A Knowledge Base Driven Solution for Smart Cloud Management

Pierfrancesco Bellini, Daniele Cenni, Paolo Nesi
Distributed Systems and Internet Technology Lab http://www.disit.dinfo.unifi.it
Department of Information Engineering http://www.dinfo.unifi.it
University of Florence, http://www.unifi.it,Florence,Italy
Email: {pierfrancesco.bellini, daniele.cenni, paolo.nesi}@unifi.it

***Abstract*—**

***Keywords*-cloud computing; smart cloud; knowledge base; elastic computing; scaling;**

I. 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. To this end, the modeling and formalization of cloud resources and information are becoming more relevant to formalize different aspects of a cloud at its different levels: IaaS, PaaS, SaaS, and towards specific resources: hosts, virtual machines, networks, memory, storage, processes, services, applications, etc., and their relationships. 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. A review about SLAs offered by commercial Cloud Providers can be obtained from [5]. The aim of the smart cloud solutions can be focused on: (i) facilitating interoperability among public and private clouds, and/or among different cloud segments managed by different cloud orchestrators or managers, (ii) formal verification and validation of resource cloud configuration (a-priori or a-posteriori with respect to the deploy; when it is a-priori, it can be regarded as a sort of simulation), (iii) discovering and brokering services and resources, (iv) computing cloud simulation, (v) reasoning and adapting cloud workload conditions, (vi) reasoning about cloud security, (vii) computing capability for horizontal or vertical scaling, thus elastic computing. In the literature, these aspects are addressed in different manners. Some of them into SLA brokers such as in [6], [1], while for the minimal monitoring you can see [9]. The work presented in the paper reports a Smart Cloud solution based on a Knowledge Base, KB, definition for modeling cloud resources, models, SLA, and their evolution. 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. In the seminal work [3], an approach to create a cloud ontology has been proposed decomposing problems into five layers: applications, software environments, software infrastructure, software kernel, and hardware. In mOSAIC EC project [4], the cloud knowledge modeling has been addressed with the aim of creating a common model to cope with the heterogeneity of different clouds vendors, and with systems with different terminology. Currently there are several efforts in building smart cloud solutions grounded on ontology on cloud computing [2]. The proposed Smart Cloud solution can be easily exploited in connection with other cloud tools such as configurators, orchestrators, monitoring, etc. Thanks to the KB, the proposed Smart Cloud 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 II, the ICARO Smart Cloud architecture is presented in relationships with the typical elements of the cloud. Section III describes the Knowledge Base ontology for modeling the cloud model and enabling reasoning and configurations, status conditions, and SLA. In Section IV the structure and solution for the Smart Cloud Engine is presented. Section V depicts some experimental results that can give you the effectiveness of the solution proposed. Conclusions are given in Section VI.

II. ARCHITECTURE

The proposed Smart Cloud solution addresses some the above issued presented in the introduction, coping with complex business configurations deployed on the cloud and coming from multiple software providers. It 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. In most of the cloud management systems a set of configurations are offered to the cloud buyers and produced by a Cloud Configuration Manager, CCM. These business configurations, as well as the 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), that has also the duty of setting up the monitoring and supervising activities, and in some case also the smart cloud, as in the IBM solution, with limited capabilities. Most of the commercial Smart Cloud solutions have limited capabilities in setting up scaling rules (vertical and horizontal, elastic, etc.), while have limited capabilities in defining and detecting complex conditions for activating changes in the cloud. The architecture of the proposed Smart Cloud solution is reported in Figure 1. It centrally includes a Smart Cloud Engine, SCE, that can be invoked by any CCM or Orchestrator, for: (i) registering new business configurations and corresponding SLA; (ii) requesting verification and validation tasks on a business configuration, and receiving back suggestions and hints related to consistency and completeness; (iii) requesting the control activation for monitoring and SLA; (iv) activating reshaping (e.g., dynamic scaling, cloning, moving, rules) for each business configuration according to some articulated firing condition; (v) controlling healthiness of a business configuration and each related resource and service.

In order to perform these activities the SCE exploits (i) a Knowledge Base, KB, in which business configurations and cloud models are registered, (ii) a set of strategies. The process of modeling the cloud knowledge allows at the KB to automatically program the Supervisor and Monitor service to set up all the specific monitoring processes for controlling services and resources. To this end, in our

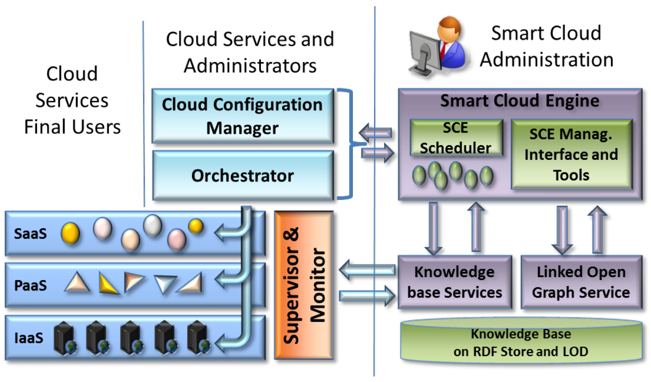

Cloud Services Final Users
Cloud Services and Administrators
Smart Cloud Administration
Cloud Configuration Manager
Orchestrator
Smart Cloud Engine
SCE Scheduler
SCE Manag. Interface and Tools
SaaS
PaaS
IaaS
Supervisor & Monitor
Knowledge base Services
Linked Open Graph Service
Knowledge Base on RDF Store and LOD

Figure 1. ICARO Smart Cloud Architecture

installations, the KB Services specifically use drivers to manage multiple Nagios instances (not discussed in this article). 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 do not prevent to do then in any case). Secondly, it allows to be sure that the SCE may automatically add all monitoring issues that permit at the SCE to have all needed information for controlling the business configuration behavior related to SLA, reshaping strategies, and of all resources involved. Moreover, the SLA are typically based on high level metrics and applicative metrics, such as: the number of time the workload exceeded a threshold for 20 minutes, the number of users registered per day, the number of contemporary streaming processes, etc. Once the monitoring is activated the useful collected data are received and accumulated in the KB. Other detailed data may be left accumulating and stored into the monitoring tools and services. In order to fastening the alignment of the Smart Cloud solution with a VMware based infrastructure already in place, the SCE also present a process to directly access at the VCenter data of a VSphere solution. The SCE exploits the cloud model contained into the KB, thus performing semantic queries in SPARQL. 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 query processes are executed per day, on a distributed scheduler of the SCE. These Smart Cloud processes are put in execution on a distributed and parallel architecture comprised of a set nodes (on virtual machines and a distributed scheduler), 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 [8]. 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 connection with these tools are performed by using REST calls and XML/JSON files. 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.

III. KNOWLEDGE BASE

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 [7]. The KB needs to store not only the structure of the cloud components (infrastructure, applications, configurations) but also the values of metrics of the components and their temporal trends to be able to answer question like ”which host machines can allocate a new VM?” or ”is a host machine over used? Which VM is using most resources?”. However storing the full history of all metric values on the KB can be too expensive and unnecessary. For this reason high level metrics has been defined in order to aggregate values of low level metrics (CPU%, memory used, disk available, etc.) to produce an indicator that is used to represent the resource load (e.g. average or maximum CPU usage % over last hour). Considering that only high level metrics values are stored on the KB while the low level ones are stored in the monitoring service (e.g., Nagios) and that high level metrics can be estimated less frequently than low level ones the number of values stored on the KB is reduced. Another aspect is related with applications, we need to store in the KB both the application as a type and the application instances, and moreover applications 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) we decided to represent the application model as an OWL Class. Another need is the possibility to aggregate different applications to build a complete business configuration (e.g., an ERP with a CRM) and also to model applications tenants and put application tenants in business configurations. The KB has to contain the SLAs associated with application or application tenants or with a whole business configuration. The SLA has been modeled as a set of Boolean expression The proposed KB provides REST services for storing and manipulating: DataCenter, Application Types, Business Configurations, Metric Types and Metric Values that are stored on an RDF Store (currently an OWLIM-SE instance). When the data is provided to be stored on the KB as RDF-XML, it is firstly validated and then stored. The KB also provides a SPARQL endpoint allowing making semantic queries.

A. *Smart Cloud Ontology*

The information stored in the KB as RDF triples uses a specific OWL ontology. The ontology developed allows modeling the different aspects of a cloud service as: infrastructure description (e.g., host machines, virtual machines, networks, network adapters), applications and services description, business configurations, metrics and SLA and also monitoring aspects.

1) Infrastructure: The infrastructure is modeled as a DataCenter with a set of HostMachines or HostMachineCluster (containing HostMachines). The HostMachines have specific attributes as the CPU core count, CPU architecture (e.g. x86), CPU speed, the RAM memory capacity, the network adapters, the LocalStorages available, the hypervisor used, etc. The DataCenter can also contain some ExternalStorage (e.g., NAS) that may be used to store virtual machines, and Routers and Firewalls that connect Networks. The HostMachines contain VirtualMachines that are used to run the services providing applications to users. VirtualMachines are characterized by virtual CPU count, RAM memory capacity, mass storage (different disks) and network adapters that connect to the network. The following is an example, written using the Turtle syntax, of a Virtual Machine with 1GB RAM, 2 CPUs, a 10GB disk that is stored on a disk of a host machine:

```
ex:vm1 rdf:type cld:VirtualMachine
    cld:hasName "vm 1, windows xp";
    cld:hasCPUCount "2";
    cld:hasMemorySize "1";
    cld:hasVirtualStorage ex:vm1_disk;
    cld:hasNetworkAdapter ex:vm1_net1;
    cld:hasOS cld:windowsXP_Prof;
    cld:isStoredOn ex:host1_disk
    cld:isPartOf ex:host1;

ex:vm1_disk rdf:type cld:VirtualStorage;
cld:hasDiskSize 10.
```

2) Applications and Services: Cloud Applications are realized using a variety of services (e.g., WebServers, Web Balancers, Application Servers, DBMS, application caches, mail servers, network file servers) that are available on machines over a network. These services may be deployed in many different ways, from all services on one machine to one machine for each service. There are some constraints to be fulfilled, for example some services are optional (e.g. application caches), some service need a specific feature (e.g. a web server supporting PHP). As already mentioned, to model Applications we have to represent both the application as a "type" (the class of Joomla applications) and applications as instances of a type. The following is the definition of the CloudApplication class written using Manchester notation:

```
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)
```

An application is defined as a piece of software with an identifier, a name, developed by some developer, whose instance was created by one creator and can be administered by administrators. Moreover it needs Services, other CloudApplications or CloudApplicationsModules, it can have at most one SLA and it uses some virtual machines. The subclass of applications representing the balanced Joomla applications are those that need one MySQL server, one http balancer, one NFS server for storing files and more than one Apache Web server supporting PHP 5, that is expressed as the following:

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

3) Metrics and SLA: For the definition and verification of Service Level Agreements are needed metrics values that need to be compared with reference values. Two kinds of metrics are defined: low level metrics whose values are provided directly from the Supervisor & Monitor sub-system (e.g. CPU%, memory used, network bandwidth used) that have point measure at a moment in time; and high level metrics that combine the values of low level metrics to provide a measure of a more general characteristics (e.g. the average CPU usage percentage over the last 30 min.) The ontology defines both low level metrics and high level metrics. The high level metric definition can combine the last value or the average, maximum, minimum, sum of values over a time interval (seconds, minutes, hours) of low level metrics using the basic mathematical operators (plus, subtract, multiply, divide). For example the ratio between the maximum memory used in the last 10 minutes and the average memory used in the last 30 minutes can be defined. Moreover SLAs can be defined and associated with applications or with business configurations. A SLA is defined as a set of ServiceLevelObjectives (SLO) that need to be verified within a certain validity interval. Each SLO is associated with a logical expression that needs to be verified, this expression is the AND/OR combination of checks of values (less than, greater than, equals) of high level metrics with reference values; the SLO is also associated with an action that needs to be done when the objective is not verified.

4) Business configurations: The business configurations contains the instances of the applications that need to work together to form a business process (e.g., an ERP application with a CRM application). Business configurations contain the application instances, the related service instances (application servers, DBMS, file storage, etc.) that are running on virtual machines. Moreover a business configuration can also contain simple virtual machines that are not used in an application and it can also contain host machines that are fully available for a specific customer.

5) Monitoring: When providing applications, services or host/virtual machines the information that should be used to enable monitoring them may be also specified, for example the monitoring IP address to be used, in case the machine has multiple IPs. Or the parameters to be used for monitoring a specific service metric.

B. Validation and Verification

The OWL ontologies has been designed for distributed knowledge representation and use the Open World Assumption (OWA) meaning that something that is not explicitly stated it is unknown if it is true or false. While for some domains the Closed World Assumption (something that cannot be proved true it is false) fits better. So for example if it is stated that an application needs at least two web servers and in the configuration is present only one, an OWL reasoner will not considered this a contradiction because in principle we do not know if a second web server exists. Another aspect is that OWL does not use the Unique Name Assumption, meaning that two different URI may identify the same thing. For example if an application must have exactly one DBMS service and in a configuration the application is associated with two DBMS identified with two different URI, a standard OWL reasoner will not identify an inconsistency, unless it is explicitly stated that the two URI identify different things. Considering these aspects, for the validation of the configurations submitted to the KB an OWL reasoner was not used but some SPARQL queries are used to check if a configuration is valid, similarly to [10] where some OWL axioms are transformed to SPARQL queries to express integrity constraints. So for example a query to list all the virtual machines in a configuration (each configuration is stored in a different graph) with not valid operative system is:

```
select ?vm ?os where {
            graph <> {
              ?vm a cld:VirtualMachine;
                  cld:hasOS ?os.
            }
            filter not exists {
              ?os a cld:OperativeSystem.
            }
      }
```

However in this case a problem arises when the range of the cld:hasOS property is declared an cld:OperativeSystem and this fact is stored in the RDF store (with reasoning enabled). In this case when the configuration with a wrong reference to the operative system is stored the rule based reasoner infers that this wrong OS identifier is an OS and the query will not identify the problem. For this reason the range declaration should not be present in the ontology. Using SPARQL queries also max/min cardinality can be checked. Since in the KB are also present the values of high level metrics (e.g., average CPU% in the last 30 min, used memory) the validation query may check if the host machine have space for the new virtual machine. A query may be also used to find the hosts where to allocate a virtual machine.

C. *Linked Open Graph Service*

The Linked Open Graph (LOG) is a tool that allows to visually navigating SPARQL endpoints and linked data services. LOG allows exploring the graph starting from a specific URI [?] and nodes can be progressively explored adding more details. The service allows to be embedded in other pages using html iframe. This service is used to explore the SPARQL endpoint of the KB. In Fig. 2 an example showing the structure of the DISIT datacenter is shown embedded in a page of the Smart Cloud Engine. This service is publicly available at http://log.disit.org.

IV. Smart Cloud Engine

In a typical cloud scenario a central engine for task scheduling is a major requirement. Common tasks such as virtual machine reconfiguration, move or cloning, memory or disk increasing, applying load balancing or fault tolerance best practices, require a scheduling mechanism that periodically checks the status of the system and, based on some reasoning, take consequent actions. The aim is to develop efficient algorithms for task scheduling in a cloud environment ([11], [12], [13]). The Smart Cloud Engine (SCE) is a core component of Icaro that periodically checks the status of the resources in the cloud infrastructure (e.g.,

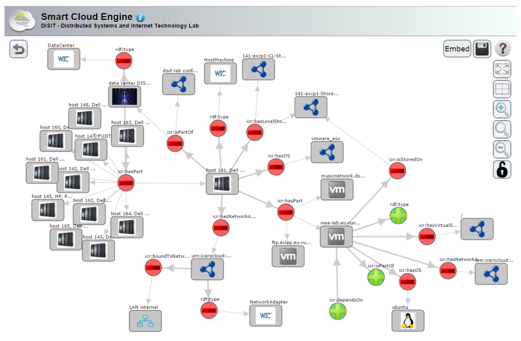


Figure 2. Linked Open Graph showing the DISIT datacenter

virtual machines and application services), connects to the Knowledge Base through the use of SPARQL queries, and invokes appropriate REST calls toward the Configuration Manager (CM), as defined in the Service Level Agreement related to the specific cloud service of interest (e.g., scaling, balancing, reconfiguration). SCE is an autonomous engine for the supervised exploitation of cloud resources, for the automation and optimization of services. SCE periodically checks for metrics and services status at IaaS, PaaS, SaaS levels of the cloud stack, and knows the current general configuration of the system and the status of the cloud; it knows the general and specific rules of production and scaling, and it works on event based logic that allows taking decisions, and dynamically balancing resources. 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). In the complex distributed cloud scenario of Icaro there is the need of distributed instances of running agents that perform cooperative tasks, and run in an integrated environment, with mechanisms for the load balancing of the various scheduled tasks. For this purpose, SCE features a multiplatform scheduling engine with cluster functionality that allows adding new scheduling nodes and defining jobs, for smart cloud management, without service downtime. 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 calls to the scheduler service (the same applies to triggers). Each schedulers node can run a job with a lock mechanism, thus modifying the status of the corresponding trigger.

A waiting trigger is waiting for its fire time, and to be acquired by a schedulers node; a paused trigger cannot put a job in execution; an acquired trigger has been selected by

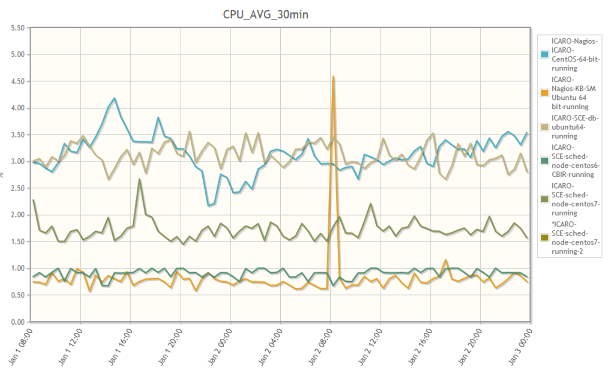
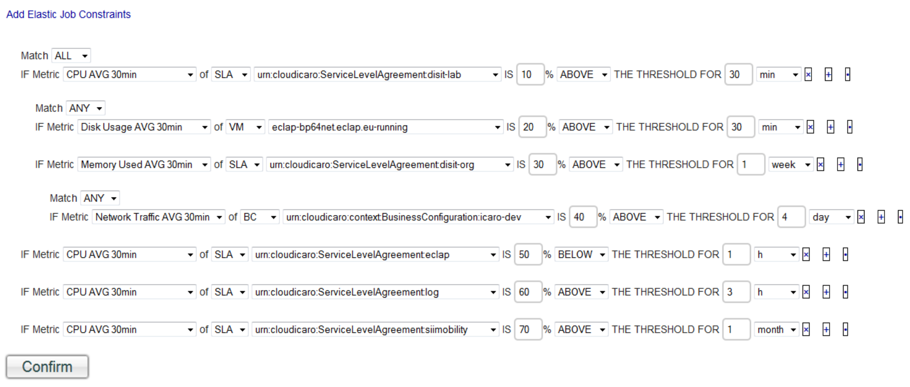


Figure 3. Metric graphs of a Service Level Agreement

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 schedulers 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. In a cloud environment, the rapid dynamic changes involving the parameters related to the various services require to define policies to maintain the coherency of data. At this regard, it is important to specify the concurrency level of the scheduled tasks. For this purpose, SCE supports both concurrent a non-concurrent schemes for jobs, where 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). SCE allows a direct monitoring of each job activity with a push interface, reporting the current status of the job, and the number of successes or failures in the last day or week, with relative percentages. Furthermore, the variety of the hardware at disposal and the jobs to be scheduled require best practices for adaptive job scheduling. For example, a reconfiguration process written for a particular CPU architecture should be bounded to run on a certain set of scheduler nodes only; nodes with high CPU load could reject the execution of further tasks, until their computation capacity is fully restored at acceptable levels; more in general, there could be the need to assign certain selected tasks only to nodes with a certain level of processing capacity. It is also of fundamental importance to be able to define mechanisms that allow scheduling tasks in a recursive way, based on the results obtained in previous tasks. For example, a reconfiguration strategy consisting of various steps could require taking different actions on the basis of dynamical parameters evaluated at runtime. At this regard, SCE allows adaptive job execution (e.g., based on the physical or the logical status of the host), and conditional job execution, supporting both system and REST calls. The user can build an arbitrary number of job conditions that must be satisfied in order to trigger a new job or a set of jobs, or can even specify multiple email recipients to be notified in case of a particular jobs result. By combining an arbitrary number of conditions, it is possible to define complex flow chart job execution schemes, for the management of different cloud scenarios. A trigger associated to a conditional job execution is created at runtime and it is deleted upon completion. It is possible to define physical or virtual constraints (e.g., CPU type, number of CPU cores, operating system name and version, system load average, committed virtual memory, total and free physical memory, free swap space, CPU load, IP address), that bind a job to a particular schedulers node. SCE supports both the execution of applications binary and Linux shell scripts, allowing specifying a custom number of parameters. Smart cloud best policies require services and tools to collect and analyze huge amount of data coming from different sources at periodic intervals. Virtual machines typically consist of hundreds of services and related metrics to be checked. SLAs often define bounds related to services or groups of services that consist of many applications, configurations, processing capacity or resources utilization. It is worth noting that collecting such a high number of data could lead to unmanageable systems, even if adopting the best practices of DMBS management or clustering, in a short period of time. For this purpose, SCE includes support for NoSQL, with the aim of allowing high performance in data retrieving and processing. SCE includes event reporting and logging services, for a direct monitoring of the smart cloud infrastructure and the activity status of every cluster node, and notifications about the critical status of a system or service (e.g., sending of emails). Notifications can be conditioned or not to the results of execution. SCE includes a status view of the currently running jobs (i.e., jobs status, previous and next fire time, job data map, jobs result or thrown exceptions, IP of the schedulers node that executed or is running the job), a history log of the schedulers activity, and a history log of the schedulers nodes status. Direct monitoring of the above view is also at disposal, and the reporting period of each schedulers 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



Add Elastic Job Constraints

```
Match ALL
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:disit-lab IS 10 % ABOVE THE THRESHOLD FOR 30 min
  Match ANY
  IF Metric Disk Usage AVG 30min of VM eclap-bp64net.eclap.eu-running IS 20 % ABOVE THE THRESHOLD FOR 30 min

  IF Metric Memory Used AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:disit-org IS 30 % ABOVE THE THRESHOLD FOR 1 week
    Match ANY
    IF Metric Network Traffic AVG 30min of BC urn:cloudicaro:context:BusinessConfiguration:icaro-dev IS 40 % ABOVE THE THRESHOLD FOR 4 day
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:eclap IS 50 % BELOW THE THRESHOLD FOR 1 h
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:log IS 60 % ABOVE THE THRESHOLD FOR 3 h
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:siimobility IS 70 % ABOVE THE THRESHOLD FOR 1 month
```

Figure 4. Elastic Metrics Editor

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 Figure XI). In order to check the coherence of a monitored service with respect to its SLA, SCE periodically forwards requests to the Knowledge Base (i.e., SPARQL queries), 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 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 is shown 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 cluster status, with detailed views of each schedulers 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). Concerning the best practices of elastic cloud computing, another requirement consists in allowing smart policies for resource scaling and reconfiguration, based on parameters not necessarily included in the Service Level Agreement. 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 data-centers) or to keep a certain quality level or in the optical of green cloud. At this regard, SCE integrates a module for the definition of elastic cloud policies, thus allowing defining custom Boolean expressions on metrics, virtual machines and business configurations (see Figure XII). Thus, it is possible to define complex periodic queries to be performed on the historical collected data, to verify the status of applications and services and to set custom alert limits, related or not to the correspondent SLAs. These periodic tasks can forward REST calls to a specified endpoint, to perform scaling and reconfiguration task on the system or service of interest.

V. EXPERIMENTS AND VALIDATION

The knowledge based driven solution described above was deployed on the Icaro cloud infrastructure for test and validation purposes. At this regard, 16 SLA and 17 business configurations were defined, with the aim of controlling the status of 66 virtual machines (consisting of 15 metrics) deployed on 12 physical host machines.

The most complex SLA has 75 conditions for an application (www.eclap.eu) 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 3 months (with about 3800 measures per metric), the time needed to evaluate a SPARQL query to evaluate the SLA and to get the current

Table I
METRIC ALARMS RECORDED ON ICARO (22/12/2014-19/02/2015)

Service Metric	**Alarms (relative approx. %**	**% (approx.)**	**Total Events**
Memory Used AVG 30 min	88348 (73.76%)	11.86%	119763
Disk Usage AVG 30 min	63825 (54.03%)	8.57%	118127
Network Traffic AVG 30 min	28723 (24.29%)	3.85%	118226
Last MySQL DB Size	15853 (59.55%)	2.12%	26620
CPU AVG 30 min	86 (0.07%)	0.01%	107887
Last Check Apache HTTP	66 (0.1%)	0.008%	35365
Last MySQL Connections	33 (0.1%)	0.004%	25506
Last Check Tomcat HTTP	24 (0.08%)	0.003%	29089
Last Apache # process	0 (0%)	0%	35207
Last MySQL # process	0 (0%)	0%	26472
Last Apache Tomcat # process	0 (0%)	0%	25006
Last Axcp Grid Node Check Alive	0 (0%)	0%	24043
Last Axcp Grid Node # process	0 (0%)	0%	23752
Last Axcp # process	0 (0%)	0%	15986
Last Axcp Check Alive	0 (0%)	0%	13644
Total	196958	-	744693

Table II
SCHEDULER'S NODE METRICS RECORDED ON ICARO (22/12/2014-19/02/2015)

Node Metric	Average Value
CPU load (JVM)	0.007%
System Load Average	0.5%
Committed Virtual Memory	5.25 GB
Free Swap Space	3.87 GB
Free Physical Memory	2.55 GB
CPU Load Average	0.37%
Free Disk Space	11.43 GB

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, 744693 different events were recorded, 196958 of them of alarm type (about 26.44%), as reported in Table V. The column alarm reports the number of alarms for each metric, with the relative percentage, with respect to the total number of metric events.

Data were recorded with an interval of 30 minutes. From the above figures, it is evident that the vast majority of alarms recorded were related to memory (11.86%), disk usage (8.57%), network traffic (3.85%), and database size (2.12%). The schedulers cluster consisted of two nodes and a shared MySQL database. The status of each node was recorded and a total number of 228630 events were collected (see Table V). For each reported alarm, a REST call was issued to a test service that recorded the data of the event.

VI. CONCLUSION

The conclusion goes here. this is more of the conclusion

ACKNOWLEDGMENT

The authors want to thank all the partners involved in Icaro, and the Regione Toscana for funding the project. Icaro has been funded in the POR CReO 2007 2013.

REFERENCES

[1] Xiong, Pengcheng, et al. "Intelligent management of virtualized resources for database systems in cloud environment." Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.

[2] D. Androcec, N. Vrcek, J. Seva, "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.

[3] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Towards a unified ontology of cloud computing. In Grid Computing Environments Workshop, 2008. GCE 08, pages 110, Nov 2008.

[4] 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), vol., no., pp.973,980, 18-21 Sept. 2011.

[5] 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)

[6] Antonio Cuomo, Giuseppe Di Modica, Salvatore Distefano, Antonio Puliafito, Massimiliano Rak, Orazio Tomarchio, Salvatore Venticinque, Umberto Villano, An SLA-based Broker for Cloud Infrastructures, Journal of Grid Computing, March 2013, Volume 11, Issue 1, pp 1-25, 20 Oct 2012.

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

[8] 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.

[9] Jonathan Stuart Ward, Adam Barker, "Observing the clouds: a survey and taxonomy of cloud monitoring", Journal of Cloud Computing 2014, 3:24 doi:10.1186/s13677-014-0024-2.

[10] Sirin, E. & Tao, J., 2009. 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. CEUR-WS.org.

[11] S. Sindhu, Saswati 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.

[12] 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 .

[13] 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 30453055