A Framework for the 3-D Cloud Monitoring based on Data Stream Generation and Analysis

Dario Bruneo*, Francesco Longo*, Clarissa Cassales Marquezan†
*Dipartimento di Ingegneria DICIEAMA, Università degli Studi di Messina, Messina, Italia
{dbruneo,flongo}@unime.it
†Huawei European Research Center, Riesstrasse 25, Munich, Germany
clarissa.marquezan@huawei.com

***Abstract*—Cloud monitoring is one important aspect for effective cloud management. Currently, cloud monitoring solutions can be classified into three groups: the ones not considering multiple layers or real time data analysis, the ones considering multiple layers but not real time data analysis, and the ones only considering real time data analysis. However, all these solutions fail to provide frameworks able to combine together monitoring in multiple layers and data stream analysis for detecting situations where multiple management actions are applicable in different layers of the cloud environment. This paper addresses this gap and proposes the Ceiloesper framework. Such a framework extends the OpenStack Ceilometer technology with Esper CEP and enables collection and analysis of information according to the principles defined in the 3-D cloud monitoring model, proposed in a previous work. The main contributions of this paper are: (i) the definition of the concept of Situation of Interest (SoI) leading to multiple management actions; (ii) the Ceiloesper architecture for a monitoring solution combining traditional monitoring elements with CEP; (iii) extensions to the Ceilometer OpenStack technology. We tested the Ceiloesper framework on a scenario based on the Wordpress application and the experimental results show its effectiveness.**

I. Introduction

Cloud management goes beyond the management of physical versus virtual resources. Inside the virtual layer there are also management decisions and actions being activated leading to multiple and most of the time isolated management decisions. We have already identified the existence of four different layers inside the cloud environment [1]: physical (PL), virtualization (VL), application architecture (AAL) and application business logic (ABL) layers. Each of them have entities that request and that suffer the effects of management actions inside the cloud environment. In a previous work, we demonstrated the need for a 3 Dimensional model (3-D Cloud Monitoring model) [2] for representing and analyzing monitoring information from multiple layers in the cloud environment to enable a better coordination among the possible different management actions in these layers. There are many solutions proposed for cloud monitoring and they can be classified into three groups: the ones not considering multiple layers or real time data analysis [3][4]; the ones considering multiple layers but not real time data analysis [5][6]; and the ones considering solely real time data analysis [7][8][9][10].

In the first group, we can find a large amount of work related to general purpose frameworks typically supporting monitoring from PL and VL layers. New solutions started to consider also the AAL layer [11]. There are also monitoring frameworks specifically targeting the cloud applications [12]. Recently, proposals belonging to the second group started to appear. The works from Trihinas et al. [5] and Montes et al. [6] take into account information from layers equivalents to the ones we consider in our 3-D Cloud Monitoring model. Nevertheless, these frameworks work basically with the concept of collecting information, storing it in a database, enabling the creation of rules to trigger events related to the stored information, and the configuration of simple events to be triggered by the monitored entities.

These approaches have several limitations when requiring a monitoring solution to detect real time situations leading to multiple management actions. The major problem is the offline and "post-morten" analysis of the stored data. The real time detection of events and measurement of metrics in each layer is not possible due to the introduction of the overhead for collecting the data, storing it, querying it, and perform the alignment of the information from all the layers from the database. There are many intermediary processing overheads, and different intervals of cycles of processing in this chain. This might lead to a very delayed identification of situations. Indeed, Ari et al. [10] alerted for the importance of velocity of the monitored data and its relationship to monitoring overheads.

The third group represents an emerging trend also associated with the increase of research in cloud analytics [7][8][9][10]. Proposals such as the one from Saleh et al. [9] explicitly define monitoring frameworks able to handle data stream processing. They basically use Complex Event Processing (CEP) engines to support the real time analysis of the collected data. These works provide a very solid basis and argumentation about the importance of using data stream processing. However, there is still a gap on combining the advantages of correlating the information monitored from all the cloud layers with the power of data stream analysis for managing the cloud environment. Creating a monitoring framework based on streams of real

At the time of writing this paper Clarissa Cassales Marquezan was not a member of the Huawei European Research Center.

time monitoring data is crucial. Based on this it is possible to enhance the quality and the timeliness of the information upon which the multiple management decisions are taken. In this paper, we propose a framework to support the 3-D Cloud Monitoring model, based on multiple cloud layers monitoring and data stream analysis.

We propose a solution that extends the functionalities of current cloud monitoring frameworks by combining them with the capabilities of CEP. The ultimate goal is to enable an online, distributed, and dynamic monitoring process for the identification of situations leading to multiple management actions. These situations are related to the concept of Context of Interest (*CoI*) and the 3-D Cloud Monitoring model, previously introduced [2]. The framework proposed in this work is based on OpenStack technology. We created the Ceiloesper framework that provides extensions to the Ceilometer technology through an integration with the Esper CEP framework. We tested the Ceiloesper framework on a scenario based on the Wordpress application. The experiments show that our framework can identify situations, trigger management actions, and does not introduce a large load in the running cloud environment. The main contributions of this paper are: (i) the definition of the concept of Situation of Interest (*SoI*) that can lead to the identification of multiple management actions; (ii) the architecture for a monitoring solution combining traditional monitoring elements with CEP; (iii) extensions to the OpenStack Ceilometer technology.

The remainder of this paper is organized as follows. Section II analyzes the current state of the art. In Section III, we show the architecture and implementation of the Ceiloesper framework revising our 3-D Cloud Monitoring model and providing a formal definition of *SoI*. In Section IV we show experimental results and the effectiveness of the proposed architecture. Finally, Section V concludes the work providing insights in possible future work.

II. Related Work

In this section, we focus solely in the related work associated with monitoring solutions for multiple layers of the cloud, and the ones using data stream processing and CEP for cloud monitoring and analysis.

Thrihinas et al. [5] propose a monitoring system that handles information from PL, VL, and AAL layers. The main purpose of the solution is to better support the provisioning of elastic applications. They provide: a solution able to keep monitoring the application metrics even when the topology of the application changes (due to scaling or migration in the cloud); the control of data aggregation in the monitoring agents; a language to combine low level metrics and generate higher level metrics. Likewise, Montes et al. [6] introduced the idea of levels of monitoring in a cloud environment. They defined the monitoring in the virtual system level (composed by the sub-levels application, platform, and infrastructure) and the physical system level. The authors defined a platform for monitoring such levels called GMonE (Global Monitoring systEm). The focus of this work is the collection of the monitoring information and not how the information is processed and associated with different layers of monitoring to be correlated. The fundamental difference of our approach to the ones described above is the use of complex event processing in combining the information coming from all the different cloud layers. Although powerful in controlling and configuring the monitoring infrastructure itself, these approaches still lack the capability of handling complex stream data analysis.

Ari et al. [10] acknowledge the fact that handling data velocity is very important and the overhead of storing data for later processing imposes loses of opportunities. The proposed work focus on the details of combining correlation techniques with rule mining based on Esper CEP engines. Although clouds are mentioned in the paper, their proposal is not explicit target to the management of the cloud environments. Tudoran et al. [7] propose a system called JetStream whose focus is the optimization of event streaming transmission over multiple cloud datacenters. Mdhaffar et al. [8] analyze the performance impact in cloud monitoring process when different architectures are configured for CEP Esper engines executing in the cloud environment. In both the above mentioned works, there is no real focus on creating a framework for cloud monitoring using data streams for detecting situations leading to multiple management actions in the cloud environment. Saleh et al. [9] uses CEP for detecting complex situations and activating management actions. However, their approach is limited to management actions related to the PL and VL layers, e.g., increasing and decreasing of virtual resources or migration. The complex situations in their case relay on temporal windows and monitored information (*i.e.*, events) related up to the VL. Leitner et al. [12] introduce a framework based on CEP where application developers can define which application metrics should be monitored. The authors collect the desired application information from single hosts (i.e., single virtual machines), pools of hosts (i.e., pools of virtual machines) and correlate the events carrying this information. However, the correlation is understood by the authors as filtering and aggregation of data to reduce the flooding of information. No sophisticated analysis specifically target to activating management actions is proposed in this work.

Monitoring in multiple layers is orthogonal to analysis based on data stream and CEP. The former collects information from different layers while the latter analyzes in real time this data. In fact, collecting data in real time does not guarantee that this data will be treated also in real time, *i.e.*, also at the moment it is collected. Therefore there is still a gap on defining monitoring frameworks that are able to collect and analyze information from all the cloud layers, using for this a data stream style of analysis. This type of analysis is vital to guarantee the timeless and accuracy of decisions in the multiple management entities spread along the cloud stack.

III. Ceiloesper Framework

In this section, we introduce the architecture and implementation of the Ceiloesper Framework. Before describing the details of this framework, we shortly revise the concepts of

3-D Cloud Monitoring model and *CoI* defined in a previous work [2]. In addition, we introduce the concept of *SoI* which is related to the concepts defined previously. The proposed architecture is designed to support these concepts. The goal of this architecture is twofold. First, it enables the timely detection of situations in the cloud environment where multiple management actions can be enforced in different layers of the cloud. Second, it triggers events associated with these multiple management actions so that a dedicated engine can receive these triggers and decide which action should be taken. This specific engine, selecting the best management action, is out of the scope of this paper. The Ceiloesper Framework is based on the following concepts and assumptions. The OpenStack Ceilometer is the basic mechanism for collecting monitoring information in the cloud environment. We believe that keeping the alignment with the well established OpenStack community is important. This means that the new components we introduce need to be compliant with the Ceilometer architecture. We consider that the framework main purpose is to support the coordination of management actions among different layers of the cloud environment. It is out of the scope of this paper to discuss economical model and the issues related to who owns which information in the cloud environment. Finally, we assume that solutions based on CEP mechanisms and principles are effective in handling data stream generation and analysis in real time. In this work, we use Esper CEP Engine due to its powerful language to express complex situations in terms of events and its processing capacity [8].

A. Background

Figure 1 graphically depicts the 3-D Cloud Monitoring model and one example of *CoI* as both defined in our previous work [2]. The 3-D Cloud Monitoring model is used to analyze the monitoring information in a cloud environment according to three dimensions: the physical machine (PM) from where metrics are collected, the application to which the metrics are associated with, and the type of metrics and their associated cloud layer. Using this model, we defined the *CoI*. In summary, a *CoI* is a collection of metrics from the 3-D space of collected data that has to be observed and analyzed simultaneously. The analysis of a *CoI* will necessarily lead to the identification of multiple (and eventually conflicting) management actions in different layers of the cloud. This is the major goal when monitoring and analyzing with the *CoI*.

The example illustrated in Figure 1, shows metrics associated with the app_i deployed over pm_j that, when monitored and analyzed together, lead to different management actions. In this example, a Web application is used and the multiple management actions could be: change the number of connections in the database associated with the application, or increase the amount of CPU, or migrate a virtual machine (VM) hosting one or more components of such an application. Nevertheless, the *CoI* definition establishes which set of monitoring metrics should be analyzed together. It does not indicate how these metrics should be collected and analyzed. Therefore, in this paper we introduce the concept of *SoI* and a framework to support the automatic use of both *CoI* and *SoI*.

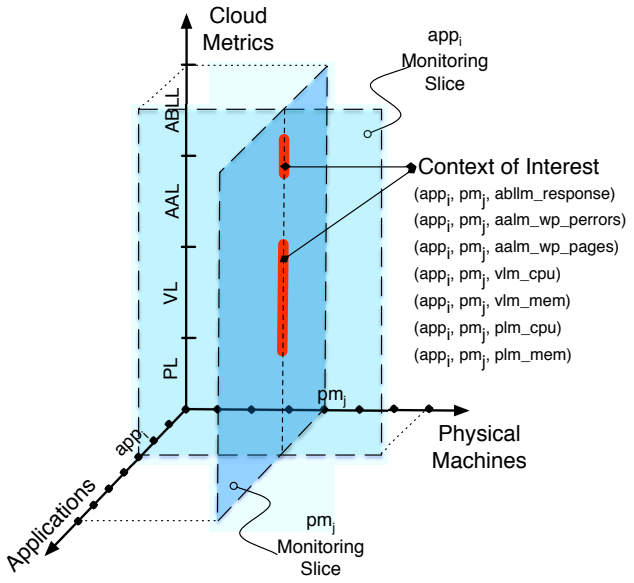


Fig. 1. 3-D Cloud Monitoring and *CoI* [2]

B. Situation of Interest

A *Situation of Interest* (*SoI*) expresses which kind of combination of monitored metric values from a *CoI* are relevant for the detection of a circumstance leading to (potentially) multiple management actions. Below, we provide the mathematical definitions necessary to describe the *SoI* concept.

Definition 1 (Application): An application is a set of components (e.g., systems, sub-systems) and it has necessarily a unique identifier in the cloud environment (app_{id}).

Definition 2 (Application Context of Interest): An application *CoI* $C^{app_{id}}$ for an application app_{id} deployed in the cloud environment is a set of distinct monitoring metrics m_j belonging to the 3-D Cloud Monitoring model, as defined in [2]. Formally, $C^{app_{id}} = \{m_j\}$ where $1 \leq j \leq J$ and J is the total number of metrics belonging to the *CoI* $C^{app_{id}}$.

Definition 3 (Application Context of Interest Set): An application *CoI* set $\mathscr{S}_{CoI}^{app_{id}}$ is the set of distinct *CoI*s associated with an application app_{id} deployed in the cloud environment. Formally, $\mathscr{S}_{CoI}^{app_{id}} = \{C_i^{app_{id}}\}$ where $1 \leq i \leq I$ and I is the total number of *CoI*s associated with the application app_{id}.

Definition 4 (Situation of Interest): A *SoI* $S^{app_{id}}$ associated with a *CoI* $C^{app_{id}}$ for an application app_{id} deployed in the cloud environment is a couple (CS, Ev), where CS is a complex statement containing the information to be analyzed in order to detect a desired circumstance and Ev is an event that should be generated when the CS evaluates to $true$.

Definition 5 (Complex Statement): A Complex Statement for an application app_{id} is a set of tuples $CS = \{((m_j, exp), T, [Men])\}$ where $m_j \in C_i^{app_{id}}$ and

$C_i^{app_{id}} \in \mathscr{S}_{CoI}^{app_{id}}$; exp is a mathematical expression with an operator op and a value v to be compared to the value of the monitoring metric m_j; T is the time window of observation of m_j; Men is an optional element of the tuple that serves as a memory storing information about the occurrence of one or more specific events before or within the observation time window T. The Men term can support the tracking of previous events, allowing for the analysis of chain of events. Our main goal is to enable the analysis of the monitoring metrics in combination with previous activation and deactivation of management actions. A Complex Statement CS evaluates to $true$ if all its tuple evaluate to $true$, i.e., if for each tuple the mathematical expression exp on metric m_j evaluates to true in the time window T together with the optional memory element $[Men]$.

Definition 6 (Application Situation of Interest Set): An application SoI set $\mathscr{S}_{SoI}^{app_{id}}$ is a set of distinct SoI associated with an application app_{id} deployed in the cloud environment. Formally, $\mathscr{S}_{SoI}^{app_{id}} = \{S_k^{app_{id}}\}$ where $1 \leq k \leq K$ and K is the total number of SoI associated with all the CoIs belonging to the application CoI set of application app_{id}.

C. Architecture and Implementation

The architecture of the Ceiloesper Framework is presented in Figure 2, and it is based on OpenStack technology. There are three main types of components in this architecture: the OpenStack Compute Node, the OpenStack Controller Node, and the Analysis Engine. All light and dark red modules in Figure 2 represent the native modules of OpenStack components. The modules added by the Ceiloesper Framework are represented with the light and dark blue colors. The yellow colored diagrams in this figure indicate native elements either of the PMs or of the VMs. In this figure, we also depicted to which modules the CoI and SoI are relevant.

A OpenStack Compute Node is a PM in which a hypervisor (usually Libvirt) is installed and on top of which the OpenStack Nova service (implementation of a Compute Node) is used to control the instantiation and the life-cycle of VMs. Thus, the set of Compute nodes belonging to an OpenStack infrastructure form what we defined as PL. An OpenStack Controller Node, in its turn, is a PM on top of which the Nova Scheduler service (along with eventually other services) is installed allowing to schedule the instantiation of VMs on a set of Nova Compute nodes and to manage them remotely. The OpenStack Controller machine also hosts the *Ceilometer Controller* that is a centralized component for data storage within a distributed database, *e.g.* Mongo DB. The Controller Node also manages the OpenStack Dashboard Horizon, where Ceilometer displays the collected information.

We introduced a set of new modules in both the OpenStack Compute Nodes and Controller Node in order to support the architecture of the Ceiloesper framework. In addition, we also introduced an *Analysis Engine* that is responsible for the real time analysis of the streams of monitored data coming from the cloud environment. The new modules and engines

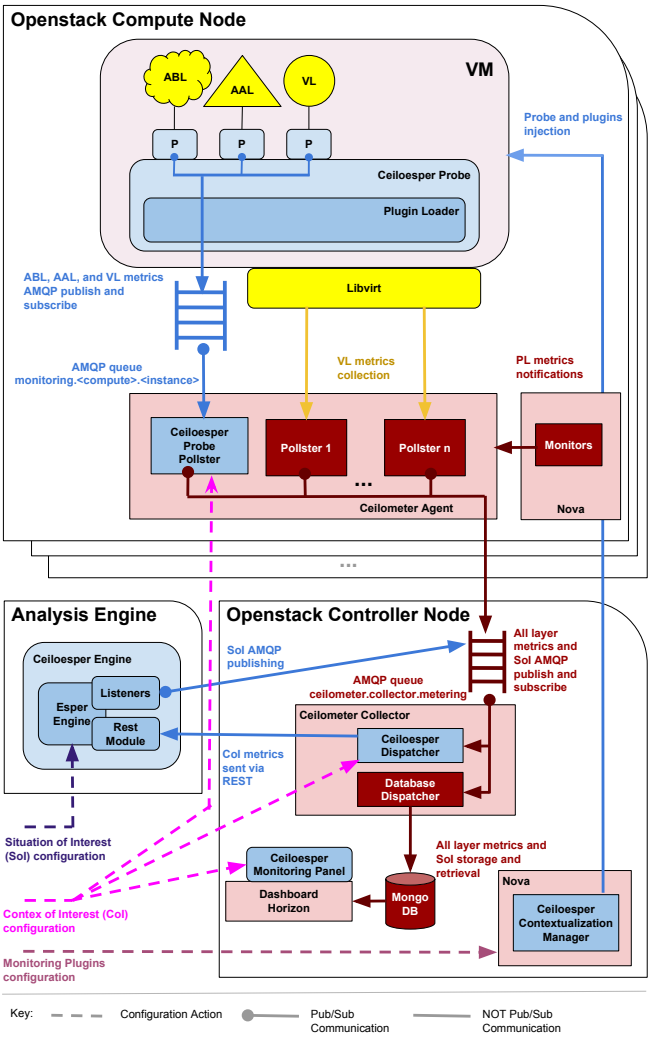


Fig. 2. Detailed architecture of Ceiloesper Framework

were designed to behave according to the specification of CoI and SoI. The CoI collection and visualization is implemented through a set of additional modules for OpenStack Ceilometer, OpenStack Horizon, and OpenStack Nova services. These new modules are spread over PMs hosting the OpenStack Compute Node and the OpenStack Controller Node, as indicated in Figure 2. The inference of a SoI is performed by a set of new modules implemented in the Analysis Engine. This engine is based on ESPER. The new modules of the Ceiloesper Framework are described as follows.

1) Inside the VMs: The *Ceiloesper Probe* inside the VM is a component injected in each VM that allows to load one or more plugins (via the *Plugin Loader*) to collect measurements and event at the ABL and AAL layers and from the systems running inside the VMs at the VL. In this case, owners of VMs might want to retrieve information from the VM different than the ones already provided via Libvirt. The Probe uses a shared library (`.so`) to send measurements and event to the OpenStack Ceilometer. The current implementation is based in Python using the Stevedore library for enabling

and loading a monitoring plugin. Examples of such plugins are: LAMP-based applications (*e.g.*, Wordpress), OS statistics (CPU, RAM, etc) and legacy application (MySQL) deployed in the VM. The collected information from the *Ceiloesper Probe* of each VM deployed in a Compute Node is transmitted to the *Ceiloesper Probe Polster* (inside the Compute Node, as it will be further discussed). The communication between *Ceiloesper Probe* and *Ceiloesper Probe Pollster* is based on the Apache Qpid messaging system, that provides the implementation of the AMQP (Advanced Message Queuing Protocol) commonly used in OpenStack components. The AMQP is an open standard application layer protocol for message-oriented middleware able to provide bus featuring message orientation, queuing, routing (including point-to-point and publish-subscribe), reliability, and security. The circle-edged arrows represent the interaction model, where information is transmitted using bus-enabled queues, publishers as well as subscribers.

2) Compute Node Side: In each Compute Node, a typical *Ceilometer Agent* is able to collect metrics from the PM itself (PL metrics) and from the instantiated VMs through the interaction with Libvirt (VL metrics). The PL metric collection became available in recent OpenStack releases. The Compute Node has been integrated with a set of *Monitors* that produce Ceilometer notifications about PL metrics. At the moment the default OpenStack Ceilometer Pollsters collect information only through the Libvirt. This solution enables the collection of information only up to the VL, and excluded the collection of data from inside the VMs. In this work, we want to keep the compatibility with the OpenStack Ceilometer architecture. The official recommendation for properly extending Ceilometer with extra metrics is to implement *Pollsters*, which are compliant and able to be integrated to the *Ceilometer* architecture. Therefore, we inserted inside the *Ceilometer Agent*, the *Ceiloesper Probe Pollster*. This pollster is responsible for collecting information from the ABL, AAL, and extra VL information as previous described. A connection is established, using Qpid method, between the *Ceiloesper Probe Pollster* and the AMQP queue of each VM instance running in the Compute Node hosting such a pollster. We configured this new pollster to be available in the *Ceilometer Agent* at boot time. The *Ceiloesper Probe Pollster* extracts from these queues the messages bearing the samples sent by *Ceiloesper Probe* plugins inside the VMs. The resuting information is used to build a Ceilometer compliant sample. Each sample is processed by the *Ceilometer Agent* and sent to the *Ceilometer Collector* hosted on the Controller Node. This communication is done using the ad-hoc queue `ceilometer.collector.metering`.

3) Controller Node Side: The Ceilometer architecture is composed of agents hosted in the Compute Nodes (as described in the previous sections) and the *Ceilometer Collector* module hosted at the Controller Node. This module is responsible for receiving and storing the collected monitored information through the *Database Dispatcher*. This solution as it is does not allow for analysis of streams of monitored data because the information not available in real time. Instead it is stored in a database and needs to be queried. Therefore we added to the *Ceilometer Collector* a new dispatcher called: *Ceiloesper Dispatcher*. This new element does not modify the core configuration of neither Ceilometer nor OpenStack overall services. This new dispatcher listens to the data published in the Ceilometer queue (`ceilometer.collector.metering`) but analyzes only measurements coming from samples marked to the processed by the Ceiloesper Framework modules. When samples are created by the Ceilometer Agents, these samples are published in the Ceilometer queue with a specific field configuration that allows the *Ceiloesper Dispatcher* to recognize the sample and send the information to the *Analysis Engine*. In addition, we also define two other Ceiloesper Framework modules that are hosted by the Controller Node: the *Ceiloesper Monitoring Panel* and the *Ceiloesper Contextualization Manager*. The former allows for the extension of the OpenStack Dashboard Horizon. The extension enable the infrastructure managers and the developers to monitor all the measurements and events collected from the infrastructure and cloud platform, up to the data collected from the applications inside the VMs. The latter, *Ceiloesper Contextualization Manager* enables the automatic installation of the *Ceiloesper Probe* and its monitoring plugins inside the VM. Such a component is a wrapper for the tools used by OpenStack Havana for VM contextualization purposes, i.e., the user data mechanism and the cloud-init package. The *Ceiloesper Contextualization Manager* takes as input the monitoring plugins provided by the infrastructure manager and the developers and automatically injects them in the VM together with the *Ceiloesper Probe*, activating it, and enabling monitoring of metrics at the AAL and ABL layers.

4) Analysis Engine: The *Analysis Engine* is a component of the Ceiloesper Framework that in Figure 2 is illustrated outside the Controller Node. However, there is no constraint prohibiting this module to be also hosted by the Controller Node. The main module of this engine is called *Ceiloesper Engine*. It is based upon Esper, a Java-based CEP (Complex Event Processor) engine for the detection and management of complex events. As depicted in Figure 2, the *Ceiloesper Engine* is configured according to the *SoI* set $\mathscr{S}_{SoI}^{app_{id}}$ for each application running in the cloud environment. The $\mathscr{S}_{SoI}^{app_{id}}$ is translated into the EPL language provide by Esper. The support for defining a Complex Statement *CS* of a *SoI* $S^{app_{id}}$ is currently based on thresholds, or pattern matching for events of higher complexity. The *CS* are implemented by *Listeners* associated with the *Esper Engine*. At the moment two types of listeners (thus, *CS*) have been implemented: detecting levels in excess of a certain threshold, and detecting levels getting again below threshold, by pattern matching. When the *Listerners* evaluate *CS* to $true$, the event Ev associated with the *SoI* $S^{app_{id}}$ will be triggered and published at the AMQP queue, as illustrated in Figure 2. In addition, the measurement values of each monitoring metric m_j associated with a $S^{app_{id}}$ are delivered to the *Analysis Engines* as events. An Esper Engine

itself is not able to collect monitoring information. It must receive the monitoring information in the form of an event that can be later interpreted by the Listerners. This process of converting monitored information into an events happens between the *Ceiloesper Dispatcher* and the *Rest Module* attached to the *Esper Engine*.

5) Configurations of CoIs, SoIs, and Plugins: In this paper we do not discuss in detail the languages and the specific mechanisms to support the automatic configuration of the *CoI*, *SoI*, and monitoring plugins of the Ceiloesper Framework. As we indicate in Figure 2, the dotted lines show the configuration actions necessary in each module. The *SoI* needs to be configured in the Ceiloesper Engine. The *CoI* is associated with three types of modules: The Ceiloesper Probe Pollsters of each VM in each PM, so that only the relevant metrics to the analysis of the *CoI* are retrieved. The Ceiloesper Dispatcher, in order to guarantee that the metrics in a *CoI* will be forwarded to the analysis in the Analysis Engine. The Ceiloesper Monitoring Panel to indicate which metrics should be visualized under the same *CoI*. Finally, it is also necessary to configure the Contextualization Manager in order for the monitoring plugins to be activated in each layer of the cloud environment in order to provide the information for the *SoIs* to be analyzed. Some of the solutions proposed in the related work can be used as baseline technology for automatizing all the required configurations in the Ceiloesper Framework.

IV. Experiments

The goal of our experiments is to demonstrate the effectiveness of the Ceiloesper Framework architecture in analyzing the metrics coming from the infrastructure, detecting specific situations, and triggering management actions by using the 3D Cloud Monitoring model, the *CoI*, and the *SoI* concepts. To achieve this goal, we use a Wordpress application case study. In Section IV-A, we describe a real OpenStack-based testbed and how it has been used to emulate the deployment of the WordPress application in a cloud environment. We also provide details on the tools used to generate synthetic load to stress the WordPress application. In Section IV-B, we discuss the numerical results observed during the experiments.

A. Testbed arrangement

We consider as a base for our experimentation the case study discussed in [2]. However, here we implement an advanced configuration in which a virtual load balancer (provided by OpenStack Neutron Load-Balancing-as-a-Service) is used to redirect requests to two VMs hosting an installation of the WordPress application over the Apache2 Web server. Moreover, on additional VM is exploited to host the MySQL DBMS and the Memchached tool. Fig. 3 shows this testbed arrangement. We exploit three IBM LS21 blade servers as PMs. The corresponding hardware configurations are shown in Fig. 3. On all the PMs, we have installed the Scientific Linux 6.5 distribution. On PM1, we have installed an OpenStack Controller node with Keystone, Glance, Nova (but without Nova-Compute), Neutron, and Horizon services. Among the others, such a PM provide the Neutron LoadBalancer functionalities. Moreover, on PM1 we have installed the Ceiloesper Dispatcher, the Ceiloesper Monitoring Panel, and the Ceiloesper Contestualization Manager. On PM2 and PM3, the Nova-Compute service has been installed in order to be able to instantiate VMs on top of it. The Ceiloesper Probe Pollster has been also installed on them.

On top of PM2, we have instantiated a couple of VMs (VM1 and VM2) where the Apache2 Web server and the Wordpress application are hosted while, on top of PM3, we have instantiated a VM (VM3) where the MySQL database and the Memchaced tool are installed. In all the VMs, we have installed the Ubuntu 14.04 distribution. In commercial cloud environments, a database service is usually purchased with a limited number of supported concurrent connections. For example, Amazon Relational Database Service (Amazon RDS) provides predefined database instances with different maximum number of concurrent connections, e.g., t1.micro (with 34 connections at most), or m1.small (with 150 connections at most). For such a reason, we have initially configured the MySQL DBMS to support a maximum of 50 concurrent connections. On a machine external to the testbed, we installed the Apache Jmeter tool that emulates clients of the WordPress application. The hardware configuration of all the VMs is also described in Fig. 3.

On each PM/VM we have also installed a set of monitoring tools. In particular, we have exploited the standard Havana OpenStack Ceilometer to collect information from the PL (CPU and memory of PM2 and PM3) and VL (vitual CPU and memory consumption of VM1, VM2, and VM3). The AWStats and MyCheckPoint tools have been used to extract data from the AAL, collecting the number of pages per second provided by the Apache Web server and the number of concurrent connections at the MySQL DBMS, respectively. Finally, at the ABLL layer, Apache Jmeter analysis plug-in has been used to collect the percentage of errors experienced by the users while accessing the WordPress Web site and the response time to perform each user action. The Ceiloesper Contextualization Manager automatically injected in all the VMs the Ceiloesper Probe while specific Ceiloesper plugins have been developed and injected to interact with the installed monitoring tools, collect metrics, and provide them to the Ceiloesper Engine through the Ceiloesper Probe. Software configurations and versions are also depicted in Fig. 3.

During the experimentation, the load on the WordPress Web site is synthetically generated using the Apache Jmeter tool. We generated the synthetic traffic according to Table I. We consider five categories of users who perform a different set of operations in a specific order. The users continuously perform their set of actions waiting for a random time between one set and the next one. We defined two different phases (*low* and *high*) in order to stress the system. For each phase, we change the number of users in each category that are concurrently performing a set of operations in the WordPress Web site.

Finally, we defined two *SoI*s for the experiments. The first one is used to detect problems in the configuration of the

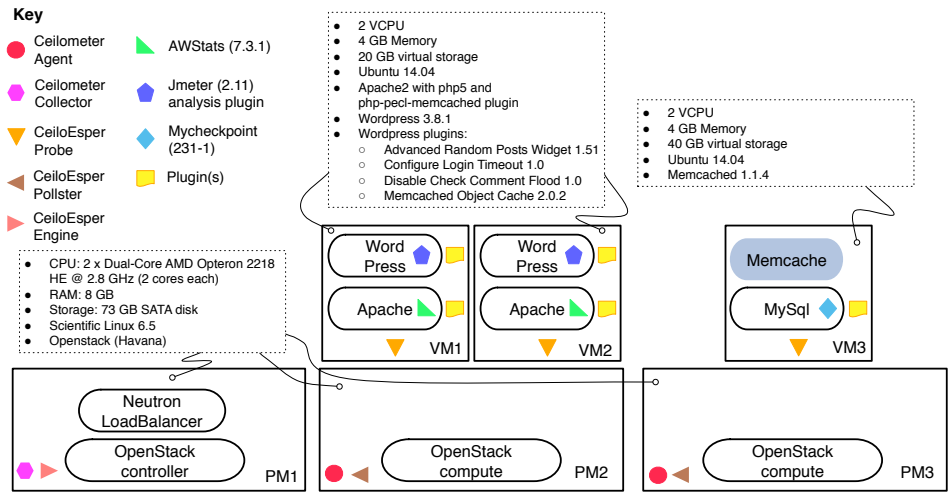


Fig. 3. Testbed arrangement with PMs, VMs, hardware/software configuration, and monitoring tools.

TABLE I
CATEGORIES OF USERS AND POPULATION CHARACTERIZATION FOR THE CONSIDERED LOAD PHASES.

User Type and Description	*Low Load*	*High Load*
Admin creating new post	1	1
Guest reading latest post	10	30
Guest reading random post	10	30
Guest reading latest post and leaving a comment	10	30
Guest reading random post and leaving a comment	10	30

MySQL server in the case of high load expressed as follows:

$$\begin{aligned}(\{&((CPU\% \geq 99\%), 4\ min, -),\\ &((Mem\ (used)\% \geq 60\%), 4\ min, -),\\ &((\#MySQL\ connections \geq 50), 4\ min, -),\\ &((Apache\ pages/sec \leq 1), 4\ min, -),\\ &((WP\ errors\% \geq 10\%), 4\ min, -),\\ &((Response\ time \geq 50sec), 4\ min, -)\},\\ &MySQL\_alarm)\end{aligned} \tag{1}$$

where *MySQL_alarm* is an event about the detection of the situation. The second *SoI* is used to detect that the problematic situation is no more present as described below:

$$\begin{aligned}(\{&((CPU\% \leq 50\%), 4\ min,\ MySQL\_alarm),\\ &((Mem\ (used)\% \leq 40\%), 4\ min,\ MySQL\_alarm),\\ &((\#MySQL\ connections \leq 40), 4\ min,\ MySQL\_alarm),\\ &((Apache\ pages/sec \geq 1), 4\ min,\ MySQL\_alarm),\\ &((WP\ errors\% \leq 1\%), 4\ min,\ MySQL\_alarm),\\ &((Response\ time \leq 1sec), 4\ min,\ MySQL\_alarm)\},\\ &MySQL\_normal)\end{aligned} \tag{2}$$

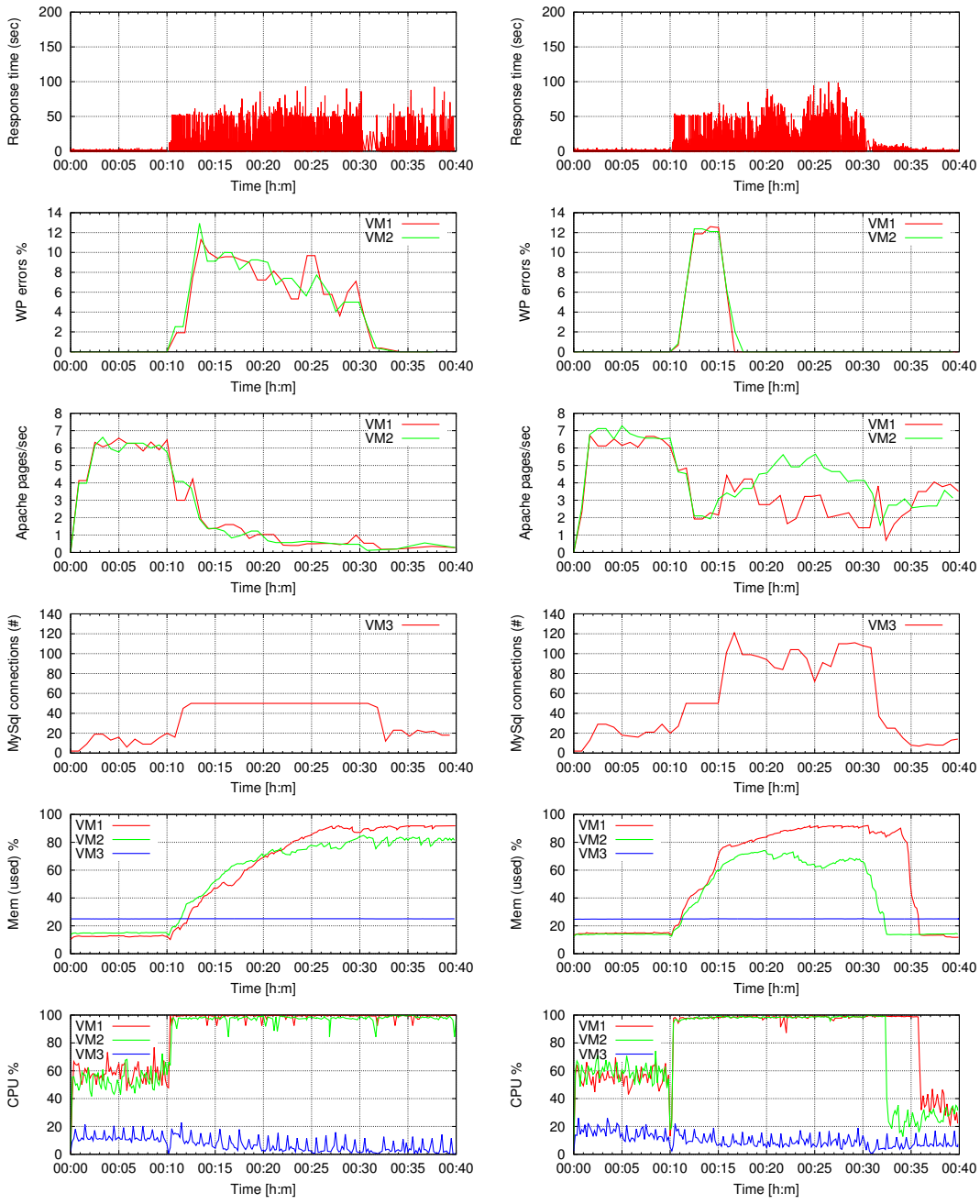
where *MySQL_normal* is the event signaling a normal situation has been detected.

B. Discussion of Results

We conducted experiments for the same case study but with two different scenarios. One is the standard OpenStack installation with basic monitoring solution provided by Ceilometer. This means that no information from multiple layers of the cloud are per default analyzed together, and that there is no real time data analysis and eventual activation of management actions. The graphics in Fig. 4(a) represent the observed metrics (equivalents of a *CoI* configuration) for this scenario without Ceiloesper. The other is the OpenStack installation with the Ceiloesper Framework. Fig. 4(b) shows the metrics of the *CoI* and the results of the changes in the environment for the *SoI* associated with this *CoI*. In this case, Ceiloesper architecture is exploited to monitor a *CoI*, detect a *SoI* in real time, and automatically activate a specific management action. The goal is to demonstrate the need for combining traditional monitoring elements with the power of CEP and to show the effectiveness of the proposed architecture in allowing improvements in the execution of cloud applications.

Figs. 4(a) and 4(b) show the monitored information observed for the WordPress application *CoI*. At the PL and VL layers, we collected the CPU load and the memory consumption. At the AAL layer, we collected the number of concurrent connections at the MySQL DBMS and the number of pages provided by the Apache Web server per second. Finally, at the ABL layer, we measured the percentage of errors experienced by the users while accessing the WordPress Web site and the response time to perform each user action. In the two figures, the first 10 minutes of experimentation from 00:00 to 00:10 represent the *low* load scenario; the interval from 00:10 to 00:30 depicts the *high* load scenario; finally the traffic load is reverted back to the *low* load in the interval from 00:30 to 00:40.

Fig. 4(a) corresponds to the standard OpenStack installation in which no *CoI*, *SoI*, and management actions are configured. In such a figure, we observe that during the *low* load phase



(a) Without Ceiloesper. (b) With Ceiloesper *CoI*, *SoI*, and management action.

Fig. 4. Monitored metrics in the case of (a) standard OpenStack installation, (b) Ceiloesper installation with *CoI*, *SoI*, and management actions configured.

the users experience a response time below 50s and no page errors are detected in the web servers (hosted by VM1 and VM2).The CPU and memory consumption of all VMs from the application are all operating without being overloaded. On the other side, during the *high* load phase, we identify a situation in which the system is overloaded and both the number of errors and the response time increase considerably, as well as the metrics related to the virtual resources of the application. In the previous work, we proved that without using a multi-layer model as proposed in [2], the administrator of the WordPress Web site would have to look in different monitoring graphics (not necessarily clustered together in the same image) and would have to manually relate the information and determine the best management action. However, even if a multi-layer monitoring tool were to be present, if no real time analysis were available the delay in querying the data from the database, analyzing them, and activating a management action to compensate the overload situation could be longer than the time to react to the problem in the best way. Moreover, if traditional auto-scaling mechanisms were configured for this

application, this would mean that typically the metrics from PL and VL layers would be observed in order to activate a traditional resource scaling action (probably increasing the number of virtual CPU associated to VM1 and VM2). However, when we apply our approach, as illustrated in Fig. 4(b), management actions are taken considering information also at the AAL and ABLL layers and a CEP engine is exploited to perform analysis in real time. Considering also information from AAL and ABLL layers, it is possible to inference that the actual problem is related to the maximum number of concurrent connections at the MySQL DBMS (as detailed in [2]). Indeed, in the *high* load this number is reached and the MySQL DBMS starts to reject connections causing the Web Page errors. This also limits the number of pages per second that the Apache server can provide to the users.

Given such an information, the infrastructure manager can define a *SoI* in which if the number of connections to the MySQL database is above a certain threshold for a given time window a management action is triggered changing the database configuration in order to force it to accept a bigger number of connection. In our experimentation, we put the first *SoI* (Eq. (1)) threshold to 50, we set the time window to 4 minutes, and we associate the activation of such a *SoI* a management action changing the maximum number of connections to the MySQL database to 120. According to this, we note that at time 00:15 the *SoI* is detected and the management action is activated. This affects the number of WP errors that quickly decreases to zero. Although the response time does not decrease, we can notice that with the activation of the management action in *SoI* for this case, the application started to serve more connections due to the automatic change triggered by the Ceiloesper Framework. Moreover, even the consumption of CPU and Memory is still high even after the activation of the management action, this consumption is not compromising the capacity of the application to serve the requests. Therefore, this also confirms that no auto-scaling is necessary at that point in time. The second *SoI* (Eq. (2)), as described previously aims at bringing the application back to its initial configuration once there is no more high load of requests. When such a *SoI* is detected, an event is triggered and the management action to restore the maximum number of connections to the database to the original value of 50 is performed. This is the example of a Complex Statement that stores the memory of previous activated events in the analysis of the situation.

V. Conclusion

In the context of cloud monitoring, state of the art solutions fail to provide frameworks in which monitoring in multiple layers and data stream analysis are both in place with the aim of detecting situations and triggering (possibly) multiple management actions. In this paper, we provided an OpenStack-based architecture, the Ceiloesper, combining multi-layer monitoring facilities and real time CEP analysis. We introduced the new concept of the *SoI*, formally describing a combination of monitored metric values that are relevant for the detection of a specific circumstance. Experimental results have been shown demonstrating the need for combining traditional monitoring elements with the power of CEP taking into consideration information coming from all the layers of the cloud environment. Future work will be devoted at automatizing the selection of the best management action in the presence of multiple possible actions ranking them accordingly to QoS and cost parameters. Moreover, more complex scenarios will be considered providing in-depth quantitative analysis of the advantages of our framework in terms of network overhead and responsiveness.

Acknowledgements

Thanks to Nicola Peditto, Carmelo Romeo, and Fabio Verboso for the support on the experimentation. This research received funding from the European Community's Seventh FrameworkProgramme (FP7/2007-2013) under grant agreement Nr. 610802 (CloudWave).

References

[1] C. C. Marquezan, F. Wessling, A. Metzger, K. Pohl, C. Woods, and K. Wallbom, "Towards exploiting the full adaptation potential of cloud applications," in *In Proceedings of 6th International Workshop on Principles of Engineering Service-Oriented and Cloud System (PESOS 2014) in conjunction with IEEE and ACM ICSE 2014*, 2014, p. 10.

[2] C. C. Marquezan, D. Bruneo, F. Longo, F. W. A. Metzger, and A. Puliafito, "3-D Cloud Monitoring: Enabling Effective Cloud Infrastructure and Application Management," in *Network and Service Management (CNSM), 2014 International Conference on*, 2014, p. 9, to Appear.

[3] M. Carvalho, R. Esteves, G. Rodrigues, C. C. Marquezan, L. Z. Granville, and L. M. R. Tarouco, "Efficient configuration of monitoring slices for cloud platform administrators," in *In Proceedings of 19th IEEE Symposium on Computers and Communications (ISCC 2014)*, 2014, p. 7.

[4] OpenStack Community, "Ceilometer," 2014, Available at: https://wiki.openstack.org/wiki/Ceilometer.

[5] D. Trihinas, G. Pallis, and M. Dikaiakos, "Jcatascopia: Monitoring elastically adaptive applications in the cloud," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014, pp. 226–235.

[6] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "Gmone: A complete approach to cloud monitoring," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2026 – 2040, 2013.

[7] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, and G. Antoniu, "Jetstream: Enabling high performance event streaming across cloud data-centers," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: ACM, 2014, pp. 23–34. [Online]. Available: http://doi.acm.org/10.1145/2611286.2611298

[8] A. Mdhaffar, R. Ben Halima, M. Jmaiel, and B. Freisleben, "A dynamic complex event processing architecture for cloud monitoring and analysis," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2, Dec 2013, pp. 270–275.

[9] O. Saleh, F. Gropengiesser, H. Betz, W. Mandarawi, and K.-U. Sattler, "Monitoring and autoscaling iaas clouds: A case for complex event processing on data streams," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th Int. Conf. on*, Dec 2013, pp. 387–392.

[10] I. Ari, E. Olmezogullari, and O. Celebi, "Data stream analytics and mining in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th Int. Conf. on*, Dec 2012, pp. 857–862.

[11] R. Esteves, L. Z. Granville, M. F. Zhani, and R. Boutaba, "Evaluating allocation paradigms for multi-objective adaptive provisioning in virtualized networks," in *In Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*, 2014, p. 9.

[12] P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar, "Application-level performance monitoring of cloud services based on the complex event processing paradigm," in *Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE Int. Conf. on*, 2012, pp. 1–8.