>IPU1 VMU02</td>
<td>CPU_load</td>
<td>30</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>IPU1 VMU02</td>
<td>Phys_RAM</td>
<td>80</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>2010-04-29 19:40:16</td>
<td>IPU1 VMU02</td>
<td>used_volatile_storage</td>
<td>0</td>
<td>int</td>
<td>%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Color correction Infrastructure monitoring results

It is important to mention that reports from both sources (Infrastructure and Application Components) must include some common identifiers in order to aggregate and to make use of the monitoring information in total afterwards. In our case study, the application components are reporting with the Component ID and Application ID while the Infrastructure Monitoring Service is reporting with the same Component ID and the VMU ID. By using those identifiers, for the monitoring of both high and low level of information that arrives on the Monitoring Central Index, the Monitoring Framework Service can refer to the SLA Manager that holds the agreed SLA of the application and exploits that information in any possible way.

VI. CONCLUSION

In this paper, we presented a two-layered Service Oriented monitoring framework that resides both on the platform services layer but also within the virtualized environment. The primary objective of the mechanism is to collect and aggregate monitoring information with regard to specific performance constraints as set by future internet applications, i.e. interactive real-time multimedia applications. The aforementioned information refers to data collected and combined from different sources: high-level application data and low-level resource data. There has to be noted that it is within our future plans to enhance the proposed mechanism in order to include an evaluation component to trigger events for runtime adaptability of resource provisioning estimation and decision making. Nevertheless, the operation of the proposed mechanism with regard to performance constraints achieves the constraints set by real-time applications, since collection and aggregation of monitoring data is performed within specific timing limits. The experiments showed promising results and therefore the performance of the mechanism is considered to be well established allowing the adoption of it in any heterogeneous and especially cloud-based system that seeks to provide QoS guarantees and facilitate “real-time” interactivity.

ACKNOWLEDGMENT

This work has been supported by the IRMOS [17] project and has been partly funded by the European Commission’s IST activity of the 7th Framework Programme under contract number 214777.

REFERENCES