Services Transactions of Cloud Computing (ISSN 2326-7550) Vol. 4, No. 4, October-December 2016

A KNOWLEDGE BASE DRIVEN SOLUTION FOR SMART CLOUD MANAGEMENT AND MONITORING

Pierfrancesco Bellini, Ivan Bruno, Daniele Cenni, Paolo Nesi
Distributed System and Internet Technology Lab: http://www.disit.dinfo.unifi.it
Department of Information Engineering, DINFO; University of Florence; Via. S. Marta 3, Firenze, Italy
[pierfrancesco.bellini, ivan.bruno, daniele.cenni, paolo.nesi]@unifi.it ; http://www.disit.org

Abstract

Cloud infrastructures needs to become more smarter and dynamic in managing resources, requiring new solutions and tools to cope with processes of configuration and reconfiguration, automatic scaling, elastic computing and healthiness control. This paper presents a Smart Cloud solution for automatically managing cloud infrastructures, and programming the monitoring aspects directly from the configuration and deploy phases. This means that the proposed solution can enforce smart cloud intelligence to almost all Cloud Configuration management systems and orchestrators by using REST calls and XML files. The solution proposed includes a Smart Cloud Engine, an advanced Supervisor and Monitor, and a Knowledge Base. The Knowledge Base is grounded on a cloud ontology modeling cloud resources and relationships, Service Level Agreements and their evolution, constraints, metrics for assessment, etc. The Knowledge Base enables the reasoning on cloud structures and permits to implement strategies of efficient smart cloud management and intelligence: (i) verify and validate the configurations and related changes according to the cloud model and available resources; (ii) easily generate the processing control for verifying cloud resource healthiness and service level agreement verification; (iii) take decision about dynamic reconfiguration on cloud, for example for scaling, balancing, migration, etc. The proposed solution has been validated in the context of ICARO project on complex cloud configurations with good performance and low operating workload. The validation has also assessed the reliability and the scalability.

Keywords: cloud computing; smart cloud; knowledge base, cloud ontology, cloud monitoring; elastic computing; scaling

1. INTRODUCTION

Almost all relevant infrastructures are using cloud based approaches to manage their resources, and set up high availability solutions addressing different layers such as IaaS, PaaS, and SaaS. Several different vendors are covering different aspects and supporting different services natively into the cloud solutions. Most of them provide specific products addressing only a limited number of features and services. On the other hand, the availability of a wide range of services is often the basis for selecting different cloud solutions. Among the requested services, there is the need of monitoring, changing, moving virtual machines and services in the same cloud for resource optimization and among different clouds to increasing reliability and for migration purposes.

Thus, cloud infrastructures are becoming every year more complex to be managed especially for process configure and reconfiguration, dynamic scaling for elastic computing, healthiness control, etc. Several thousands of different resource definitions are available and corresponding relationships among entities on the cloud can be established. Thus, every day new models and types are added, increasing complexity and demanding a very high level of flexibility in cloud management and definitions. These can be related to structures and resources on cloud (hosts, VM, services, storages, process, applications, nets, etc.), on their corresponding service level agreements, SLA; and on the metrics to be assessed for computing the business costs of the business on cloud in "as a service" basis.

Therefore, in order to add some level of intelligence to the cloud management a number of solutions have been proposed (in most cases called of Smart Cloud). They are grounded on the formalization of cloud modeling, service level agreement (SLA) modeling, cloud monitoring, and decision support reasoning about cloud structures and SLA taking into account of monitored data. Open cloud solutions such as: Eucalyptus, Nimbus, OpenNebula, etc., provides support for cloud monitoring and only marginally for SLA negotiation and enforcement. In most commercial solutions at level of IaaS, the SLA is not negotiable. It is simply proposed leaving the capability of setting specific service parameters: CPU, network, memory, etc. of availability. A review about SLAs offered by commercial Cloud Providers can be obtained from (Wu and Buyya, 2011). SLA brokers perform a part of the Smart Cloud work, such as in (Cuomo et al., 2012), where the SLA brokering is performed for cloud resource allocation and verification of resource consumption among several cloud providers.

The aim of the Smart Cloud solutions should be focused on: (i) formal verification and validation of resource for cloud configuration/reconfiguration (a-priori or a-posteriori with respect to the deploy; when it is a-priori, it can be regarded as a sort of simulation), (ii) controlling services and resources for present and reconfiguration allocation, (iii) making decision on horizontal or vertical scaling, reconfigurations, optimization, thus elastic computing, and in some cases of dynamic orchestration.

Moreover, the above described assessment phases need to be grounded on a detailed cloud model that may provide support for reasoning on cloud resource allocation, adaptation, optimization, simulation, workload, security, consumption, configuration, etc. In the literature, these aspects are addressed in different manners.

Starting from the cloud modeling. In (Youseff et al., 2008), a cloud ontology with limited expressivity has been proposed decomposing cloud into five layers: applications, software environments, software infrastructure, software kernel, and hardware. In (Zhang et. al., 2012) a solution to search services and resources in the cloud with CoCoOn ontology has been proposed. For the description at level of IaaS, the INDL ontology (Infrastructure and Network Description Language) defines nodes connected via links and interfaces in (Ghijsen et al., 2012). In mOSAIC project (Moscato et al., 2011), cloud knowledge modeling has been addressed to simplify the interoperability among cloud systems, presenting some lacks on the modeling connections. Linked-USDL in (Pedrinaci et al., 2014) provides a set of ontological model for the description of services, as SLA, security, price and intellectual property. At application level of the cloud, the standard OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), of (Binz et al., 2014; Binz et al., 2012), describe the application components, their dependencies, the plan (workflow) for provisioning on the infrastructure, thus formalizing the work of the Orchestrators. In (Bernstein and Vij, 2010), the ontological model is used for describing intercloud architectures adopting an ontology for describing cloud services. This approach is at the basis of the IEEE P2302 standard and exploits the mOSAIC model for the services and resources with the related limitations. Currently, there are a few efforts in building smart cloud solutions grounded on ontology on cloud computing (Androcec et al., 2012). In (Bellini et al., 2015), a review on cloud ontology modeling is provided.

In (Fang et al., 2015), SAMOS cloud service operation specification approach has been proposed that can be applied to diverse cloud service models and resource types. SAMOS enforce flexibility and interoperability to industrial elastic cloud solution as AWS and load balancers. Other solutions for managing heterogeneous clouds can be Cloud Data Imager (CDI) (Federici, 2014) for managing storage areas, (Povedano-Molina et al., 2013) for managing multi-tenant resources.

In the management of cloud infrastructures, the smart cloud engines have to provide some capabilities for assessing the best suitable configuration and allocation for planning the cloud reorganization for example in green cloud, or for simple scaling. This kind of capability is partially similar to that of cloud simulation that is mainly addressed as an offline process as in (Ahmed et al., 2014; Badii et al., 2016).

The adoption of a knowledge base approach to model the cloud knowledge with a cloud ontology and its instances can be a solution to enable the reasoning on cloud structures, and thus for implementing strategies of smart cloud management and intelligence. The proposed Smart Cloud solution can be easily exploited in connection with other cloud tools such as configurators, orchestrators, monitoring, etc. Thanks to the cloud ontology and Knowledge Base, the proposed Smart Cloud solution is particularly suitable for managing complex configurations, related SLA and related strategies for dynamic scaling and elastic computing. The proposed solution has been developed into ICARO cloud project and tested on the cloud infrastructure of Computer Gross. Computer Gross is a cloud service provider for IaaS, PaaS and SaaS, in which allocated applications as SaaS level are provided by several different vendors and belong to categories of multitier solutions for CRM (Customer Relationship Management), ERP (Enterprise Resource Planner), workflow, marketing, business intelligence, etc. This variety increase complexity in the cloud management, and motivate the need of flexible smart cloud engine. The validation phase of Smart Cloud solution proposed has been focused on assessing its effectiveness with respect to the automation of scaling, dynamic scaling, reconfiguration, and continuous verification of SLA and resource healthiness.

This paper is structured as follows. In Section 2, the Smart Cloud architecture is presented in relationships with the typical elements of cloud (configuration, orchestration, monitoring). In Sections 2.1 and 2.2 the Supervisor and Monitoring and the measuring metric approaches are presented. Section 3 describes the ontology at the basis of the Knowledge Base for smart cloud. In Section 4, the Smart Cloud Engine, which is the core element of the Smart Cloud is presented, together with its main features. Section 5 presents the validation of the smart cloud solution, putting in evidence the validation of configurations, the SLA verification process, and the data related to the experimental results performed in the Computer Gross CSP cloud. Conclusions are drawn in Section 6.

2. ARCHITECTURE

The Smart Cloud solution proposed addressed some of the challenges mentioned in the introduction. The solution is a knowledge base driven solution for smart cloud management, providing more flexibility and programmability with respects to state of the art and commercial solutions.

The architecture is reported in Figure 1. In most of the cloud management systems a set of ready to use configurations are offered to cloud buyers. They are

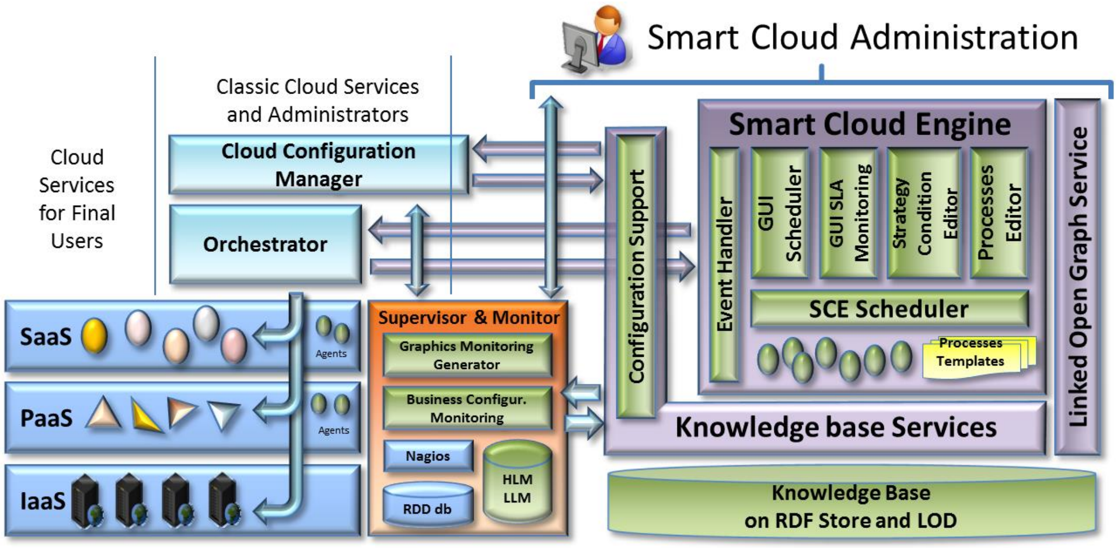
Smart Cloud Administration
Classic Cloud Services and Administrators
Cloud Services for Final Users
Cloud Configuration Manager
Orchestrator
SaaS
PaaS
IaaS
Agents
Agents
Supervisor & Monitor
Graphics Monitoring Generator
Business Configur. Monitoring
Nagios
RDD db
HLM LLM
Configuration Support
Smart Cloud Engine
Event Handler
GUI Scheduler
GUI SLA Monitoring
Strategy Condition Editor
Processes Editor
SCE Scheduler
Processes Templates
Knowledge base Services
Linked Open Graph Service
Knowledge Base on RDF Store and LOD

Figure 1. ICARO Smart Cloud detailed architecture

produced by a **Cloud Configuration Manager**, CCM. Simple, complex configurations, and changes of configurations, are typically deployed on the cloud by using an **Orchestrator** (for example VCO, MS, etc. as well as other open source solutions). The cloud expert programming the orchestrator has typically the duty of producing the program for configuring multiple monitoring and supervising activities, and in some case also the smart cloud, as in the IBM solution.

The main components of the architecture are the **Smart Cloud Engine**, SCE (see Section 4); the **Knowledge Base**, KB (see Section 3); and the **Supervisor and Monitor**, SM (see Section 2.1).

The **Configuration Support** of the SCE can be invoked by any CCM for: registering new business configurations and corresponding **Service Level Agreement**, SLA (that are stored into the knowledge base); requesting verification and validation tasks on a business configuration, and receiving back suggestions and hints related to consistency and completeness. For shortening the alignment of the SCE/KB with a VMware based cloud infrastructure in place, the Configuration Support of the SCE may directly access at the vCenter data of a vSphere solution. The **Event Handler** of the SCE can be invoked by the Orchestrator or by the CCM for: (i) requesting the control activation for monitoring and SLA and on some specific resource on cloud; (ii) activating SCE rules for cloud adaptation as dynamic scaling, cloning, moving, rules) for each business configuration according to some articulated firing condition; (iii) controlling healthiness of any business configuration and of any cloud resource and service. The SCE also includes editing tools for defining strategies (**Strategy Condition Editor**), and processes (by means of the **Processes Editor**). The processes are activated to perform periodic or on demand verification on: SLA, healthiness, etc. The processes are generated on the basis of templates and put in execution by the SCE Scheduler on a distributed set of node executors (located in virtual machines on cloud), thus obtaining a scalable and fault tolerant solution for smart cloud. To this end, the SCE tool presents a suitable user interface for: SCE process definition and management, SLA monitoring and assessment, value trend assessment, SCE strategies setup, etc. Moreover, for facilitating the formalization of semantics queries a suitable graphical user interface based on Linked Open Graph to access at the KB and browsing the semantic model has been used (Bellini et al., 2014).

The SCE exploits the KB in which configurations of cloud resources are registered/modeled. The **Configuration Support** of the KB receives the configuration requests and automatically programs the **Supervisor and Monitor**, SM to set up all the specific monitoring processes for controlling cloud services and resources. To this end, the KB may use multiple SM instances and the SM multiplied Nagios instances. This approach has a couple of advantages. Firstly, it simplifies the work on the Orchestrator since all the monitoring issues do not have to be programmed into the deploy workflow, also reducing that error prone process (but does not prevent to do them in any case). Secondly, it allows to be sure that the KB automatically add all monitoring issues that permit at the KB and thus at the SCE to have all needed information for controlling the business configuration collected data are received and accumulated in the KB. Other detailed data may be left accumulating and stored into the monitoring tools and services. The SCE exploits the cloud model contained into the KB, thus performing semantic queries in SPARQL (Prud'Hommeaux and Seaborne, 2008). Queries are performed for estimating firing conditions strategies (reconf, scaling, cloning, migration, etc.), verification and validation processes, estimation of high level metrics, and thus for controlling healthiness of each cloud resource, SLA and business processes. Thus, thousands and thousands of queries are executed per day, on a distributed scheduler of the SCE.

The SCE can be invoked and configured by the CCM as well as directly by some Orchestrator. In the validation case, it was directly managed by a higher level CCM addressing different kinds of orchestrators. As a general consideration, the proposed solution for Smart Cloud can be easily integrated with any CCM, and/or cloud Orchestrators, and Monitoring tools since the connections with these tools are performed by using REST calls and XML/JSON files.

2.1 Supervisor & Monitor

The Supervisor & Monitor, SM, is the subsystem for supervising and monitoring cloud resources at all levels: IaaS, PaaS and SaaS. The SM is able to monitor agent-less and agent-based physical machines (host, server hypervisor), virtual machines, network devices and storage, services and applications by using standard protocols such as SNMP (Simple Network Management Protocol), WMI (Windows Management Instrumentation), WBEM (Web-Based Enterprise Management), SMI-S (Storage Management Initiative Specification), (Ward and Barker, 2014).

The SM provides to the CCM graphs, list of meters to be used and integrated in presentation web applications and tools (such dashboard, monitoring panels), via specific API. The SM ensures interoperability and integration between the KB and the SCE through Restful APIs that allow:

- receiving monitoring configurations for each resource, service and application configurations from KB;
- remove and update one or multiple resources/services from the KB requests;
- activate or suspend monitoring for specific resources;
- provide metric values for each specific configuration, or single resource or application.

The SM itself provides an administrative console panel with a dashboard providing low and high level metrics trends over time with different kind of graphs. The SM allows browsing the monitored resources according to the configurations. The SM is an abstraction monitoring system that allows using existing open source and commercial monitoring tools. For the ICARO project purpose, the SM was built on Nagios Monitoring tools [Nagios].

2.2 Low and High Levels Metrics

The SM monitors cloud resources and activities sampling them by means of low and high level metrics.

Low Level Metrics, LLM, related to IaaS level such as CPU, memory, network bandwidth, storage; and to PaaS such as: Operating System of VMs, and services aspects as status and performances of: Web Server, FTP Server, HTTP, etc. In the case of ICARO solution, LLM are collected by Nagios (https://www.nagios.org/). Thus, LLM values are stored in a Round Robin Database provided by PNP4Nagios tool that allows managing historical data and drawing graphs.

High Level Metrics, HLM, combine values of LLMs to provide more concise and abstract measures of resources trends and status. For example, overall use of storage by a multi hard-disk application, complete memory usage of a multitier configuration, average uptime, inbound and outbound network traffic, maximum RAM memory used in the last 10 minutes and the average RAM memory used in the last 20 minutes.; and/or applicative metrics, such as: the number of registered users, web access counting, number of used services, web storage size, counters, etc. The HLM can use the last value or min/max/average/sum of LLMs over a time period (e.g. last 30 min) and combine them using the standard math operators (+, -, *, /).

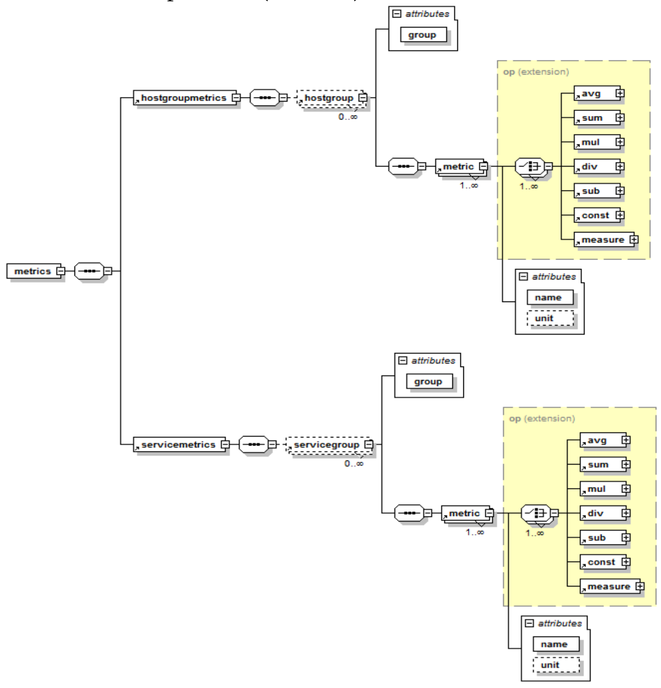

Figure 2. The HLM XML schema and model

HLM are defined by the XML schema reported in ***Figure 2***. The root *metrics* contains the *hostgroupmetrics* and the *servicemetrics* set. The former is the set of metrics for all *hostgroup* indentified by a name assigned to the *group* attribute, the latter is the set of metrics for all services in a *servicegroup*. Each set contains the own list of *metric* identified by *name* and with *unit* of measure. A metric is a combination of measures, operators (*sum*, *avg*, *div*, *mul*, and *sub*), constants (*const*) or simply a single measure. A *measure* is defined by:

- **lmetric**: it is the name of corresponding LLM.
- **perfdata**: the field is optional, it could be used to select a particular value in presence of multi-valued output.

- **operator**: it specifies the operator (min, max, avg, last) to be applied to the raw data retrieved by the historical LLM.
- **timeinterval:** it specifies the time range to use in terms of time unit (day, hour, minute). It is omitted when *operator* is set to *last*.
- **multivalue**: it is an attribute that allows choosing the avg|min|max|sum operator when the evaluation of measure provides multi values as in the case of the space available for all volumes on the machine.

The HLM evaluation is performed by a specific module automatically associated to all configurations when they are deployed. The plugin asks to the KB for the HLM definition associated with the configuration. It computes all metrics defined according to a depth-first approach on the tree of the metrics, then it goes down in the first subtree *hostgroupmetrics* and then in *servicemetrics*. In either case, for each child on a group, the subtree is computed (containing various children *metric*) by iterating on the metrics defined for that group. The computed values are sent to SM that stores them in the HLM database and sends their RDF representation to the KB.

Both, LLMs and HLMs, can be used for defining logical rules into SLAs. HLM are stored in the SM and forwarded to the KB where can be used for the verification of SLAs.

3. Knowledge Base and Ontology

The Knowledge Base, KB, stores the configuration of the whole cloud service ranging from the data center infrastructure to applications structure as well as the applicative metrics definitions and values. A review on KB usage in the context of cloud can be recovered on (Bellini et al., 2015). The adoption of KB enables the reasoning on cloud structures, and thus the implementation of strategies for smart cloud management and intelligence. The use of a KB facilitates interoperability among public and private clouds, and/or among different cloud segments managed by different cloud orchestrators or managers. The KB permits formal verification and validation of resource cloud configuration, discovering and brokering services and resources, reasoning about cloud security, computing capability for horizontal or vertical scaling, thus elastic computing. To this end, the KB needs to store, not only the structure of the cloud components (infrastructure, applications, configurations), and also the values of metrics of the components and their temporal trends to be able to answer question like ``which are the host machines that can allocate more **Virtual Machines**, VMs, with these features?" or ``which are the host machines over used in the last week? Which VM is exploiting resources closer to the bounds?". However, the storing of metric values in time on the KB (at the classical rate of 5 seconds as monitoring systems) can be too expensive and unnecessary. For this reason, HLMs have been defined to aggregate values of LLMs to use more significant indicators to represent the resource status at higher level (e.g., averaged value, number of time the maximum CPU usage % over last hour has been used). Therefore, only HLMs values are stored on the KB while the low level metrics are stored in the SM database (e.g., Nagios, in Resource Description Format, RDD format). Typically HLM can be estimated less frequently than LLMs, thus the number of values stored on the KB is limited. In the KB for the SCE we need to store both the application as a ``type" and the application instances. Moreover, each application can have specific constraints, as the number of services involved (e.g., number of front-end web servers). For this reason, in order to avoid to duplicate the type/instance relation (already modeled in RDF) and to leverage on the modeling features available in OWL2 to express constraints (e.g. max/min cardinality) the application model is profitably modeled as an OWL Class (McGuinness and Van Harmelen, 2004).

Another needs, it is the possibility to aggregate different applications, servers, virtual machines to build a complete **Business Configuration**, BC, including multiple hosts, VM and their details to model a complex requests of cloud customer (for example, a company requesting an integrated software solution including an ERP, Enterprise Resource Plan, with a CRM, Customer Relationship Management). Each of these BC may have its own single or multiple SLA and strategies for dynamic adapting the configuration and cloud consumption of resources. In some cases, BC may include multiple application tenants, while the VM containing them may be associated to a different BC.

In literature some ontologies has been proposed for cloud description, as discussed in Section 1. However, most of them are focused on the description of the cloud services with the aim of searching/matching cloud offers with cloud demand, or for cloud services discovery, and no one addresses all the needs expressed before.

For the above reasons, the ICARO cloud ontology has been developed allowing modeling the different aspects of a cloud service as: infrastructure description (e.g., host machines, virtual machines, networks, and network adapters), applications and services description, business configurations, metrics and SLA and also monitoring aspects. In ***Figure 3,*** the structure of the ontology is reported, the ontology is accessible at http://www.disit.org/cloud_ontology/core and more details can be found on http://www.disit.org/6568. The ontology is accessible as the code of the tool on GitHub/disit.

3.1 Cloud Resources on the Ontology

In the cloud ontology, the DataCenter concept is used to represent the part of the physical infrastructure available to provide cloud services. The Data Center is modeled as a set of HostMachines or HostMachineClusters connected through a Network.

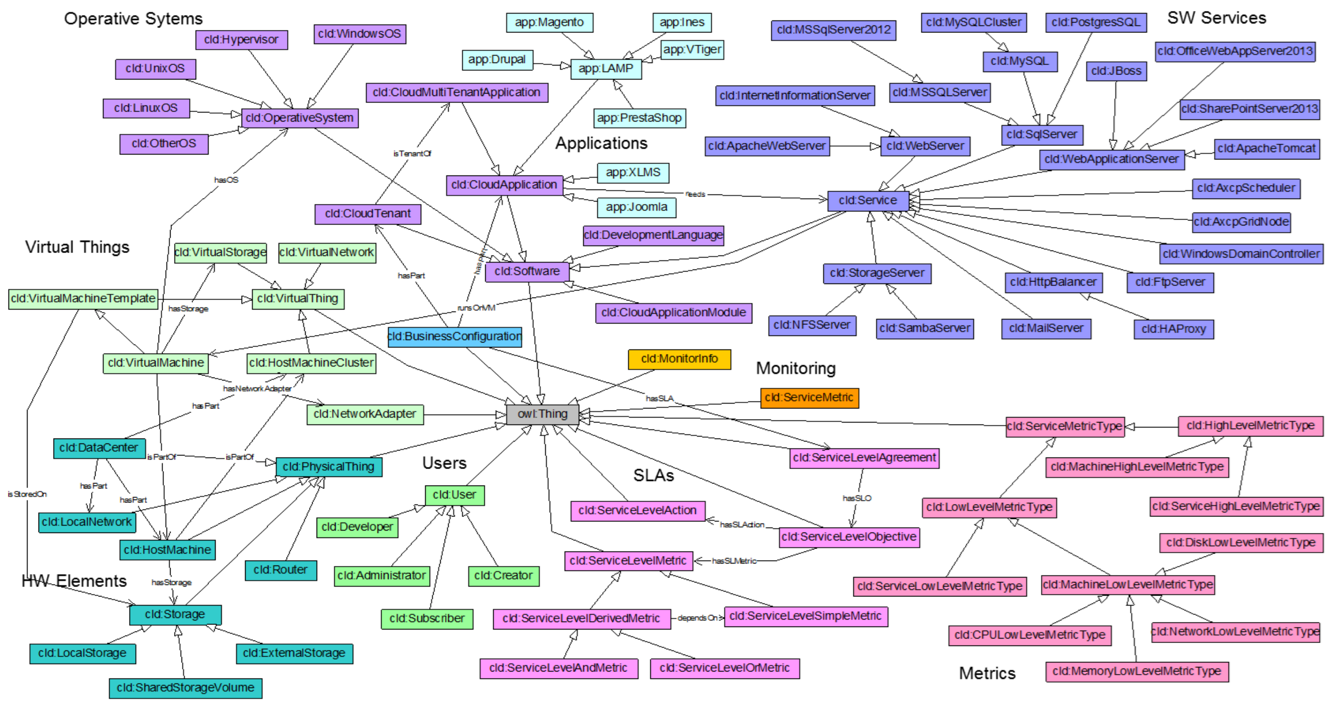

Figure 3. Main Classed and relationships of the ICARO Cloud Ontology for the KB

The different hosts and ExternalStorage elements as NAS/SAS, Firewalls and Routers are connected on the Network, all having some NetworkAdapter, by using a specific IP address. The HostMachine has a number of CPUs with a certain capacity (GHz), amount of RAM memory, NetworkAdapters and LocalStorage. A Hypervisor installed on the host allows sharing the different computing and storage resources among a set of VirtualMachines that are used to provide the different cloud services. More specifically VirtualMachines are characterized by a number of virtual CPUs, RAM capacity, one or more virtual disks on LocalStorage or on an ExternalStorage on the network and NetworkAdapters used to access the Network infrastructure.

Class CloudApplication is built by using many different Services as WebServers, HttpBalaners, ApplicationServers, DBMSs, applications caches, MailServers, NetworkFileServers that cooperate to provide the application functionalities. The CloudApplication uses a specific instance of each service and also a specific kind/version of the service (e.g. Apache Tomcat 5.5 ApplicationServer). An application may share some service with other applications (e.g. a single mail server handling the mails of several applications). Thus a CloudApplication can be deployed in many different ways on the different VirtualMachines, we can deploy all the needed services on a single machine or on multiple machines up to using a single virtual machine for each service. Moreover there are some constraints on the service in order to be used for the application as a specific feature is needed (e.g. web server supporting PHP scripting). The CloudApplications is represented in the KB both as "type" with its characteristics (e.g., the Drupal applications) and as application instances that are running on specific virtual machines. For this reason, we decided to represent the different cloud applications as subclass of a generic CloudApplication class and using the modeling features of OWL2 to describe the different constraints. For example, the application *DrupalBalanced* is represented (using Manchester notation) as:

DrupalBalanced SubClassOf **CloudApplication**
  and (needs exactly 1 **HttpBalancer**)
  and (needs exactly 1 **MySQLServer**)
  and (needs exactly 1 **NFSServer**)
  and (needs min 1 (**ApacheWebServer** and
    (*supportsLanguage* value php_5)))

meaning that a DrupalBalanced application is a subclass of CloudApplication that needs one Http Balancer, one *MySqlServer*, one NFS server and a minimum of one *Apache Web Server* supporting PHP scripting.

The *CloudApplication* class provides some generic aspects as the app identifier, application name, the developer, administrator and creator users, the SLA associated with the application and the *VirtualMachines* used, moreover the cloud application 'needs' only Services, other Cloud applications and cloud applications modules. These constraints are represented using Manchester notation as:

CloudApplication = **Software**
and (*hasIdentifier* exactly 1 string)
and (*hasName* exactly 1 string)
and (*developedBy* some **Developer**)
and (*developedBy* only **Developer**)
and (*createdBy* exactly 1 **Creator**)
and (*createdBy* only **Creator**)
and (*administeredBy* only **Administrator**)
and (*needs* only (**Service** or **CloudApplication** or **CloudApplicationModule**))
and (*hasSLA* max 1 **ServiceLevelAgreement**)
and (*hasSLA* only **ServiceLevelAgreement**)
and (*useVM* some **VirtualMachine**)
and (*useVM* only **VirtualMachine**)

Moreover, an application may be defined as subclass of *MultiTenantCloudApplication* (which is a sub class of *CloudApplication*). Thus, the application can be sliced in many tenants each one logically providing the functionalities of the whole application, while, in reality, only one application is deployed in the infrastructure. In this case, the multi-tenant application has associated many different tenants where each one could have its own SLA. The SLA, in this case, may assert a maximum number for an applicative metric (for example, a maximum number of documents allowed to be created from tenant users or the maximum number of users allowed to be registered).

3.2 MODELING CONFIGURATIONS

A Business Configuration, BC, represents a set of applications or application tenants, single virtual machines and even host machines that are bought from an user to collaborate on a given business process (e.g., multitier solution with scalable frontend/backend). Thus a BC refers to the application instances, services instances, DBMS, file storage servers, etc. needed by the applications and that are running on the several related virtual machines. A BC can contain one or more application tenants, and/or even a single host machine fully available for a single customer. A specific SLA is associated with the BC combining for example metrics from different applications, and represent the formal contract about the service level for the acquired BC. The definition and verification of the SLA is performed on the basis of metrics defined on the cloud and by the configurations under control.

All requests in the KB for managing new or modifying configurations are routed to the SM through the exposed API CRUD services. The incoming RDF of configurations is converted by means a XSLT transformation in the SM internal model and then stored in the SM database. All new or modified configurations are converted in the Nagios model. According to such model each host or virtual machine of the SM configurations is a host and it is assigned with a hostsgroup while each application service is a service and it assigned with a servicesgroup. Each of them is associated with the SM configurations by means the URI provided by the KB.

3.3 SLA AND HIGH LEVEL METRICS

A Service Level Agreement, SLA, is proposed by the CCM at the cloud customer and thus is part of the contract negotiation from the Cloud Service Provider, CSP, and the Cloud Customer who want to take some set of cloud resources according to the “as a Service” paradigm. The KB is capable to collect both LLM and HLM. This also means that in the SLA the CSP can provide contractual offers on cloud services by using any combination of LLMs and HLMs. In the SLAs formalization, not all the details need to be provided, since the general assessment may refer to general contractual aspects for default, for example the fact that host and VM will be in any case monitored to keep their resources under critical values.

The SLA is modeled as a set of Boolean expressions that relate low and/or high level metrics values of a component with a reference value (e.g., a VM average CPU use over last 30 min has to be less than 60% and the number of *httpd* processes running on a VM has to be greater than 0).

4. SMART CLOUD ENGINE, SCE

The Smart Cloud Engine, SCE, is the core part of the proposed solution for Smart Cloud according to the architecture depicted in ***Figure 1***. The SCE allows formalizing, finalizing and monitoring the processes that are executed in a distributed engine.

SCE is an autonomous engine for the supervised exploitation of cloud resources, for the automation and optimization of services. SCE implements rules that define automatic policies for the management of emergencies and events, to exploit resources distributed on various datacenters (e.g., to increase the current computational capacity of a system or for an automatic data migration). To this end, the processes of the SCE typically perform a periodic check of cloud resource status in the infrastructure (e.g., accessing to the KB/SM data, to get info about configuration, SLA, and current value of metrics about BCs, VMs, hosts, applications and services). In this scenario, the SCE central engine for task scheduling (SCE Scheduler) is a major requirement, which has to be scalable and reliable. The SCE processes connect to the KB by using SPARQL queries.

The SCE automatically performs queries on the KB to detect changes into the cloud configuration due to the arrival of new allocations and commands from the CCM. In this manner, the SCE identifies and get operating and bounding parameters for all the allocated cloud resources, BC, SLA, etc., and thus activate/reschedule related smart cloud processes. SCE periodically checks for metrics and services' status at IaaS, PaaS and SaaS levels of the cloud stack, knows the current general configuration of the system and the status of the cloud. The processes of SCE are related to the verification and computing of:

- cloud resource healthiness. In this case, the SCE provide a process rule for each major resource (e.g., host, VM, etc.) to verify if the resource consumption are under the critical thresholds, also taking into account HLM and not only the typically LLM adopted in monitoring solutions.
- BC status according to corresponding SLA. Each BC includes a set of cloud resources that are automatically monitored by the SCE. Moreover, the BC includes a SLA which may related changes in BC according to some parameters to adapt the provided resources to the workload. For example, when the front end workload is too high a new front end web server is added in balance to the present (scaling process), and may be the opposite to scale down and reduce the costs.
- condition rules for changing cloud configurations. These kinds of rules/processes are implemented as periodic processes that verify the conditions for dynamic reorganization of the cloud (firing conditions can be based on temporal constraints and/or on reference values of HLM/LLM). For example, according to the optimization workload by observing the status trend of cloud workload or even by using simulation forward (Badii et al., 2016).

When critical conditions are detected (according to provider firing rules) alarms are sent and/or actions are performed as planned. Among the possible actions, the invocation of the CCM and/or to the Orchestrator is also possible, with a REST or WS call. This approach allows activating procedures of rescaling, cloning, moving, balancing, reconfiguration (e.g., in memory, HD, network, CPU clocks), etc. Changes can be based on LLMs and/or HLMs reference values and parameters according to the SLA and strategy rules. The former are defined in the CCM, while the latter are formalized by the Strategy Condition Editor of the SCE.

4.1 SCE Scheduler

To cope with the above described process management and scheduling, of the SCE, a multiplatform scheduling engine with cluster functionality that allows adding new scheduling nodes and defining jobs without service downtime has been designed and developed. It is challenging to develop efficient solutions for task scheduling in a cloud environment, as can be seen in various examples (Sindhu and Mukherjee, 2011; Ma et al., 2013; Tsaia et al., 2013).

Each job has a name and a reference group, a fire instance ID, a repeat count, a start and an end time, a job data map, a job status (i.e., fired, running, completed, success, failed), and one of more associated triggers with their relevant data (i.e., name and group, the period and priority of execution). A job can be edited, deleted, stopped, paused, resumed, and manually triggered from the user interface or by direct REST/WS calls to the scheduler service (the same applies to triggers). Each scheduler's node can run a job with a lock mechanism, thus modifying the status of the corresponding trigger.

Each trigger is waiting for its fire time, and to be acquired by a scheduler's node; a paused trigger cannot put a job in execution; an acquired trigger has been selected by a node to be fired (once completed it can be rescheduled or deleted, if the job must be executed a finite number of times); a blocked trigger is prevented to be executed, because it is related to a job that is already executing. In order to keep running unresponsive jobs (e.g., jobs querying not available services) a timeout can be defined as well.

Similarly, the scheduler can be started, stopped and paused, and all of its associated jobs can be globally paused or resumed with a call to any of the scheduler's node, or through the web interface. In clustering mode, a job can be marked to request recovery, thus allowing job fail-over; in this way, during the shutdown of the scheduler (i.e. a process crashes, or the machine is shut off), the job is re-executed when the scheduler starts again.

Another major requirement that has been faced off is related to the concurrency of scheduled tasks. At this regard, it is important to specify the concurrency level of the scheduled tasks. Jobs can be defined to avoid concurrency; non-concurrent jobs avoid multiple executions of the same job at the same time (i.e., the job is disallowed to execute concurrently, and new triggers that occur before the completion of the running job are delayed). Each job activity can be monitored with the aim of a web interface that reports its status and other relevant data (e.g., failures and executions that completed with success).

SCE includes support for dynamic execution of jobs, constrained to the hardware or software resources at disposal; jobs can be bounded to execute only on selected hosts or, for example, depending on the current CPU load of the host itself, operating system or memory occupation. In this way, it is possible to schedule different tasks on nodes that are able to process them efficiently.

SCE can deal with multiple depending tasks by running a job, depending on the result of a previous one; then it is possible to define complex recursive chains of jobs that follow an execution flow chart. For example, it is possible to define a job that is executed when a set of condition is satisfied, sending a notification alert otherwise.

SCE includes a recovery system that allows defining policies to apply in case of misfired jobs (e.g., reschedule an existing job with existing or remaining job count), and allows graphing of monitored metrics, with customizable time intervals. For example, the scheduler could be instructed to fire a job as soon as it can, after a misfire event has occurred. For each metric it reports the total amount of times when the metric was found to be out of the requested bounds and the total number of checks performed. Metrics can be seen and graphed grouped per SLA or not. Single metrics provide the list of SLA violations occurred for them in the selected time period, with relevant data (e.g., the time at which the violation occurred, the name of the metric, the registered value, the threshold, and the related business configuration, virtual machine and SLA).

SCE reports a global view of the SCE Scheduler cluster status, with detailed views of each scheduler's node (e.g., last job execution time, number of jobs processed since the last restart, CPU load and utilization, time of last check, free physical memory, currently executing jobs, free disk space, operating system), and the total consumed computational capacity of the cluster (i.e., total CPU utilization, total capacity in terms of GHz and percentage of consumed capacity, total and free memory, number of CPU cores, total number of jobs executed by the cluster and total number of jobs executed on average per hour, number of jobs executed and relative percentages in the last day and in the last week, with respect to successfully completed and failed jobs).

4.2 SCE Monitoring and Strategies

SCE includes a status view of the currently running jobs (i.e., job's status, previous and next fire time, job data map, job's result or thrown exceptions, IP of the scheduler's node that executed or is running the job), a history log of the scheduler's activity, and a history log of the scheduler's nodes status. Direct monitoring of the above view is also at disposal, and the reporting period of each scheduler's node can be adjusted.

SCE provides a web interface that gives a view of the infrastructural status of the cloud platform (i.e., hosts, virtual machines, applications, metrics, alerts and network interfaces), with relevant details about the status of the SLAs (i.e., violations occurred and checks performed at a particular time), and a summary view of the global status of the scheduling nodes in the cluster. Graphing facilities are at disposal of the user to perform more deep analysis on the collected data (see for example ***Figure 4***).

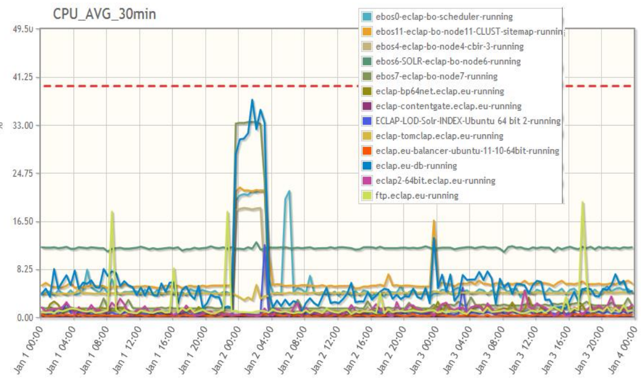

Figure 4. Metric graphs of a Service Level Agreement

In order to check the coherence of a monitored service with respect to its SLA, SCE periodically forwards requests to the KB including the identification of the SLA. Once evaluated the system status of interest (i.e., registered values of all the metrics included in the SLA, with respect to their thresholds, evaluated at the specified time), SCE eventually instructs the CM to take reconfiguration actions (e.g., increment of storage, computational resources or bandwidth), by sending a REST call to a defined endpoint, as specified in the SLA. SCE logs every event related to incoherent status of the monitored services. Checking periods can be adjusted for each service.

SCE integrates smart strategies for resource scaling and reconfiguration. For example, it could be required to partition the resources in a certain manner for administrative purposes (e.g., costs optimization, server maintenance or migration, uniform distribution of resources across the various datacenters) or to keep a certain quality level or in the optical of green cloud.

SCE allows defining elastic cloud strategies with Boolean expressions, via the Strategy Condition Editor. The user can build nested Boolean expressions to specify conditions on metrics or at various levels (i.e., virtual machines, business configurations). These expressions used to check the status of services and applications and their compliance to the SLA. Depending on the result, it is possible to issue scaling or reconfiguration requests to an endpoint. More formally the expressions are defined as:

Expr ::= **ALL** <Expr>+ | **ANY** <Expr>+ | <sExpr>
sExpr ::= "<metric name>" (**VM** | **SLA** | **BC**) <id>
<number>% (**ABOVE** | **BELOW**)
FOR <time> (**s** | **min** | **h** | **day** | **week** | **month**)

Where <metric name> is the name of the metric (e.g. "CPU AVG 30min"), <id> is the identifier for the virtual machine/SLA/BC where the metric is calculated; <number 1> *p* is the percentage of threshold *t* defined in the SLA for the metric; and <time> is the time duration that the metric is above/below the threshold percentage. Therefore, the condition is true, when the value *v*, of metric *m*, for *id* is $|v(m,id) - t| / t \times 100 <= p$ in the BELOW case (with > in the ABOVE case). The ALL condition indicates that all sub-expressions have to be true, and ANY means that at least one has to be true. For example:

ALL
"CPU AVG 30min" VM vmx18 30% ABOVE FOR 10min
"Memory Used AVG 30min" VM vmx12 10% ABOVE FOR 30min

Where metric: "*CPU AVG 30 min*" for virtual machine vmx18 has to be 30% greater than the threshold defined in the SLA for at least 10 minutes; and metric "*Memory Used AVG 30min*" for vmx18 has to be greater than 10% of the threshold defined in the SLA for 30 min. to be fired.

5. VALIDATIONS ON SMART CLOUD

In section some validation and verification processes performed in the SCE solution are described. They are related to: the continuous validation of configurations (see Section 5.1); the verification of SLA conditions, in section 5.2; and an example of data regarding experimental results in Section 5.3.

5.1 VALIDATING CONFIGURATIONS

The proposed KB provides REST services for storing and manipulating new configurations or changes in terms of: DataCenter, Application Types, Business Configurations, Metric Types and Metric Values that are stored on an RDF Store (currently an OWLIM-SE instance). Technically, the KB provides a SPARQL endpoint allowing making semantic queries.

As first step, when a new configuration request reaches the Configuration Support of the KB (in RDF-XML), its consistency is validated against the ontology formalization.

For the validation of the configurations submitted to the Configuration Support of KB, an OWL reasoner was not used, because of the Open World Assumption and not Unique Name Assumption used by OWL that do not allow consistency checks, but some SPARQL queries are used to check if a configuration is valid, similarly to (Sirin and Tao, 2009) where some OWL axioms are transformed to SPARQL queries to express integrity constraints. So for example the query to check in a business configuration (each configuration is stored in a different graph) to list all the virtual machines with storage that is not a LocalStorage or a SharedStorageVolume is the following:

```
SELECT ?vm ?s WHERE {
   GRAPH <…> {
     ?vm a cld:VirtualMachine;
          cld:isStoredOn ?s.
   }
   FILTER NOT EXISTS {
      { ?s a cld:LocalStorage. } UNION
      { ?s a cld:SharedStorageVolume. }
   }
}
```

Using SPARQL queries also max/min cardinality used for the *cld:needs* property can be checked as in the following:

```
SELECT DISTINCT ?app ?at ?t ?n ?c where {
  GRAPH <bc_graph> {
    ?app a cld:CloudApplication.
    ?app a ?at.
    FILTER(?at != cld:CloudApplication)
  }
  ?at rdfs:subClassOf [ owl:intersectionOf ?xx ].
  ?xx rdf:rest* ?yy.
  ?yy rdf:first ?f.
  ?f a owl:Restriction.
  ?f owl:onProperty cld:needs.
  ?f owl:minQualifiedCardinality ?n.
  ?f owl:onClass ?t.
      {
         SELECT ?x ?t (count(*) as ?c) WHERE {
            GRAPH <bc_graph> {
              ?x cld:needs ?s.
            }
            ?s a ?t.
         } GROUP BY ?x ?t
      }
  FILTER(?c<?n)
}
```

Moreover using SPARQL queries allow checking also other aspects that using the standard OWL2 language cannot be expressed. For example a SPARQL query may check if the host machine has space for the new virtual machine. And a SPARQL query may be also used to find the hosts where to allocate a new virtual machine. Since the SPARQL queries are stored in a configuration, it provides a high flexibility allowing changing the ontology to address new needs without changing the application code.

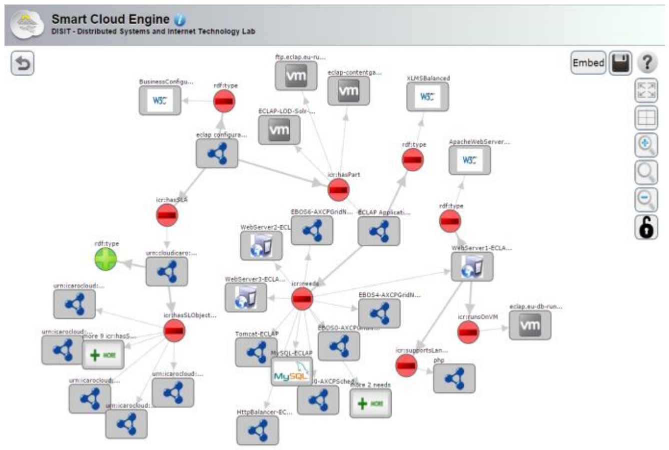

Figure 5. Linked Open Graph showing the ECLAP Business Configuration

For the validation of the system, and to visualize the different graphs and structures submitted to the KB, the Linked Open Graph, LOG, tool was used. LOG allows to visually navigating SPARQL endpoints and linked data services. LOG allows exploring the graph starting from a specific URI (Ghijsen et al., 2012) and the nodes can be progressively explored. The service allows graphs to be embedded in other pages. This service is used to explore the SPARQL endpoint of the KB. In ***Figure 5,*** an example with the structure of the ECLAP Business Configuration is shown embedded in a page of the Smart Cloud Engine. This service is publicly available at http://log.disit.org.

5.2 SLA Checking Process

The above presented Smart Cloud solution can cope with complex configurations, both at infrastructure and application level. In a typical scenario a set of requirements related to services/applications is modeled in the SLA document, and ingested in the ontology structure as detailed in Section 3.1. The SCE includes an autonomous engine for SLA checking and validation, that allows the creation of periodic executing jobs that verify both the integrity and the consistency of a deployed infrastructure.

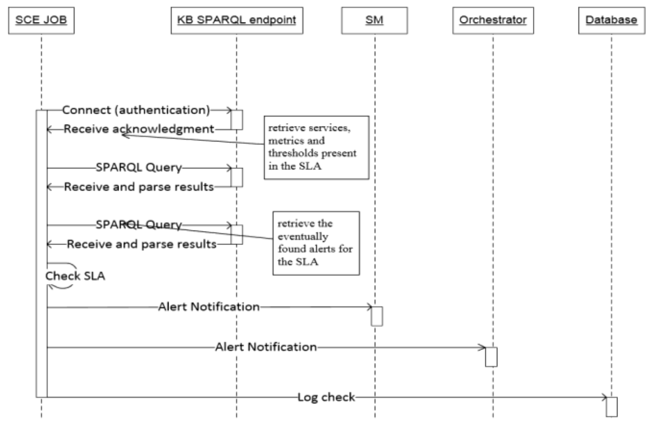

Figure 6. Sequence Diagram for SLA checking.

A sequence diagram detailing the process is depicted in ***Figure 6.*** Once triggered, a SLA-checking job connects to the KB SPARQL endpoint (authenticated or not) and issues a SPARQL query, to retrieve services, metrics and thresholds present in the SLA. Upon parsing of the obtained results (in JSON format), a second query is issued towards the SPARQL endpoint, to retrieve the eventually found alerts for the SLA. This is accomplished by retrieving the relevant values and metadata for each monitored metric of interest:

- *metric*, the metric ID;
- *metric name*, the label of the metric;
- *metric unit*, (e.g., kB, MB, GB, s, m, h);
- *timestamp*, the time at which the check was performed against the selected metric;
- *virtual machine*, the VM which the metric belong to;
- *threshold*, the threshold that must not be violated;
- *relation*, (e.g., less than, greater than, equal)
- *value*, the current metric's value;
- *bc*, the business configuration related to the metric;
- *URL*, the endpoint to be called upon verification of an alarm condition;

Every SLA consistency check is logged with the above mentioned details, and eventually notified to the user (e.g., by email with full details of the violated metrics). In case of metric values outside the bounds specified in the SLA, a notification URL is eventually called, as detailed in the SLA itself (i.e., with the correspondent parameter call URL), for example to inform the Orchestrator Service of the occurred event. Finally, it is possible to inform the SM, if a violation occurred with a POST notification to a specified endpoint.

The user interface for the job creation allows the user to specify the KB SPARQL endpoint to perform such checks, the SLA to be checked, and eventually a timestamp (i.e., the upper time limit) to be considered for the check (default is current timestamp). More complex decision rules can be produced in Java and SPARQL. These complex rules are regarded as Templates (see ***Figure 1***). For example, a complex rule may estimate the new configuration on the basis of some ARMA or ARIMA models for cloud resource prediction on the basis of weekly or monthly seasonal data (Chase, 2013).

5.3 Experiments and Validation

The KB driven solution for Smart Cloud described above has been developed in ICARO project and deployed on the ComputerGross partner cloud infrastructure for test and validation purposes. In that case, a datacenter with 36 business configurations were deployed on the system with 19 SLA and a total of 61 virtual machines. The most complex SLA has 75 conditions for an application (www.eclap.eu, a social network with scalable frontend and backend in balance with CDN, indexing & search and UGC ingestion) using 13 VMs and running 12 services (1 HTTP balancer, 3 Web Servers, 1 Apache Tomcat, 1 MySQL, 1 AXCP Scheduler, 5 AXCP Grid Nodes). With an history of service metric values of about 4 months (with about 3800 measures per metric), the time needed to evaluate a SPARQL query to evaluate the SLA and to get the current values for the metrics involved is of about 30 s, while for a SLA on a single VM with four conditions (with bounds on CPU usage percentage, memory, disk storage and network metrics) it takes about 2.0 s. At this date, about 27.84% of the recorded events were alarms. Averaged (AVG) data were recorded every 5 minutes and making the average every 30 minutes. From the data reported in ***Table 1***, it is evident that the vast majority of alarms recorded were related to memory (42.63%), disk usage (32.01%), network traffic (14.45%), database size (7.4%) and CPU usage (3.44%). The 73.1% of checks on memory metrics were alarms. For each reported alarm, a REST call was issued to a service that recorded the data of the event. The SCE Scheduler's cluster consisted of two computational nodes and a shared MySQL database.

As regards scalability, the costs connected to the monitoring, supervising and managing the cloud structures and resources have to be considered.

Table 1. Metric alarms recorded (Nov. 26 2014-Mar. 16 2015)

Service Metric monitored. Averaged values every 30 minutes, other values have been assessed every 5 minutes	% of overvalues with respect to the ref. values defined in the SLA	% on total of overvalue
Memory Used AVG	73,10%	42,63%
Disk Usage AVG	55,43%	32,01%
Network Traffic AVG	25,01%	14,45%
MySql DB Size AVG	60,02%	7,40%
CPU AVG	6,58%	3,44%
Apache http, response time	0,25%	0,04%
MySql Connections, response time	0,15%	0,02%
Check Apache Tomcat http, response time	0,09%	0,01%
Apache Process, num process	0,00%	0,00%
MySql Process, num process	0,00%	0,00%
Apache Tomcat Process, num process	0,00%	0,00%
AxcpGridNode Check Alive, response time	0,00%	0,00%
AxcpGridNode, num Processes	0,00%	0,00%
AxcpScheduler, num Processes	0,00%	0,00%
AxcpScheduler Check Alive, response time	0,00%	0,00%

On one hand, one of the main cost factors is the accumulation of monitoring data. The monitoring solutions typically collect data every 5 seconds and provide mechanisms for reducing data resolution over time, i.e., subsampling old measured values, thus introducing relevant errors. On the other hand, the Smart Cloud Engine needs to have HLM for longer time, for example for estimating the seasonal evolution and prediction. The estimation of HLM is typically performed every 30 minutes on the original data and not on the subsampled, thus reducing the errors and limiting the storage. In terms of storage, the KB for 6 months of work on a cloud data center cluster with about 120 hosts and a mean of 11 VMs for host, with a SLA every 3.4 VMs and about 35 services and metrics for each VM, lead to have a single non federated RDF store of about 95 millions of triples. The cost of the scheduling solution in terms of computational capacity per node (measured in CPU MHz) was extremely low (an average of 108.43 MHz). The adopted RDF stores as OWLIM can scale up on federated storages. In these conditions, monitoring VM healthiness, and SLA we had about 288,1 SCE processes per hour, executed on computational nodes. The success rate for the jobs scheduled is typically of the 98% in the week. The failures have been mainly due to the lack of connection during the restart of fail over solutions, almost all recovered. The SCE can save a lot of time for the operators that do not need to monitoring trends and data to making decision about business configuration adaptation according to the SLA.

6. Conclusions

This paper presented a knowledge base driven solution for smart cloud. The solution includes: a Supervisor and Monitor for cloud infrastructure at level of IaaS, PaaS and SaaS; a Knowledge Base for modeling cloud resources, business configuration with their SLAs, and all detailed resources in the cloud; and a Smart Cloud Engine. The availability of the Knowledge Base allows to: (i) verify and validate the configurations and related changes according to the cloud model and available resources; (ii) easily generate the processing control for verifying cloud resource healthiness and SLA verification; (iii) take decision about dynamic reconfiguration on cloud, for example for scaling, balancing, migration, etc. The SCE has been described and the core parts put in evidence providing validation section in which the most interesting validating aspects have been discussed. The SCE solution proposed is based on a scalable SCE scheduler together with the Knowledge Base provide smart reasoning support and management for cloud platforms and can be applied to different contexts (e.g., hybrid or federated clouds), supporting load balancing and fault tolerance policies, automatic scaling and elastic computing features. The Knowledge Base is capable to models cloud resources, Service Level Agreements and their evolution, and enabling the reasoning on structures by implementing strategies of efficient smart cloud management and intelligence. The proposed solution has been validated in the context of ICARO project on complex cloud configurations with good performance and low operating workload. The validation has also assessed the reliability and the scalability as shown in Section 5. The limits of the solutions are mainly located in the limited complexity of the decision rules that one can formalize directly in the user interface. More complex decision rules can be also produced by formalizing them in Java and SPARQL. The mechanism of templates for rules have been defined and provided to satisfy these complex cases. The proposed solution is based on simple REST call and thus can be easily exploited and integrated with other cloud configuration and orchestration tools. In addition, the Supervisor and Monitor tool can exploit and integrate multiple monitoring tools at IaaS level.

7. ACKNOWLEDGMENT

The authors want to thank all the partners involved in ICARO project (ComputerGross and LiberoLogico), and the Regione Toscana for funding in the POR CReO 2007 - 2013. A specific thanks to the many coworkers involved in the development, and to the industries during the trials and the final assessments. Details of these phases and related workshops are reported on DISIT Lab portal Http://www.disit.org, while the code of SCE, SM and KB are accessible on GitHub/disit.

8. REFERENCES

[1] Wu L., Buyya R., "Service Level Agreement (SLA) in utility computing systems", in Cardellini, V., et al. (eds) Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions. IGI Global, USA (2011).

[2] Cuomo A., Di Modica G., Distefano S., Puliafito A., Rak M., Tomarchio O., Venticinque S., Villano U., ”An SLA-based Broker for Cloud Infrastructures”, Journal of Grid Computing, March 2013, Volume 11, Issue 1, pp 1-25, 2012.

[3] Youseff L., Butrico M., and Da Silva d., "Towards a unified ontology of cloud computing", in Grid Computing Environments Workshop, 2008. GCE '08, pages 1-10, Nov 2008.

[4] Zhang, M, Ranjan, R, Haller, A, Georgakopoulos, D, Menzel, M, Nepal, S. "An ontology-based system for Cloud infrastructure services' discovery". 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 , pp.524,530, 14-17 Oct. 2012.

[5] Ghijsen M, van der Ham J, Grosso P, de Laat C. “Towards an Infrastructure Description Language for Modeling Computing Infrastructures”. In The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.

[6] Moscato F., Aversa R., Di Martino B., Fortis T., Munteanu V., "An analysis of mOSAIC ontology for Cloud resources annotation", Federated Conference on Computer Science and Information Systems (FedCSIS) pp.973,980, 18-21 Sept. 2011.

[7] Pedrinaci, C, Cardoso, J, Leidig, T. [2014] Linked USDL: a Vocabulary for Web-scale Service Trading, 11th Extended Semantic Web Conference [ESWC 2014], Springer. http://linked-usdl.org/.

[8] Binz T, Breitenbücher U, Kopp O, Leymann F. “TOSCA: Portable Automated Deployment and Management of Cloud Applications”. In “Advanced Web Services”, Springer, 2014.

[9] Binz T, Breiter G, Leymann F, Spatzier T. “Portable Cloud Services Using TOSCA”, IEEE Internet Computing, May 2012.

[10] Bernstein D, Vij D. "Using Semantic Web Ontology for Intercloud Directories and Exchanges". In Proc. International Conference on Internet Computing, 2010.

[11] Androcec D, Vrcek N, Seva J. ”Cloud Computing Ontologies: A Systematic Review”. Proc. of MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, Chamonix, France, April 29, 2012.

[12] P. Bellini, D. Cenni, P. Nesi, ”Cloud Knowledge Modeling and Management”, Chapter on Encyclopedia on Cloud Computing, Wiley Press, 2015.

[13] Fang D., Liu X., Romdhani I., Pahl C., An approach to unified cloud service access, manipulation and dynamic orchestration via semantic cloud service operation specification framework, Journal of Cloud Computing, 2015, 4:14, doi:10.1186/s13677-015-0039-3 l

[14] Federici C (2014) Cloud data imager: a unified answer to remote acquisition of cloud storage areas. Digit Investig 11(1):30–42

[15] Povedano-Molina J, Lopez-Vega JM, Lopez-Soler JM, Corradi A, Foschini L (2013) DARGOS: a highly adaptable and scalable monitoring architecture for multi-tenant Clouds. Future Gener Comp Syst 29(8):2041–2056

[16] Ahmed, A.; Sabyasachi, A.S., "Cloud computing simulators: A detailed survey and future direction," IEEE International Advance Computing Conference (IACC), pp.866-872, 21-22 Feb. 2014, doi: 10.1109/IAdCC.2014.6779436

[17] C. Badii, P. Bellini, I. Bruno, D. Cenni, R. Mariucci, P. Nesi, "ICARO Cloud Simulator Exploiting Knowledge Base", Journal: Simulation Modelling Practice and Theory, Elsevier, 2016, 10.1016/j.simpat.2015.12.001.

[18] P. Bellini, P. Nesi, A. Venturi, ”Linked Open Graph: browsing multiple SPARQL entry points to build your own LOD views”, International Journal of Visual Language and Computing, Elsevier, 2014.

[19] Prud’Hommeaux E, Seaborne A. SPARQL query language for RDF. W3C recommendation. 2008 Jan 3;15.

[20] McGuinness DL, Van Harmelen F. OWL web ontology language overview. W3C recommendation. 2004 Feb 10;10(10):2004.

[21] S. Sindhu, S. Mukherjee, ”Efficient Task Scheduling Algorithms for Cloud Computing Environment”, High Performance Architecture and Grid Computing Communications in Computer and Information Science Volume 169, 2011, pp 79-83.

[22] L. Ma, Y. Lu, F. Zhang, S. Sun, ”Dynamic Task Scheduling in Cloud Computing Based on Greedy Strategy”, Communications in Computer and Information Science Vol 320, 2013, pp 156-162 .

[23] J. Tsaia, J. Fanga, J. Chou, Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm, Elsevier, Computers & Operations Research, Vol 40, Issue 12, 2013, pp 3045-3055

[24] Sirin E., Tao J., ”Towards Integrity Constraints in OWL”, In Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009.

[25] Jonathan Stuart Ward, Adam Barker, ”Observing the clouds: a survey and taxonomy of cloud monitoring”, Journal of Cloud Computing 2014.

[26] Chase CW. ARIMA Models. Demand-Driven Forecasting: A Structured Approach to Forecasting, Second Edition.:203-37, 2013.

Authors

Paolo Nesi is a full professor at the University of Florence, Department of Information Engineering, chief of the Distributed Systems and Internet Technology lab and research group. His research interests include massive parallel and distributed systems, physical models, semantic computing, object-oriented, real-time systems, formal languages, and computer music. He has been the general Chair of IEEE ICSM, IEEE ICECCS, DMS, WEDELMUSIC, AXMEDIS international conferences and program chair of several others. He is and has been the coordinator of several R&D multipartner international R&D projects of the European Commission such as RESOLUTE, ECLAP, AXMEDIS, WEDELMUSIC, MUSICNETWORK, MOODS and he has been involved in many other projects. He is the ICARO Cloud project coordinator. He has been co-editor of MPEG SMR.

Pierfrancesco Bellini is a Researcher and Aggregated Professor at the University of Florence, Department of Information Engineering. His research interests include object-oriented technology, real-time systems, formal languages, computer music. Bellini received a PhD in electronic and informatics engineering from the University of Florence, and has worked on projects funded by the European Commission such as: RESOLUTE, ECLAP, AXMEDIS, MOODS, WEDELMUSIC, IMUTUS, MUSICNETWORK, VARIAZIONI and many others as ICARO. He has been co-editor of MPEG SMR, and the cloud ontology and KB responsible in the ICARO project.

Ivan Bruno is a research fellow at the University of Florence, Department of Information Engineering. His research interests include object-oriented technology, cloud monitoring systems, grid computing, computer music. Bruno received a PhD in electronic and informatics engineering from the University of Florence, and has worked on projects funded by the European Commission such as: ECLAP, AXMEDIS, MOODS, WEDELMUSIC, IMUTUS, MUSICNETWORK, VARIAZIONI and many others as ICARO cloud.

Daniele Cenni is a research fellow at Department of Information Engineering of the University of Florence. Cenni received a PhD in Telematics and Information Society from University of Florence. His research interests include smart city, cloud engines, social networks, semantic database, semantic computing, recommendations, and clustering. Cenni received a degree in informatics engineering from the University of Florence, and has worked on projects funded by the European Commission such as: ECLAP, AXMEDIS and VARIAZIONI, and research projects as ICARO.