Future Generation Computer Systems 78 (2018) 142–154

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Managing cloud via Smart Cloud Engine and Knowledge Base

Pierfrancesco Bellini [a,b], Ivan Bruno [a,b], Daniele Cenni [a,b], Paolo Nesi [a,b,*]

[a] Distributed Systems and Internet Technology Lab, DISIT lab, University of Florence, Florence, Italy [1]

[b] Department of Information Engineering, University of Florence, Florence, Italy [2,3]

HIGHLIGHTS

- Knowledge based Smart cloud engine is proposed.
- Low and high level metrics for cloud service level agreement are computed.
- Flexible and scalable smart cloud engine.

ARTICLE INFO

Article history:
Received 31 March 2016
Received in revised form
10 September 2016
Accepted 6 October 2016
Available online 24 October 2016

Keywords:
Knwoledge base
Smart cloud
Cloud computing
Service level agreement

ABSTRACT

Complexity of cloud infrastructures needs models and tools for process management, configuration, scaling, elastic computing and cloud resource health control. This paper presents a Smart Cloud Engine and solution based on a Knowledge Base, KB, with the aim of modeling cloud resources, Service Level Agreements and their evolutions, and enabling the reasoning on structures by implementing strategies of efficient smart cloud management and intelligence. The solution proposed provides formal verification and intelligence tools for cloud control. It can be easily integrated with a large range of cloud configuration manager, cloud orchestrator, and monitoring tools, since the connections with these tools are performed by using REST calls and XML files. The proposed solution has been validated in the context of large ICARO Cloud project and in the cloud facility of a national cloud service provider. Some data resulting from the validation phases have been reported and are referring to the dynamic management of real ECLAP social network http://www.eclap.eu.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction and related work

Any relevant software infrastructure is presently deployed on cloud to manage resources in an efficient manner. The resource management is becoming a relevant hot topic for solutions that need to provide high availability and quality of service. Therefore, specific solutions for cloud monitoring, analyzing and dynamically changing configurations and services in the cloud are becoming mandatory to increase resilience and reliability. To this end, the modeling and formalization of cloud resources and information are becoming more relevant to manage different aspects of a cloud at its different levels (i.e., IaaS, PaaS, SaaS), and towards specific resources: hosts, virtual machines (VMs), networks, memory, storage, processes, services, applications, etc., and their relationships and technical tools. Cloud infrastructures are becoming every year more complex to be manually managed, especially for the presence of process configuration and reconfiguration tools, for the needs of dynamic scaling and elastic computing, and for monitoring resources health. The resources to be managed are obviously related to elements on cloud such as: hosts, VM, services, storages, processes, software applications, networks, etc., and thus on their corresponding composition and Service Level Agreements (SLA). The SLA can be regarded as the contract associated

* Corresponding author. Fax: +39-055-2758570.
E-mail addresses: pierfrancesco.bellini@unifi.it (P. Bellini), ivan.bruno@unifi.it (I. Bruno), daniele.cenni@unifi.it (D. Cenni), paolo.nesi@unifi.it (P. Nesi).

[1] http://www.disit.dinfo.unifi.it

[2] http://www.dinfo.unifi.it

[3] http://www.unifi.it

http://dx.doi.org/10.1016/j.future.2016.10.006
0167-739X/© 2016 Elsevier B.V. All rights reserved.

with a set of cloud resources and their configuration, contributing to a given service and fixing some parameters to service control. For example, a SLA signed by the customer with a Cloud Service Provider (CSP) describes its requests and requirements with respect to cloud services configuration and quality of service (e.g., 5 hosts, 4 CPUs at 5000 MHz minimum, 3 VMs in a certain configuration, minimum network speed of 50 Mbps, availability greater than the 99.99% of time). Thus, with the SLA the CSP declares the service level guaranteed and the constraints associated with the usage of cloud services. To this end, a number of metrics are defined and assessed for computing the business costs on cloud on "as a Service" basis, and are used to verify the SLA conformity. A review about SLAs models and types offered by commercial Cloud Providers can be obtained from Wu and Buyya, 2011 [1].

As a general consideration, the automation of cloud management may imply to cope with a number of activities also with respect to the SLA, such as: (i) formal verification and validation of cloud configurations in terms of resources, their relationships and matches (this action, can be performed before or after the resource's deployment; when it is performed before, it can be regarded as a sort of simulation with respect to the available resources); (ii) verification and reasoning about cloud security, taking into account networking and storage aspects; (iii) facilitating interoperability among public and private clouds, and/or among different cloud segments managed by different cloud orchestrators or managers in the same cloud infrastructure; (iv) discovering and brokering services and resources; (v) reasoning about cloud workload conditions, maybe via simulation; (vi) computing capability for horizontal and/or vertical scaling, thus elastic computing. In the literature, these aspects are addressed in several different manners. Some of them into SLA brokers such as in Cuomo et al., 2012 [2], and in Pengcheng et al., 2011 [3], while for the minimal monitoring you can see Ward and Barker, 2014 [4].

Therefore, reasoning tools about constraints and configurations on cloud are needed, and they have to model the infrastructure, the resources and the rules to manage all of them, with respect to the real data collected on the cloud via some monitoring solution. To this end, a data model representing the cloud complexity is mandatory.

In a seminal work of Youseff et al., 2008 [5], an approach to create a cloud ontology has been proposed, decomposing cloud modeling problems into five layers: applications, software environments, software infrastructure, software kernel, and hardware. Moreover, Zhang et al., 2012 [6] proposed a solution to make easier the searching services and resources into the cloud by presenting the CoCoOn ontology, also integrating other QoSOnt ontology (Dobson et al., 2005 [7]), for the description of service discovering and parameters. For the description at level of IaaS, the INDL (Infrastructure and Network Description Language) ontology defines nodes connected via links and interfaces, as described in Ghijsen et al., 2012 [8]. Virtual Nodes are used to model VMs in execution on former nodes. Node Components are used to represent resources as memory, storage, CPU, while many details are missing as the real network addresses, and the information describing physical aspects of VMs and Hosts. In mOSAIC EC project (Moscato et al., 2011 [9]), 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 terminologies. The mOSAIC ontology allows describing aspects of hosts and VMs, while presenting some lacks on modeling connections. For modeling the main cloud entities some generic ontologies could be used. For example, https://geni-orca.renci.org/owl/owl-test/compute.rdf# ontology has been defined into ORCA project https://geni-orca.renci.org, which is mainly focused on modeling the hierarchical aspects of networking, while it presents limited capabilities in modeling cloud entities and applications at all cloud levels of NIST.

For the description of general cloud services, Linked-USDL (Unified Service Description Language) in Pedrinaci et al., 2014 [10] provides a set of ontological models for describing services, SLA, security, prices and intellectual property. Linked USDL is a remodeling of USDL language and reused other RDF(S) vocabularies such as: GoodRelations (http://www.heppnetz.de/projects/goodrelations/), Minimal Service Model (http://iserve.kmi.open.ac.uk/wiki/index.php/IServe_vocabulary), and FOAF (http://www.foaf-project.org/). At application level of the cloud, the standard OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) of Binz et al., 2012 [11], and in Binz et al., 2014 [12], allows to describe via XML the application components, their dependencies, and the plan (workflow) for provisioning on the infrastructure; thus formalizing the work of the Orchestrators. In the work of Bernstein et al., 2010 [13], 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 (https://standards.ieee.org/develop/project/2302.html) 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 an ontology on cloud computing, Androcec et al., 2012 [14].

As a more lightly related work, the modeling of services via OWL-S has been proposed for describing web services with their service profile and the WSDL formalism. For the SLA of Web Services, WSLA has been proposed via an XML schema, and allows composing metrics on specific services, see Ludwig et al. 2003 [15]. In this context, WS-Agreement has been developed by "*Grid Resource Allocation Agreement Protocol Working Group*" (GRAAP-WG) with the aim of describing SLA among distributed entities in the grid (Andrieux et al. 2007 [16]). On the basis of the WS-Agreement, in Oldham et al., 2006 [17], an ontology has been defined. All these results have not been open to the public and are strongly focused on web services and related to the quality of service.

The work presented in this paper describes a Smart Cloud Engine (SCE) and solution based on modeling cloud resources and information via a Knowledge Base (KB). The proposed KB and tools also cope with SLAs, detailed cloud resource descriptions, and monitors information associated with resources; thus allowing the monitoring of the SLA evolution, managing complex configurations, according to the SLA and related strategies for dynamic scaling and elastic computing. In this context, the SLA is modeled and addressed as the service agreement between the CSP and the cloud customer and not at level of web service, or network service. The adoption of a KB to model the cloud knowledge grounded on a cloud ontology and data instances enables the reasoning on cloud structures and their evolutions. And thus, it is suitable for implementing strategies of smart cloud management and intelligence. Moreover, the proposed Smart Cloud Engine and solution can be exploited in connection with a large rangeof cloud management tools such as configurators, orchestrators, and monitoring tools. The solution proposed in this paper has been developed in the context of the ICARO Cloud project. It has been validated with respect to the cloud infrastructure of Computer Gross, a CSP providing cloud services at different levels (i.e., IaaS, PaaS and SaaS), in which allocated applications at SaaS level are provided by several different vendors. The tools allocated on the cloud of Computer Gross belong to categories of multitier solutions for CRM (Customer Relationship Management), ERP (Enterprise Resource Planner), workfiow, marketing, business intelligence, cultural heritage, social media, etc. A large variety of solutions on cloud increases the complexity of cloud management, that is configuration and dynamic management. Moreover, these problems motivate the needs of a fiexible smart cloud engine as that presented in this paper. Therefore, the validation phase of the proposed Smart Cloud Engine has been mainly focused on assessing

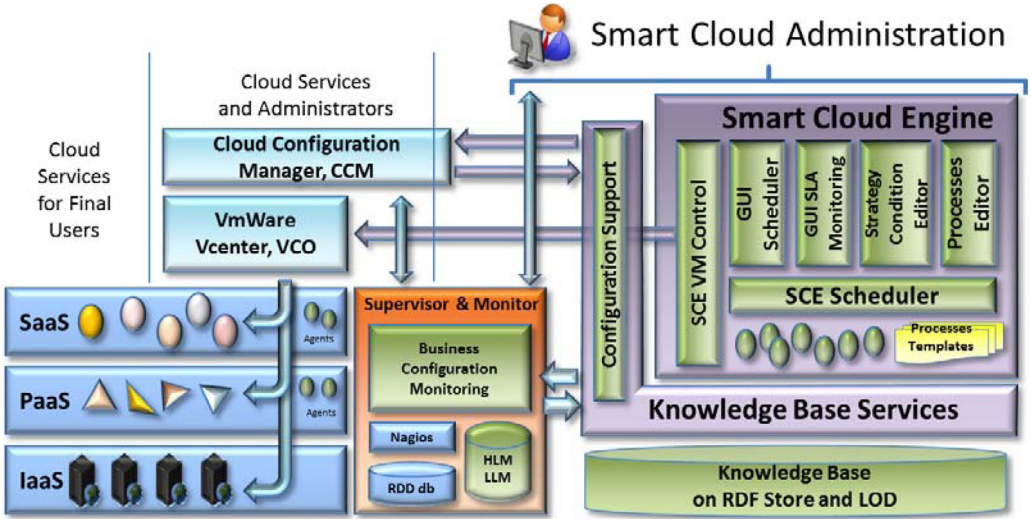

Fig. 1. ICARO Smart Cloud architecture with connection with VMware, while all the interaction are generally defined in terms of well-defined API and JSON/XML files.

its effectiveness with respect to the automation of scaling/balancing, reconfiguration, and continuous verification of SLA and resource health.

This paper is structured as follows. In Section 2, the ICARO Smart Cloud architecture is presented in relationships with the typical elements on the different layers of cloud infrastructures. Section 3 describes the ICARO cloud ontology for modeling the cloud at the basis of the Knowledge Base, adopted for smart cloud reasoning on configurations, status conditions, scaling, and SLA. In Section 4, the structure and solution for the Smart Cloud Engine are presented. Section 5 depicts some experimental results to show the effectiveness of the proposed solution, and the scalability aspects. The validation has been performed on a real context and the data presented are related to the dynamic management of ECLAP social network. Conclusions are given in Section 6.

2. The ICARO Smart Cloud architecture

The proposed Smart Cloud Engine addresses some of the critical issues put in evidence in the introduction, coping with complex business configurations deployed on the cloud and coming from multiple software providers. In most of the cloud management systems, a set of ready-to-use configurations is offered to the cloud buyers and produced by a Cloud Configuration Manager (CCM). The business configurations, as well as changes of configurations, are typically deployed on the cloud by using an Orchestrator (for example VCO by VMware, or that by Microsoft, 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 for some smart cloud features, such as in the IBM solution. Most of the commercial Smart Cloud solutions have limited capabilities in formally defining and detecting complex conditions about consumption and status of resources, which can be used for activating changes in the cloud, and firing scaling up or down rules (vertical and horizontal, elastic, etc.).

The proposed Smart Cloud architecture is reported in Fig. 1, which includes the Smart Cloud Engine (SCE), with its frontend. The SCE can be invoked and exploited by any CCM or Orchestrator, for: (i) registering new business configurations and their corresponding SLA; (ii) requesting verification and validation of specific business configurations, and receiving back suggestions and hints related to feasibility, consistency and completeness; (iii) activating sporadic and/or periodic monitoring of the SLA associated with a contract; (iv) activating the sporadic and/or periodic control of the health of a business configuration and/or of any specific resource and service on the cloud at any level; (v) enabling the assessment of firing conditions for activating reconfiguration strategies associated with business configurations; for example, for dynamic scaling, VM cloning, VM moving, activating reconfiguration rules To perform these activities the SCE exploits a: (a) Knowledge Base (KB), in which business configurations, cloud models, SLA and data are registered; (b) a distributed scheduler (Distributed SCE Scheduler, DISCES) to periodically and sporadically put in execution processes capable to assess conditions, compute possible solutions, and provide suggestions to CCM and Orchestrators; (c) one or many monitoring tools (called in the figure Supervisor and Monitor, SM) for collecting data from the cloud in massive manner according to different layers (i.e.,: IaaS, PaaS, SaaS), and business configurations.

The KB automatically programs the SM (Supervisor and Monitor) services and tools to set up and activate the monitoring processes for controlling services and resources. To this end, SM uses drivers to manage multiple Nagios instances (not discussed in this article). In this manner, the SCE automatically adds all the monitoring processes and structures that permit at the SCE to have the needed information for controlling the business configurations behavior related to SLA, for scaling and reshaping strategies, and of all the resources involved. The SLAs are typically based on High Level Metrics (HLM) as applicative metrics, such as for example: the number of times the workload exceeded a given threshold in the last 20 min, the average number of registered users per day, the number of contemporary streaming processes, etc. Once the SM monitoring processes are activated, the useful measured data are received and collected into the KB. Other detailed data may be left cumulating as historical data into the monitoring tools and services. In order to fastening the alignment of the Smart Cloud solution with an eventual VMware based infrastructure already in place, the SCE also presents a process to directly access the vCenter data of a vSphere solution.

The SCE interface allows (for each business configuration and cloud resource) to easily: (i) activate strategies (reconfiguration, scaling, cloning, migration, etc.), (ii) verify the health of cloud resource, and of SLA of business processes allocated on cloud, and (iii) control the whole cloud conditions with high level parameters. To this end, the SCE presents a suitable user interface called SCE Management Interface and Tools, which includes: (i) editors for programming the Smart Cloud, for the definition and management of SLA monitoring, assessment, and strategies setup, and (ii) graphics rendering tools for depicting the trends of metrics and SLA parameters with respect to firing conditions. These activities are performed without requesting to program the processes that are activated by the SCE on the DISCES. The processes of DISCES can be sporadic and/or periodic and exploit the KB by performing semantic queries in SPARQL. Thousands and thousands of processes are executed per day by the DISCES, that put them 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 smart cloud solution. If needed, the processes of DISCES can be manually formalized. To this end, for the formalization of semantic queries, a suitable graphical user interface based on Linked Open Graph, LOG (http://log.disit.org) can be used to access the KB and browse the semantic model and data, as described in Bellini, Nesi and Venturi, 2014 [18].

Please note that the proposed Smart Cloud solution can be easily integrated with any CCM, and/or Cloud Orchestrators, and by exploiting some monitoring tool since the connection with these subsystems are performed by using REST calls and XML files. In the validation case, the solution was directly managed by higher-level CCM addressing different kinds of orchestrators. The fact that the KB accepts in input XML files allows to almost all CCMs and Orchestrators to communicate the information they generate by a suitable XML for the KB. In the cases in which the XML data is already produced by those tools, it can be adapted to the KB XML compliant by using an XSLT (https://www.w3.org/TR/xslt) transformation. Moreover, the ICARO SCE and KB are capable to also directly read (get) the configurations from VMware vCenter for shortening the migration to the Smart Cloud solution.

The SCE also provides a SCE VM Control that sends to the VMware vCenter VCO commands for Virtual Machines, and in particular for cloning, starting, stopping, etc. The above described approach based on XSLT can be adopted for data and commands produced by the SCE VM Control. In fact, A XSLT has been already adopted in the connection KB-to-SM to transform the cloud monitoring configuration coming from KB to set up the monitoring configuration in the SM (which can be of different kinds).

3. The Smart Cloud Engine and knowledge base

The KB collects and organizes the configurations of the whole cloud services, ranging from the infrastructure elements to the applications, and also addressing the applicative metric definitions and values. A review on cloud ontologies and KB usage in the context of cloud can be recovered in Bellini, Cenni, and Nesi, 2015 [19]. The use of a KB: facilitates the interoperability among public and private clouds, and/or among different cloud segments; allows formal verification and validation of resource related cloud configuration, discovering and brokering services and resources, reasoning about cloud security, computing capability for horizontal or vertical scaling; and thus implementing strategies for elastic computing. To this end, the KB needs to store not only the cloud structures (infrastructure, applications, configurations, SLA) and also the values of metrics in time to be able to answer questions such as "*Which host machines can allocate a new VM with certain features?*" or "*Which are the overloaded hosts?*", "*Which business configuration has violated SLA parameters?*". The storage of the full history of all metric values on the KB can be too expensive and unnecessary. Therefore, in most cases, only metrics related to the SLA and HLMs are stored with respect to Low-Level Metrics (LLM) values (CPU percentage, memory used, disk available, database size, etc.), which can be in any case accessible from SM. High-level metrics can be used to represent the resource workload less affected by small sporadic changes (e.g. average or maximum percentage of CPU usage over last hour, percentage of time over the threshold). On the other hand, values of LLMs are stored in the monitoring service every 5 s, consuming more space in the storage. Regarding the application description in the KB both the application model/type and instances need to be stored. Applications may have specific constraints, like the number of services involved (e.g., number of front–end web servers). In order to avoid duplicating the type/instance relation (already modeled in RDF/OWL) and to leverage the modeling capabilities of OWL2 to express constraints (e.g., max/min cardinality), the application model is represented as an OWL Class. Another need is the possibility to aggregate different applications, servers, VMs to build a complete business configuration (e.g., an ERP with a CRM) and also to model applications tenants and to be able to put application tenants in business configurations. The KB also keeps the SLAs associated with business configurations and applications. The SLA is modeled as a set of Boolean expressions that relate metric 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 web server processes running on a VM has to be greater than 0). The adoption of the KB has been demonstrated to be a valid support also for cloud simulation, as in Badii et al. [20].

3.1. Smart cloud ontology of ICARO

The cloud description ontologies available in literature and mentioned in the introduction (for example, Androcec et al., 2012 [14]), are focused on the description of cloud services mainly aiming at cloud services discovery or searching/matching cloud offers with respect cloud demand. Moreover, the cloud model adopted on SCE in the KB has to be suitable for controlling configuration and monitoring cloud evolution. For this reason a cloud ontology has been developed allowing the representation of the different aspects of cloud services at all levels (IaaS, PaaS and SaaS) as: infrastructure description (e.g., host machines, virtual machines, networks, network adapters), platforms, applications and services description, business configurations, metrics and SLAs at cloud level and related monitoring aspects. In Fig. 2 the structure of the ontology with major classes and relationships is reported. The whole ontology is accessible at http://www.disit.org/cloud_ontology/core (as Linked Open Vocabulary) and other details are available at http://www.disit.org/6715. The proposed cloud ontology is grounded on vocabularies as standard OWL, RDFS, and exploits Dublin Core and FOAF for general ontology information. The proposed ontology can be easily combined with other ontologies for Web Services and their related SLA, while it includes the formalization of a cloud SLA with conditions and references for low and high level metrics directly computable by the SCE. For this reason, it was not possible to exploit state of the art SLA models.

The main aspects of ICARO cloud ontology are described as follows.

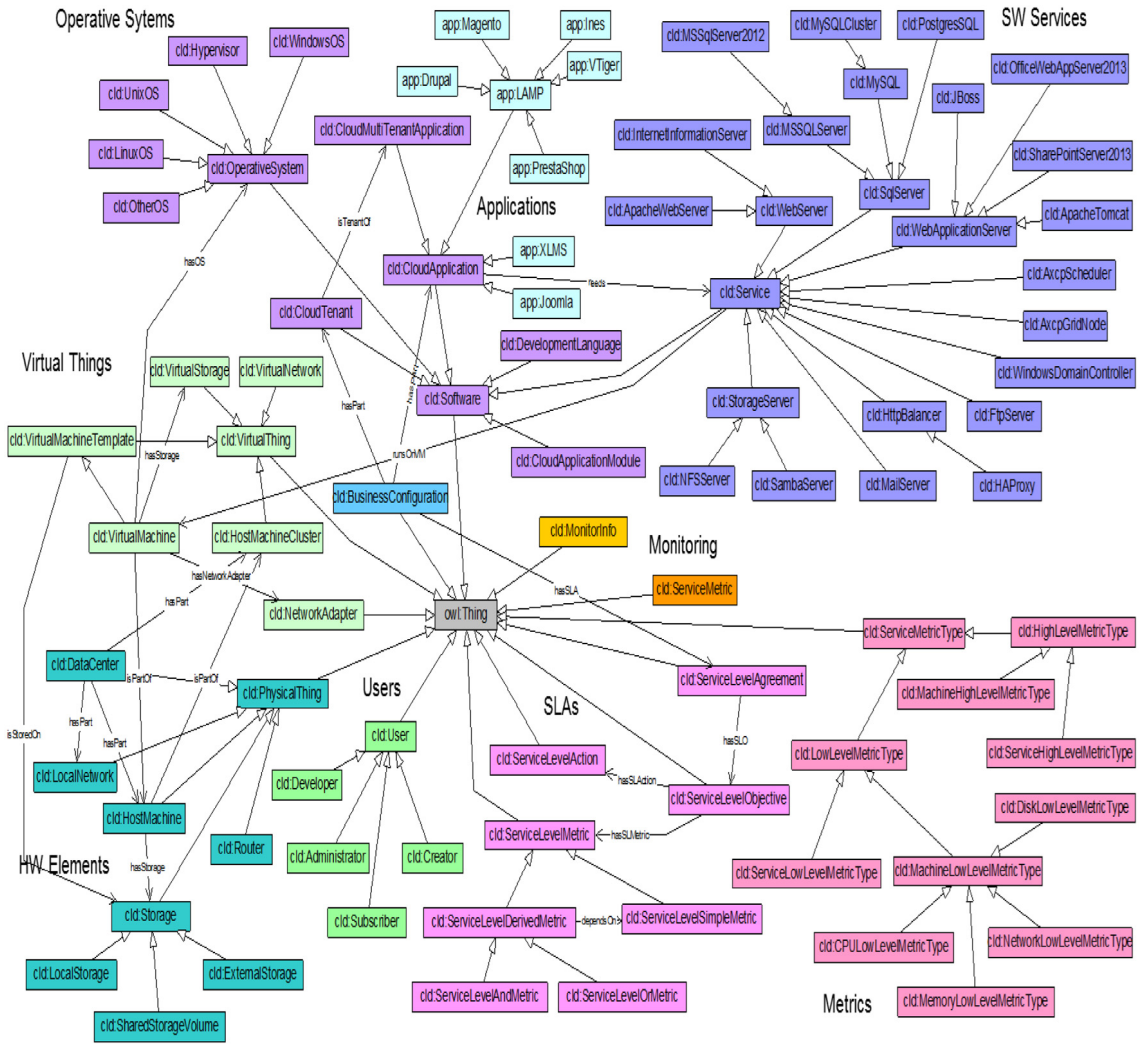

Fig. 2. Cloud ontology showing main relationships, the full model can be accessed from the formalization document (see link in the text) or browsed from the LOG.disit.org.

Infrastructure: the infrastructure is modeled as a *DataCenter* with a set of *HostMachines* or *HostMachineCluster* (containing HostMachines) and *LocalNetworks*. The *HostMachines* have specific attributes, such as CPU core count, CPU architecture (e.g., x86), CPU speed, RAM capacity, network adapters, *LocalStorages*, used hypervisor, etc. The *DataCenter* can also contain *ExternalStorages* that may be used to store virtual machines, *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.

Applications and services: cloud applications are realized using a variety of services (e.g., web servers, HTTP 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 a single machine to one machine for each service. There are some constraints to be fulfilled, for example some services are optional (e.g., application caches), while other services need specific features (e.g., a web server supporting PHP). As already mentioned, to model applications we have to represent both the application as a "type" (e.g., the class of Drupal applications) and applications as instances of a type. The following is the definition of the *CloudApplication* class written by using the 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)

A cloud application is defined as a 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*, and can have at most one *SLA* and it uses some *VirtualMachines*. The applications representing the balanced Crossmedia Learning Management System (XLMS) applications (the application used for the experiments reported in the following as the major engine of ECLAP social network) are those that need at least two Apache web servers supporting: PHP, some Balancer, 2 MySQL server, 2 Tomcat application servers one AXCP Scheduler with one or more AXCP Grid Nodes for media processing, FTP server, etc. All these facts are expressed as the following:

```
XLMSBalanced subClassOf CloudApplication
    and (developedBy value disit)
    and (needs min 2 (ApacheWebServer and (supportsLanguage value php)))
    and (needs exactly 3 Balancer)
    and (needs exactly 2 MySQL)
    and (needs exactly 2 ApacheTomcat)
    and (needs max 2 StorageServer)
    and (needs exactly 1 AxcpScheduler)
    and (needs some AxcpGridNode)
    and (needs exactly 1 FTPserver)
    and (needs only (XlmsModule or ApacheTomcat  or ApacheWebServer
      or AxcpGridNode  or AxcpScheduler or HttpBalancer or MySQL or StorageServer))
```

Moreover, it is possible to define multi-tenant applications, where more tenants can be associated with the application instance, each tenant can be bought separately and can have a specific SLA.

Metrics and SLA: for the definition and verification of SLAs, metrics are compared with reference values. To this end, two kinds of metrics are defined: LLMs (e.g., CPU percentage, memory used, network bandwidth used); and HLMs that combine the values of LLMs to provide a measure of a more general characteristics (e.g. the average CPU usage percentage over the last 30 min). The ICARO smart cloud ontology defines both HLMs and LLMs. The high level metric, HLM, 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 min and the average memory used in the last 30 min can be defined as a HLM.

For the ECLAP test case, we used as a HLM the "*average CPU usage percentage over last 30 min*", that is defined using RDFXML as:

```
<cld:MachineHighLevelMetricType
        rdf:about="urn:cloudicaro:MachineHighLevelMetricType:CPU_AVG_30min">
    <cld:hasMetricName>CPU AVG 30min</cld:hasMetricName>
    <cld:hasDescription>…</cld:hasDescription>
    <cld:hasMetricUnit>%</cld:hasMetricUnit>
    <cld:forGroup>%URN%</cld:forGroup>
    <cld:hasExpression>
        <cld:MetricMeasure>
        <cld:onLowLevelMetricType
            rdf:resource="urn:cloudicaro:CPULowLevelMetricType:CPU_AVG"/>
        <cld:useOperator>avg</cld:useOperator>
        <cld:useMultiValueOp>avg</cld:useMultiValueOp>
        <cld:useTimeInterval rdf:parseType="Resource">
            <cld:hasValue rdf:datatype="&xsd;decimal">30</cld:hasValue>
            <cld:hasUnit>min</cld:hasUnit>
        </cld:useTimeInterval>
    </cld:MetricMeasure>
  </cld:hasExpression>
</cld:MachineHighLevelMetricType>
```

A very similar HLM has been used by making the average over last 10 min.

The LLM (used in the above presented HLM) reports the average CPU percentage extracted from the SM, using the SNMP protocol, and it is defined as:

```
<cld:MachineLowLevelMetricType rdf:about="urn:cloudicaro:CPULowLevelMetricType:CPU_AVG">
    <rdf:type rdf:resource="&icr;CPULowLevelMetricType"/>
    <cld:hasMetricName>CPU AVG</cld:hasMetricName>
    <cld:hasDescription>average CPU</cld:hasDescription>
    <cld:hasMetricUnit>%</cld:hasMetricUnit>
</cld:MachineLowLevelMetricType>
```

SLAs can be defined and associated with applications, application tenants 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, equal) of high-level metrics with reference values; the SLO is also associated with an action that needs to be performed/started when the objective is not verified. For example, the SLA used to check that the average CPU used over last 10 min is less than the 70% for the front–end machine has the following formulation:

```
<cld:ServiceLevelAgreement rdf:about="urn:cloudicaro:ServiceLevelAgreement:eclap">
  <cld:hasSLObjective>
    <cld:ServiceLevelObjective>
      <cld:hasSLAction>
        <cld:ServiceLevelAction>
          <cld:hasName>activate ECLAP-FE-2</cld:hasName>
          <cld:callUrl>http://...:8080/SmartCloudEngine/?...activate...</cld:callUrl>
        </cld:ServiceLevelAction>
      </cld:hasSLAction>
      <cld:hasSLMetric>
        <cld:ServiceLevelAndMetric>
          <cld:dependsOn>
            <cld:ServiceLevelSimpleMetric>
              <cld:hasMetricName>CPU AVG 10min</cld:hasMetricName>
              <cld:hasMetricValueLessThan rdf:datatype="&xsd;decimal">
                70
              </cld:hasMetricValueLessThan>
              <cld:hasMetricUnit>%</cld:hasMetricUnit>
              <cld:dependsOn rdf:resource="urn:cloudicaro:VirtualMachine:vm-118"/>
            </cld:ServiceLevelSimpleMetric>
          </cld:dependsOn>
        </cld:ServiceLevelAndMetric>
      </cld:hasSLMetric>
    </cld:ServiceLevelObjective>
  </cld:hasSLObjective>
  <cld:hasStartTime rdf:datatype="&xsd;dateTime">2014-11-10T00:00:00</cld:hasStartTime>
  <cld:hasEndTime rdf:datatype="&xsd;dateTime">2018-11-10T00:00:00</cld:hasEndTime>
</cld:ServiceLevelAgreement>
```

When CPU percentage is greater than 70%, the URL in the *callUrl* property is called to start and activate additional Front End VM. see Section 5 for a more detailed description.

Business configurations: the business configurations contain the instances of the applications that need to work together to form a business process (e.g., multitier solution with scalable frontend/backend). 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; host machines that are fully available for a specific customer; as well as an application tenant with a specific SLA.

Monitoring: when providing applications, services or host/virtual machines the information that should be used to enable their monitoring 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.

3.2. Validation and verification via the KB

The KB allows storing and manipulating DataCenters, Application Types, Business Configurations, Metric Types and Metric Values using specific REST services, and SLA. The KB is physically stored in an RDF Store (an OWLIM-SE instance). The KB provides a SPARQL endpoint to perform semantic queries. The data is provided via REST services as RDF-XML files and it is validated to check if it is consistent with the ontology.

As a general consideration, the OWL model for ontologies is designed for distributed knowledge representation and uses the Open World Assumption (OWA), meaning that something that is not explicitly stated it is unknown if it is true or false. For example, if it is stated that an application needs at least two web servers, while in the configuration it presents only one, then, a simple reasoner will not consider this a contradiction, because in principle we do not know if a second web server exists, and neither that it does not exists now. This problem can be solved by adding specific information, completing the knowledge base. Another aspect is that OWL does not use the Unique Name Assumption, meaning that two different URIs 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 URIs, a standard OWL reasoner will not identify an inconsistency, unless it is explicitly stated that the two URIs identify different things. For these reasons, for the validation of the configurations submitted to the KB, SPARQL queries have been directly used to check if a configuration is valid, similarly to Sirin and Tao, 2009 [21] where some OWL axioms are transformed to SPARQL queries to express integrity constraints. Therefore, for example, a query to list all the VMs in a configuration (each configuration is stored in a different graph) with a non-valid operative system is:

```
SELECT ?vm ?os WHERE {
  GRAPH <...> { ?vm a cld:VirtualMachine; cld:hasOS ?os. }
  FILTER NOT EXISTS { ?os a cld:OperativeSystem. }
}
```

However, a problem arises when the range of the cld:hasOS property is declared as cld:OperativeSystem and this fact is stored in a reasoning-enabled RDF store. In this case, when the configuration with a wrong reference to the operating 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. Using SPARQL queries to validate the configurations allows checking also aspects that the OWL2 language cannot express. For example a SPARQL query may check if the host machine has space for the new virtual machine. Moreover a SPARQL query may be used to find a host where a new virtual machine can be allocated. Since the SPARQL queries are stored in a configuration this provides a high fiexibility, and the query can be changed when the ontology is changed to address new needs (e.g., support for Linux Containers); this can be done without changing the application code.

4. Smart Cloud Engine, the SCE

A major requirement in a Smart Cloud environment consists of a scalable engine for monitoring and making decisions. In this context, the goal is to develop efficient scheduling solutions for Smart Cloud management and scheduling, see for example Sindhu and Mukherjee, (2011 [22]); Ma et al. (2013 [23]); Tsaia et al. (2013 [24]). To this end, a distributed scheduler for putting in execution monitoring and making decision processes has been developed. The DISCES (Distributed SCE Scheduler) is a core component of the ICARO infrastructure that periodically checks the status of the resources in the cloud infrastructure (e.g., virtual machines and application services). DISCES consists of a set of distributed instances of running agents performing concurrent tasks on a set of virtual machines also allocated on cloud. DISCES is connected to the KB on which performs SPARQL queries on the basis of the SLA of the specific cloud service and may invoke REST calls toward the CM to get specific monitoring data when needed. SCE includes DISCES multiplatform scheduling engine with cluster functionality that allows adding distributed nodes and defining jobs, without service downtime. Each scheduling job includes a name and a related 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 some associated triggers with their metadata (i.e., name and group, period and priority of execution).

SCE can be used for VM scaling and reconfiguration (VM moving, cloning or migration, memory or disk increasing, enabling load balancing or fault tolerance). Based on cloud reasoning policies, the SCE takes consequent actions upon detecting critical issues in the cloud infrastructure, both at physical or application level. SCE can check both metric and service status at all levels of the cloud stack. It is aware of the current system and cloud configuration and applies general or specific rules for production and scaling, working with an event based logic that allows taking decisions, for dynamic balancing of resources.

SCE allows defining rules for (i) verifying the cloud status at different level of resources and firing recovering tools and/or alarms actions, (ii) defining firing conditions for activating policies, for example for the management of alarms and events, to optimize the resource utilization across various datacenters (e.g., increasing of computational capacity, automatic data migration). A trigger associated with 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 scheduler's node. Smart cloud best policies require services and tools to collect and analyze huge amount of data coming from different sources at periodic intervals.

SCE allows monitoring the status of the cloud platform (i.e., hosts, virtual machines, applications, metrics, alerts and network interfaces), with details about the compliance of metrics with respect to the SLA. SCE provides graphs of all the relevant metrics in order to perform a deep data analysis. SCE allows to define policies to apply in case of misfired events (e.g., reschedule a job with existing or remaining job count), and allows to produce detailed graphs for every metric (grouped per VM or not), with customizable time intervals. SCE reports for each metric the total amount of times it was found to be out of scale, with respect to the total number of performed checks. Logged metrics report the list of SLA violations occurred in the selected time period, with relevant data (e.g., time at which violation occurred, metric name, registered value, threshold, and the related business configuration, virtual machine and SLA).

The user is able to define custom elastic policies with the aim of a Strategy Condition Editor. In a typical scenario, it is necessary to monitor several hardware parameters (at physical or virtual level), for example the averaged amount of RAM and CPU utilization of a host over a defined period of time (e.g., in the last hour or minutes). The scheduler of the SCE periodically checks the average value of the requested parameters in the chosen time interval, with respect to what is defined in the SLA. Each check contributes as a Boolean value to the whole Boolean expression that is evaluated. The global Boolean expression is evaluated upon completion of all the parameters' checks. If the overall check returns an alarm (true) then a REST call may be performed to invoke an URL defined in the SLA (alternatively, in the SCE rule editor, it is possible to define custom calls with parameters). In this manner, it is possible to design and perform consistency checks at various levels (e.g., at hardware and/or application level) to maintain the system within the requested boundaries and constraints. The available metrics include, among the others, CPU, disk, memory, network related parameters, which can be related to different configurations (i.e., SLA, Virtual Machine, Business Configuration). In this sense, the proposed solution proves to be efficient for the smart cloud monitoring and optimization of services at various levels, since it can deal with different aspects of a service platform, in heterogeneous environments and infrastructures.

SCE rules can communicate decisions to the Orchestrator via the SCE VM Control, which is directly connected to the vCenter via the VMware APIs. They include functionalities for the VM on each ESXi hypervisor and in particular to: take snapshots, perform migration, put in execution (start), stop, change configuration, manage fault tolerant groups, rebooting, suspending, resetting, removing snapshots, reverting a virtual machine to the current snapshot, etc. The SCE VM Control has a modular architecture with plugin support which allows defining further communications layers with third party cloud entities. In particular, according to Fig. 1, the SCE calls the SCE VM Control with a REST call, and the call implements the invocation to the Orchestrator. Presently two different versions SCE VM Control have been realized for adapting the SCE call: (a) to connect the SCE with the VMware VCO, and (b) to connect the SCE with a third tool produced by ComputerGross (partner of ICARO) that allows managing a range of Orchestrators. Thus, it is possible to translate in a transparent manner the SCE VM Control APIs to those related to the target cloud of interest or to other cloud APIs (e.g., LibCloud [25], jClouds [26]).

4.1. Computing high level metrics

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) [4].

Low Level Metrics (LLM) are focused to the 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. LLM are collected by Nagios (https://www.nagios.org/), and stored in a Round Robin Database provided by the PNP4Nagios tool. High Level Metrics (HLM), combine values of LLMs to provide more concise and abstract measures of resources related trends and status. 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 (+, −, *, /). The evaluation of HLMs is performed by a module associated with each configuration when it is deployed. The plugin asks to the KB for

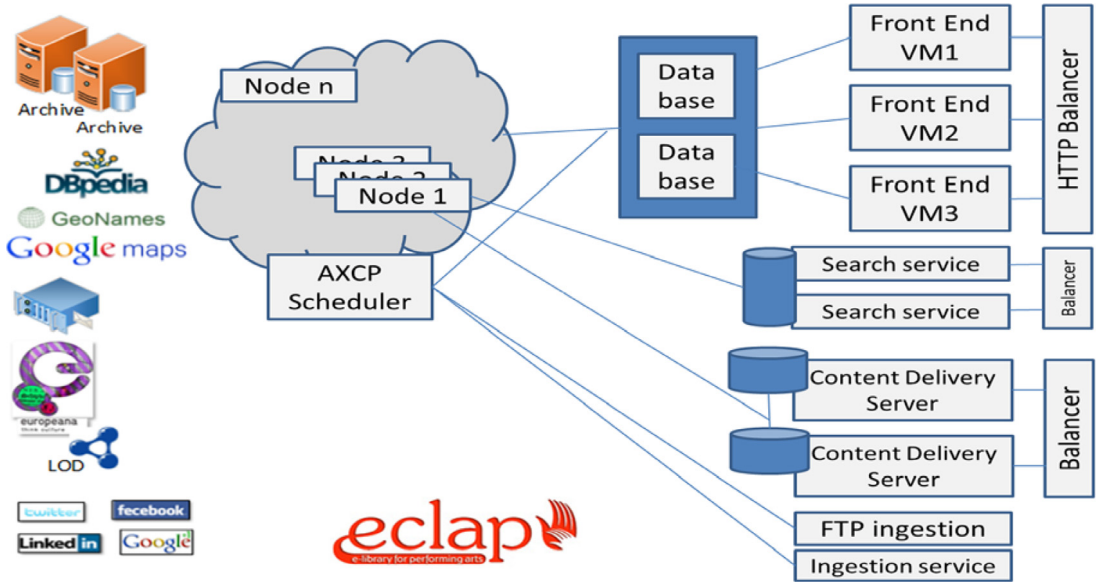

Fig. 3. ECLAP social network distributed architecture. Each VM is represented by a square box.

the HLM definition associated with the configuration, and computes them according to their hierarchical synthetic definition. LLMs and HLMs can be used for defining logical rules into SLAs.

The HLMs values can be accessed using a SPARQL query via the standard REST interface on the KB as in the following example:

http://kb.mycloud.com/KB/sparql?query=SELECT ...

where: the query to retrieve the last value of metric "*CPU AVG* 30 min" of VM identified by <*urn:mycloud:VM2*> is the SPARQL encoding as follows:

```
SELECT ?value ?t WHERE {
        ?s a cld:MetricValue;
          cld:hasMetricName "CPU AVG 30min".
          cld:dependsOn  <urn:mycloud:VM2>;
          cld:hasMetricValue ?value;
          cld:atTime ?t.
} ORDER BY DESC(?t) LIMIT 1
```

On the other hands, LLM values can be accessed calling the SM via a REST API to obtain graphs or single values of metrics. For example, to access at the last value of a LLM the URL to call has the following structure:

http://sm.mycloud.com/SM/api/monitor/meters/<confid>/hosts/<VMid>/CPU%

which returns the last value of metric "*CPU%*" for virtual machine <*VMid*> of configuration <*confid*>.

5. Experiments and validation

The ICARO cloud platform is currently used to manage the dynamic workload of several complex configurations on cloud. In this section, an example is provided to show the SCE effectiveness, and to release some data and consideration about the typical behavior.

The discussed case study is referring to the ECLAP real social network and service accessible and running on http://www.eclap.eu. ECLAP is a thematic best practice network on performing arts, used for entertainment and social learning purposes by about 35 institutions in Europe and many thousands of users. Being related to education and training, it suffers for a strong seasonal evolution of the workload, with daily peaks for the exams and exercitations. As depicted in Fig. 3, the ECLAP solution presents a scalable frontend and backend with indexing and searching facilities, automatic content ingestion service, and load balancing, using a range of 11–25 VMs according to the workload, and running several services (i.e., HTTP balancers, front–end VM with Web Servers, Apache Tomcat Servlet Containers, federated MySQL database, an AXCP Scheduler and several AXCP Grid Nodes for managing the parallel ingestion and adaptation of cross media, for example tens of thousands of long videos per day, slides, images, documents, etc.) [27]. The ingestion is performed by both the FTP ingestion server and from the back office node, respectively in push and pull, mainly.

The scalability of the front–end part of the whole ECLAP service is managed with a configuration where one Front-End VM is active running the Apache Web Server (behind an HTTP Balancer), and a SLA is defined to assure that the average CPU percentage workload over last 10 min has to be less than 70%. If the SLA is violated a set of actions have to be performed: an additional front–end VM is powered on, the HTTP balancer is updated, and thus an updated configuration for monitoring and management has to be set. In the configuration with two front–end VMs, according to the SLA, the cloud could pass to activate a third VM on front end or to reduce the front–end engagement of VMs, by returning to a single front–end VM. Thus, the SLA rule activated in the condition of two VM includes: (i) when the average CPU percentage on 30 min of both frond-end VMs are above the 80% the third VM is activated (when all of three are above a warning message is sent); (ii) when both of the front–end VMs present the averaged CPU percentage on last 30 min below of 10%, then the second front–end

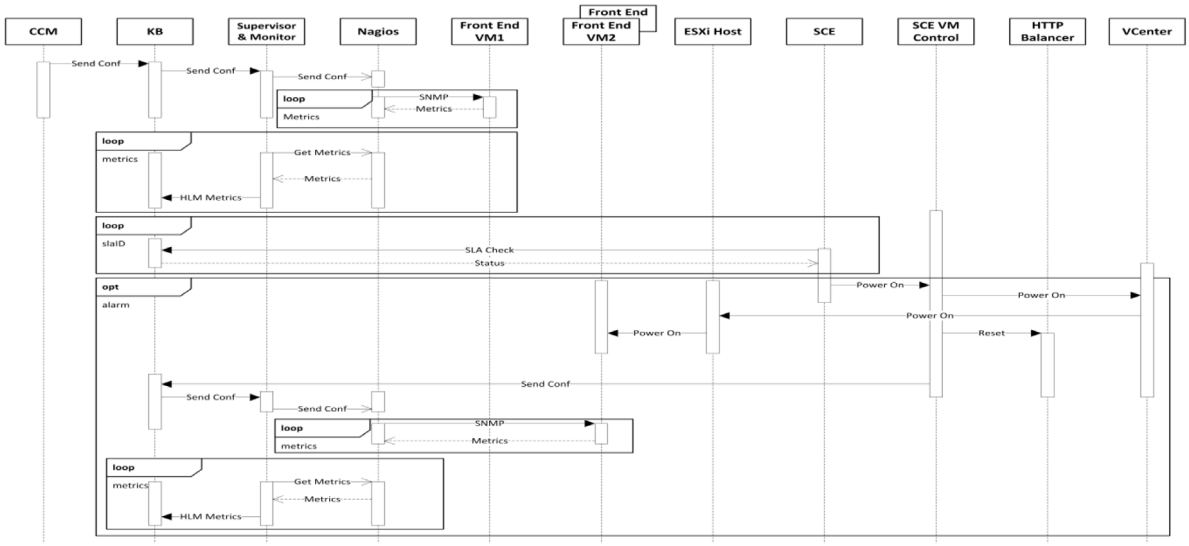

Fig. 4. Sequence diagram about the web front end scaling up and down managed by the SCE.

VM is shut-down. In any case, the balancer and the monitoring VMs have to be reconfigured; thus all the changes on the configurations have to be sent to the KB. Thus, according to the requests and SLA, the solution may returns to work on single front end VM with 70% CPU. The two different metrics over 10 and 30 min for scaling up and down with a sort of hysteresis. The 10 min has been defined to rapidly identify a high workload condition avoiding reacting to sporadic pikes. And, the 30 min has been set to slowly switching back to the low workload configuration.

In Fig. 4, a sequence diagram reporting the interactions among the components of SCE cloud manager is reported. At the beginning, the CCM uploads to the KB the starting configuration with one front–end VM; the configuration is verified by the KB and then is forwarded to the SM, to setup the monitoring of the configuration, by means of Nagios, by using LLM. The Nagios monitoring begins acquiring the values of LLMs using the SNMP protocol, and in the meanwhile the SM begins to evaluate the HLMs (defined in the SLA) for the different configurations stored on the KB (once every 10 min). The SCE periodically evaluates the SLA conditions by making SPARQL queries on the KB. When a SLO of a SLA is not true (e.g., the AVG CPU over 10 min is more than 70%) the URL specified in the action is called and a new front–end VM is turned on and started (by using the SCE VM Control and thus the vCenter API). Then, the HTTP balancer is reconfigured and the new configuration is uploaded on the KB with the new front–end VM (activating a new segment of the SLA). From this point, the SCE will continue evaluating the SLAs, and when the workload decreases the SLA triggers a call to a URL that reconfigures the balancer: shutting down the not useful front–end VM, and uploading a new configuration to the KB; thus moving to a cheaper management of the lower workload.

In Fig. 5, the temporal evolution of averaged CPU percentage (CPU AVG %) of the above-described activity of scaling (described in Fig. 4) is shown. The data have been taken every 10 min and the averaged values estimated every 10 and 30 min, according to the above description and SLA HLMs. The above values have been set up according to the typical workload trend on the social network.

In the example of Fig. 5, the workload on the front end of ECLAP started slowly in the morning and close to the 10:40 am reached 50%. Later, the CPU AVG% (estimated every 10 min) become greater than 70%. Therefore, the SCE (according to the SLA) activated a second front–end VM. Thus, after 20 min (WSTS), the started VM completed the boot and also activate the Web Server, thus the global workload for the front–end decreased for the presence of two VMs providing the service. As a consequence of the reconfiguration the SLA passed to the new modality (potentially waiting for adding more VM in the front–end or removing one for scaling down). Later, at 15:01, the HTTP workload on the balanced service decreased the CPU AVG % was lower than 10% on both VMs, thus the second front–end VM has been powered off, and the SLA activated previous rules, returning to work with a single front–end VM.

5.1. Statistical data and considerations

The above described SCE and solution has been deployed on the cloud infrastructure of ComputerGross. The most complex SLA consists of 75 conditions with respect to a service application (i.e., http://www.eclap.eu, a social network with scalable frontend and backend, with indexing and searching facilities, automatic content ingestion service, in load balance) as described above.

The first part of Table 1 reports the reference values for SCE and KB in terms of KB size and time period for HLM computation. The second part of Table 1 reports the typical performance value related to the SCE management on cloud, registered for scaling activities of the ECLAP configuration: passing from 1 to 2 VMs and then returning again to a single front–end VM. In the 30 days of the measure the ECLAP service (http://www.eclap.eu) has been scaled 11 times, and the average duration of the scaling has been of about 3 h:40 m. The VMTS is the time from the SLA violation to start a second VM (from its off condition); it has been of 5 min on average (it was a Windows Server 2008 64 bits). This time interval is about one half the SLA check interval (10 min) used by the SCE to evaluate the HLM metrics and in case of violation do the appropriate action (start the VM2 in this case). WSTS time interval is the VMTS plus the time to boot the VM plus the time to have the web server running, it can be regarded as the reaction time that has been measured to be about 20 min, with a low standard deviation.

The VMTS (and thus the WSTS) can be reduced by diminishing the SLA checking interval while the most relevant part of WSTS can be diminished by keeping the VM paused/hibernated (not totally off). The averaged time for estimating the SLA violation in the scaled

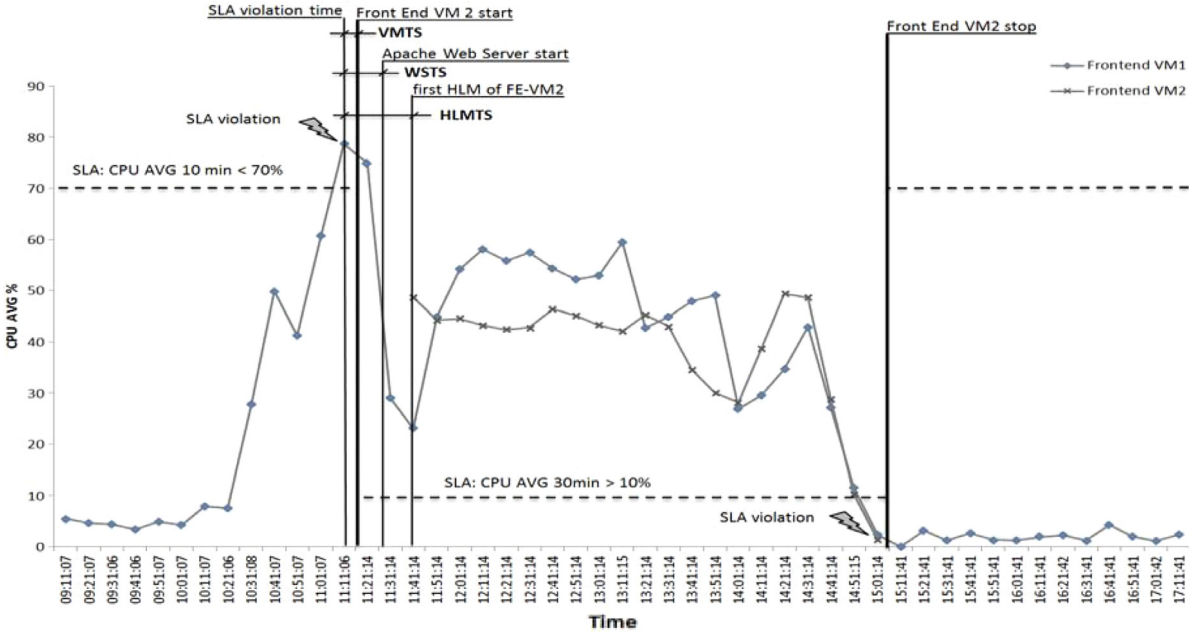

Fig. 5. Typical trend of the CPU AVG % when the web front–end of ECLAP passes from 1 to 2 VMs and then return again to a single front–end VM.

Table 1
SCE and KB reference values. SCE and cloud performance registered for scaling activities on the ECLAP configuration: passing from 1 to 2 VMs and then returning again to a single front–end VM.

General aspect of the SCE and KB				
Number of service metrics in the KB store	1.163.195	Triples		
Number of triples (stated)	7.050.075	Triples		
Number of triples (stated + inferred)	18.535.693	Triples		
SLA check interval	10	Minutes		
HLM calculation interval	10	Minutes		
Time period for the following data	30	Days		
Number of times scaling started/completed	11	Time		
SCE and cloud performance in scaling ECLAP	**Averaged values**	**Standard deviation**	**Max value**	**Min value**
Scaling duration time	3 h 40 m	2 h 26 m	9 h 9 m	49 m
Time to start front end VM 2 machine from SLA violation time (VMTS)	5 m	2 m		
Time to start apache on frontend VM 2 from SLA violation time (WSTS)	20 m	6 m		
Time to get a new HLM measure of front end VM 2 from SLA violation time (HLMTS)	30 m	10 m		
Time to upload a configuration to KB	20 s	20 s		
Time to upload a configuration to SM	37 s	5 s		
Time for SLA evaluation query	137 ms	672 ms		

configuration is greater than VMTS as reported in Table 1. If a different approach is taken, pausing the VM, the WSTS time is in the range of 5 min and the SLA time is reduced to 7–10 min.

Moreover, the SCE also controlled a number of additional values such as memory usage, disk usage, network workload, MySQL size, Apache response time, MySQL response time, Tomcat response time. The majority of critical overvalues detected in the period were related to memory (11.86%), disk usage (8.57%), network traffic (3.85%), and database size (2.12%). As a general consideration, the ECLAP business configuration experienced a number of changes in the back office, in addition to those on the front–end.

The critical aspects of Smart Cloud engine is the scalability. The scalability regards mainly the costs connected to the monitoring, supervising and managing the cloud structures and resources. Almost all cloud infrastructures are endowed of some monitoring engine, for monitoring cloud resources and processes at IaaS and PaaS levels. The state of the art presents a range of them, and almost all are quite scalable since they can manage with a single server several thousands of processes and may be scaled up by replicating them, cluster by cluster. Their scalability is also assured by the fact that the data they collect are typically decimated in the time (passing from a sample every 5 s to a sample every hour). Thus, they store recent data at a high temporal resolution, while older data are progressively reduced mediating them, or subsampling them. This approach can be a problem for the SCE that may need to compute high-level metrics, some of them may be grounded on historical data of the year or more. To this end, the SCE collects all data and estimates the high level metrics, minimizing the needed 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, led to have a single non federated RDF store of about 95 millions of triples. The present big data RDF stores as OWLIM and Virtuoso can scale up to billions of triples on federated storages. In these conditions, monitoring VM health and SLA, we have about 288.1 SCE processes per hour that are executed on 3 VMs, for a total of 32 GHz loaded at the 25% in average. The success rate for the scheduled jobs is typically the 98% in the week. The failures are

mainly due to the lack of connection due to the restart of some fail over solution. All of them are recovered at the successive execution and/or retrial, and when this is not possible an alarm is set to the operator. The scaling of the solution can be easily obtained by creating a federated cluster of RDF store, while the scheduler is fully distributed and any new node can be attached at run time without problems of scalability. The recurrent costs of having a SCE solution with respect to a simple monitoring service is about 3 times, that is to have two more servers for each monitoring server. On the other hand, the SCE can save a lot of time for the operators that do not need to monitor and assess trends and data, to take decision about the business configuration adaptation, according to the SLA.

6. Conclusions and applicability of the ICARO solution

This paper presented a smart cloud solution based on an innovative knowledge model, allowing a fiexible management of the cloud resources: verification and management of solution for cloud control and reconfiguration. The solution proposed is scalable and provides smart reasoning and management for cloud platforms; it can be applied to different contexts, load balancing fault tolerance, automatic scaling, and elastic computing features. It includes a Knowledge Base (KB), for modeling cloud resources, Service Level Agreements (SLA) and their evolution, and enabling the reasoning on structures by implementing strategies of efficient smart cloud management and intelligence. Via the KB, it provides formal verification and intelligence tools for cloud control and can be easily integrated with any cloud configuration manager, cloud orchestrator, and monitoring tools, since the connections with these tools are performed by using REST calls and XML files. It has been validated with cloud service providers on complex cloud configurations with good performance in terms of low operating workload and scalability The limits of the solution are mainly located in the limited complexity of the decision rules that one can formalize directly in the user interface. When complex rules for taking decision are needed the solution can be to implement specific processes in Java that are executed by DISCES of the SCE.

Acknowledgments

The authors want to thank all the partners involved in ICARO, as ComputerGross and LiberoLogico, and the Regione Toscana (grant no. 567075 CUP ARTEA) for funding the project. ICARO has been funded in the POR CReO 2007–2013 programme.

References

[1] L. Wu, R. Buyya, Service level agreement (SLA) in utility computing systems, in: V. Cardellini, et al. (Eds.), Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions, IGI Global, USA, 2011.
[2] A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, U. Villano, An SLA-based broker for cloud infrastructures, J. Grid Comput. 11 (1) (2013) 1–25. 2012.
[3] X. Pengcheng, et al., Intelligent management of virtualized resources for database systems in cloud environment, in: 2011 IEEE 27th International Conference on Data Engineering (ICDE), IEEE, 2011.
[4] J.S. Ward, A. Barker, Observing the clouds: a survey and taxonomy of cloud monitoring, J. Cloud Comput. (2014).
[5] L. Youseff, M. Butrico, D. Da Silva, Towards a unified ontology of cloud computing, in: Grid Computing Environments Workshop, 2008. GCE '08, Nov 2008, pp. 1–10.
[6] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, M. Menzel, S. Nepal, An ontology-based system for cloud infrastructure services' discovery, in: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, 2012, 14–17 Oct. 2012, pp. 524,530.
[7] G. Dobson, et al. QoSOnt: a QoS ontology for service-centric systems, in: 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005. 2005, pp. 80–87.
[8] M. Ghijsen, J. van der Ham, P. Grosso, C. de Laat, Towards an infrastructure description language for modeling computing infrastructures, in: The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.
[9] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, V. Munteanu, An analysis of mOSAIC ontology for cloud resources annotation, in: Federated Conference on Computer Science and Information Systems, FedCSIS, 18–21 Sept. 2011, pp. 973–980. http://www.mosaic-cloud.eu/.
[10] C. Pedrinaci, J. Cardoso, T. Leidig, Linked USDL: a Vocabulary for web-scale service trading, in: 11th Extended Semantic Web Conference, ESWC 2014, Springer, 2014, http://linked-usdl.org/.
[11] T. Binz, G. Breiter, F. Leymann, T. Spatzier, Portable cloud services using TOSCA, IEEE Internet Comput. (2012).
[12] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, TOSCA: Portable automated deployment and management of cloud applications, in: Advanced Web Services, Springer, 2014.
[13] D. Bernstein, D. Vij, Using semantic web ontology for intercloud directories and exchanges, in: Proc. International Conference on Internet Computing, 2010.
[14] D. Androcec, N. Vrcek, J. Seva, Cloud computing ontologies: A systematic review, in: Proc. of MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, Chamonix, France, April 29, 2012.
[15] H. Ludwig, A. Keller, A. Dan, Web Service Level Agreement (WSLA) Language Specification, January 28 2003, available at: http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf.
[16] A. Andrieux, K. Czajkowski, A. Dan, et al. Web Services Agreement Specification, WS-Agreement, March 14 2007, available at: http://www.ogf.org/documents/GFD.107.pdf.
[17] N. Oldham, K. Verma, A. Sheth, F. Hakimpour, Semantic WS-agreement partner selection, in: Proceedings of the 15th International Conference on World Wide Web (Edinburgh, Scotland, May 23–26, 2006), WWW'06, ACM Press, New York, NY, 2006, pp. 697–706.
[18] P. Bellini, P. Nesi, A. Venturi, Linked open graph: browsing multiple SPARQL entry points to build your own LOD views, Int, J. Vis. Lang. Comput. (2014) Elsevier.
[19] P. Bellini, D. Cenni, P. Nesi, Cloud knowledge modeling and management, in: Encyclopedia on Cloud Computing, Wiley Press, 2015, (Chapter).
[20] C. Badii, P. Bellini, I. Bruno, D. Cenni, R. Mariucci, P. Nesi, ICARO cloud simulator exploiting knowledge base, J.: Simul. Model. Pract. Theory (ISSN: 1569-190X) 62 (2016) 1–13. http://dx.doi.org/10.1016/j.simpat.2015.12.001, Elsevier.
[21] E. Sirin, J. Tao, 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.
[22] S. Sindhu, S. Mukherjee, Efficient task scheduling algorithms for cloud computing environment, High Perform. Archit. Grid Comput. Commun. Comput. Inf. Sci. 169 (2011) 79–83.
[23] L. Ma, Y. Lu, F. Zhang, S. Sun, Dynamic task scheduling in cloud computing based on greedy strategy, Commun. Comput. Inf. Sci. 320 (2013) 156–162.
[24] J. Tsaia, J. Fanga, J. Chou, Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm, Comput. Oper. Res. 40 (12) (2013) 3045–3055. Elsevier.
[25] https://libcloud.apache.org/.
[26] https://jclouds.apache.org/.
[27] P. Bellini, I. Bruno, P. Nesi, N. Paolucci, Institutional services and tools for content, metadata and IPR management, Int. J. Softw. Eng. Knowl. Eng. 25 (08) (2015) http://dx.doi.org/10.1142/S0218194015500242, World Scientific Publishing Company.

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 Ph.D. 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 Ph.D. 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 Ph.D. 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.

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.