A Practical Approach to Easily Monitoring and Managing IaaS Environments

Paolo Bellavista, Carlo Giannelli, and Massimiliano Mattetti
CIRI-ICT – University of Bologna - ITALY
{paolo.bellavista, carlo.giannelli, massimiliano.mattetti}@unibo.it

Abstract — **Private cloud computing has been recently pushed as a promising solution to more efficiently exploit available hardware equipment and quickly reconfigure software components depending on the current load. However, private cloud computing, and in particular the Infrastructure as a Service (IaaS) model, has been primarily adopted only by large companies, at least so far. In fact, Small and Medium Enterprises (SMEs) usually have difficulties in adopting the private computing paradigm, either since they are not able to afford the cost of off-the-shelf proprietary solutions or they do not have the know-how required to adapt open-source solutions to their own requirements. The paper presents our novel framework for Easy Monitoring and Management of IaaS (EMMI) solutions, with the main objective of making easier the adoption of the private cloud paradigm. The EMMI framework is based on open-source software components, i.e., OpenStack for IaaS management, Collectd for distributed system monitoring, and Apache jk for load balancing, allowing to adopt the IaaS model easily and in a cost effective manner. The reported performance results demonstrate that the EMMI framework efficiently supports two primary features of the IaaS model, i.e., failover and scale-out, promptly identifying crucial events and autonomously taking suitable countermeasures.**

Keywords - Cloud Computing; Private IaaS; Failover Management; Scale-out Management; OpenStack.

I. INTRODUCTION

In recent years, academic and industry researchers have strongly pushed for the adoption of private cloud computing solutions in enterprise environments. In particular, the Infrastructure as a Service (IaaS) model has been widely advertised as the most promising solution to fully exploit available hardware capabilities, promptly react to traffic/computational load variations, and efficiently reduce energy consumption. In fact, IT departments can exploit the IaaS model to create one virtual homogeneous data center on top of multiple heterogeneous physical nodes. In this manner, not only services can be easily migrated among nodes, e.g., to lower the computational load of a physical host, but also the management complexity can be reduced, e.g., since virtual machines can be cloned/restored more easily if compared with traditional physical hosts.

However, the adoption of IaaS solutions is currently relatively limited to large companies managing high amounts of physical hosts and exploiting expensive proprietary solutions, mainly offered by cloud computing big competitors such as IBM and Oracle. Instead, Small Medium Enterprises (SMEs) rarely move towards IaaS solutions since software and hardware infrastructure solutions designed for large companies do not usually fit their requirements, both in terms of cost and complexity. On the one hand, they cannot afford the cost of completely renewing their infrastructure while they prefer to reuse their own hardware capabilities. On the other hand, while the migration towards cloud solutions based on open-source platforms may reduce costs, it requires additional efforts and know-how not only to virtualize offered services, but also to configure/manage the private cloud. In fact, open-source private cloud platforms support basic features (such as virtual machine creation and migration), while the integrated monitoring of resources and services is not supported.

The objective of the paper is twofold. First of all, it aims at demonstrating that even SME IT departments can easily and effortlessly deploy private cloud computing solutions and adopt the IaaS model to run their own services. In particular, the paper reports about the practical experience of creating and managing a small data center on top of off-the-shelf physical hosts exploiting open-source frameworks: OpenStack, widely adopted and sufficiently mature IaaS platform, Collectd, a scalable performance monitoring system suitable for both physical and virtual machines, and Apache jk, a widely spread HTTP load balancer. The proposed solution has been developed within the Center for Industrial Research on Information and Communication Technologies (CIRI-ICT) of the University of Bologna, Italy. CIRI-ICT has the objective to promote sustainable innovation within industries and SMEs offering technological support, knowledge transfer, and business development (further details at http://www.ciri-ict.unibo.it/en).

Secondly, the paper describes our framework for Easy Monitoring and Management of IaaS (EMMI) solutions, originally designed and implemented for the smart management of services offered on top of private clouds. EMMI exploits OpenStack and Collectd to manage and monitor physical and virtual machines and Apache jk to perform load balancing, autonomously performing failover and scale-out procedures. EMMI has been tested and verified in deployment scenarios where Web applications are executed; the reported performance results demonstrate the effectiveness of the proposed solution in dynamically and autonomously managing private clouds.

The rest of the paper is organized as follows. Section II briefly introduces exploited open-source tools, i.e., OpenStack, Collectd, and Apache jk. Section III details the EMMI framework, underling its architecture and offered API. Section IV reports about achieved performance results when exploiting the proposed solution in case of failover and scale-out. Related work and conclusion sections end the paper.

II. OPENSTACK, COLLECTD, AND APACHE JK

The proposed solution exploits open-source software frameworks to support the IaaS model, monitor active services, and dynamically reconfigure them. In this manner

SMEs can effectively reduce their cost of initial investment when moving towards private cloud solutions.

OpenStack is the most mature and adopted open-source IaaS platform [1]. The first version has been jointly developed by Rackspace Cloud and NASA in 2010, while it is currently supported by the OpenStack Foundation. The OpenStack architecture (Essex release) is composed of five primary components:

1) OpenStack Compute, supporting the creation and management of virtual machines;
2) OpenStack Storage, to store objects and blocks in a redundant, scalable and persistent manner;
3) OpenStack Dashboard, a graphical interface to monitor and manage the cloud;
4) OpenStack Identity Service, to create/modify users and manage access rules;
5) OpenStack Image Service, to copy, store, and discover both virtual disk and virtual machine images.

In particular, the OpenStack Compute component provides five main services:

a) nova-compute, interacting with the libvirt library [2] to manage virtual machine life cycle in an hypervisor-independent manner;
b) nova-volume, to create and de/attach persistent volumes to nova-compute instances;
c) nova-network, manipulating nova-compute network configurations, e.g., adding/removing network interfaces;
d) nova-schedule, dispatching virtual machine creation requests to nova-compute instances;
e) nova-api, providing access to features offered by other services via API, supporting both OpenStack own RESTful methods and Amazon EC2 SOAP-based ones.

It is worth noting that OpenStack fully supports the creation, instantiation, migration, and removal of virtual machines. In fact, administrators can exploit the OpenStack Dashboard to allocate available virtual machine instances to nova-compute nodes. In particular, the OpenStack Dashboard can be useful in the initial deployment phase to setup the environment and at service provisioning time to monitor the status of available virtual machines. However, OpenStack does not provide any feature to quantitatively evaluate the current state of physical and virtual machines, e.g., CPU and RAM usage, traffic load, and available disk space. Moreover, it does not provide any feature to autonomously identify critical events, e.g., overloaded physical hosts, and to appropriately take countermeasures, e.g., dynamically migrating virtual machines to less loaded physical hosts.

As better detailed in the following section, we have developed a novel framework to dynamically monitor and manage services offered on top of virtual machines. To this purpose, we exploit the Collectd framework to gather information about virtual machines and physical hosts [3]. Collectd is a Linux daemon based on a plug-in architecture enabling the flexible and scalable monitoring of distributed computing environments. In particular, we deploy Collectd on physical machines hosting nova-compute and nova-network modules to monitor performance of network interfaces (delivered packets, traffic throughput, transmission failures), disks (read/write throughput, available disk space), and CPU/RAM workload. Moreover, Collectd supports a libvirt plugin allowing to gather the same information (with exception of the RAM) on guest virtual machines without any modification of virtual machine themselves. Thus, it is possible to monitor not only the current status of physical hosts, but also to discriminate among the workload of each virtual machine. Finally, Collectd allows to periodically spread gathered information via unicast/multicast UDP packets, eventually also with the capability of notifying events.

Finally, we exploit Apache HTTP server and Tomcat to offer Web servers. Each virtual machine runs only one Web server, while different virtual machines may run the same Web server to increase performance and reliability. We have exploited Apache jk as HTTP load balancer [4]: instances of the same server are split in different clusters with sticky session management (same tomcat domain), i.e., requests from the same client are always forwarded to the same cluster, while instances within the same cluster share the client session state. Note that we adopt the general rule of replicating the same Web server in multiple clusters with few replicas in each cluster. In this manner, we support state replication while limiting the time required to add a server replica, which would greatly increase in case of clusters with many replicas.

Figure 1 shows an example of the proposed deployment environment with four physical hosts (Host A/D), eight virtual machines providing three Web servers (Server 1/3) clustered in four groups of two instances each one. In particular, Server 1 is replicated four times in two clusters, while Server 2 and Server 3 two times in one cluster each one.

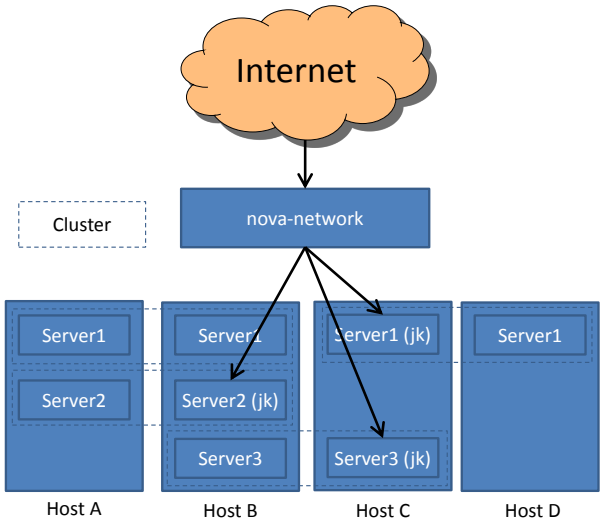

Figure 1. Deployment environment example.

III. THE EMMI FRAMEWORK

OpenStack, Collectd, and Apache jk tools provide basic features to run HTTP servers in a private IaaS cloud in a replicated manner. However, at the current stage of OpenStack development, it is required that a human operator monitors and configures these tools. For instance, if Server 2 receives

a burst of requests greater than the amount Host A and Host B are able to serve, replies could be considerably delayed. In this case, an administrator should actively identify the issue and manually start new instances of Server 2 on other physical hosts, also reconfiguring Apache jk to forward requests to the new instances.

To make easier the management of private cloud solutions, we have designed and implemented the EMMI framework. EMMI takes care of monitoring and managing the private cloud, taking appropriate actions in case of unexpected behaviors. Leveraging system administrators from the burden of controlling the private cloud, the adoption of the IaaS model can become suitable even for SME IT departments with limited background in cloud computing.

In particular, EMMI supports two primary features: failover and scale-out. The former is in charge of detecting server failures and automatically restore them, e.g., instantiating new virtual machines. The latter periodically monitors the behavior of the system migrating servers in case the Quality of Service (QoS) goes below fixed thresholds, e.g., instantiating and configuring a new virtual machine instance to better serve requests to an overloaded Web server.

In the following, we present the architecture and API of the EMMI framework, our failover and scale-out mechanisms, and the metrics implemented to select the most appropriate physical host to activate a new virtual machine on and evaluate the current QoS.

A. The EMMI Framework Architecture and API

The EMMI framework is composed of four primary components (Figure 2): Cloud Controller, Cloud Monitor, Load Balancer Manager, and Web Server Manager.

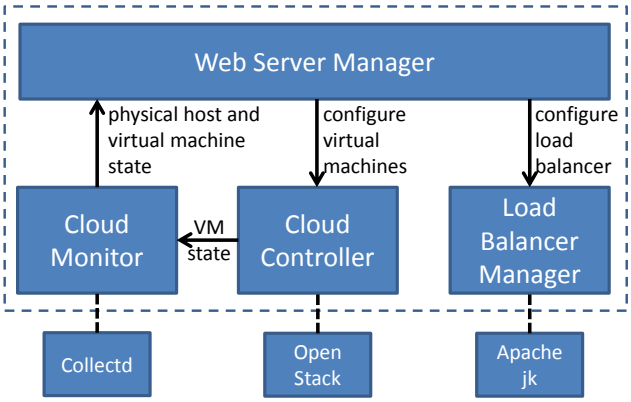

Figure 2. EMMI framework architecture.

Cloud Controller encapsulates OpenStack RESTful API and offers an API independent from OpenStack, thus providing a homogeneous and simpler access to the underlying cloud platform. It exploits the novaclient sub-component provided by the OpenStack community to easily invoke OpenStack RESTful API via Python. For instance, novaclient provides methods to create a new virtual machine based on an image or to retrieve the set of active virtual machines. On top of novaclient, we have developed the Cloud Wrapper sub-component to provide higher level API. For instance:

- `newVM(vmName, vmImageName, vmSize)`, creating a new image of size `vmSize`, e.g., tiny, small or medium, with name `vmName`, exploiting the image `vmImageName`;
- `migrateVM(vmId, destHost)`, migrating the virtual machine with id `vmId` from its current location to the host `destHost`.

These methods greatly facilitate the administration of the cloud, not only hiding low-level details, e.g., referring to images with meaningful names instead of numerical ids, but also managing most common errors, e.g., retrying to start a virtual machine in case it fails at boot. Moreover, to support other cloud platforms it is only required to provide an appropriate Cloud Wrapper implementation, while high-level Cloud Monitor API keep their validity. In this manner we also improve the interoperability among heterogeneous environments; in fact, the EMMI framework can be easily adopted on top of other cloud computing platforms simply implementing appropriate versions of the Cloud Wrapper sub-component.

Cloud Monitor gathers data from Cloud Control and nova-compute nodes via Collectd providing to upper layers a uniform view of the environment. To this purpose, it encapsulates gathered data in a dictionary-based data model allowing to quickly retrieve a wide range of information of both physical and virtual machines, ranging from the amount of available interfaces and traffic load, to the available RAM, free mass storage, and CPU usage statistics. Note that the description of each host and virtual machine requires about 15KB and 5KB respectively.

Load Balancer Manager provides the capability of dynamically modifying the load balancer in a programmatic way. It directly interacts with Apache jk to gather the state of the load balancer and to modify its configuration. Moreover, prior to instantiate a new virtual machine, it injects a configuration file appropriately customized in relation to the current network environment. The configuration file defines the name of the cluster for Tomcat workers and specifies the URL of the Web app, the list of the workers, and the IP address of the EMMI host allowed to change the load balancer configuration for Apache jk. Note that our Load Balancer Manager can instantiate and configure virtual machines containing either only the Web server (appropriately configured to join a cluster) or the Web server together with Apache jk (appropriately configured to forward requests to active replicas).

Web Server Manager is in charge of instantiating server replicas and creating/migrating/suspending/resuming/destroying virtual machines in relation to the current state of the cloud. On the one hand, it exploits Cloud Monitor to periodically check the state of the cloud. On the other hand, it encapsulates the application logic exploited to trigger virtual machine migrations (additional details in the following subsections).

Let us stress that EMMI components are designed and implemented to be highly configurable and reusable in many different scenarios. In particular, Cloud Monitor and Cloud Manager provide API that can be exploited when offering other services also different from Web-based applications.

For instance, we are currently evolving our prototype to offer Dropbox-like and distributed DBMS services on top of Cloud Monitor and Cloud Manager. Instead, Load Balancer Manager is specifically designed to manage HTTP-based services.

Finally, Web Server Manager is the only component IT administrators have to configure via a two-phase process: first of all, they have to prepare a virtual machine containing the desired Web service in one of the many standard formats OpenStack is able to accept. For instance, in case of KVM hypervisor, supported formats are raw, QEMU qcow2, VirtualBox VDI, VMWare VMDK, and Amazon AMI. Secondly, the IT administrator has to provide an XML document specifying the desired configuration (in a future version of our framework we will provide a GUI to hide the manual creation of the XML document from the perspective of final users). For instance, the XML document below specifies that the client with id "a1" has priority 10, there is only one service with configuration id "c1", the service replica with Apache jk is deployed on a virtual machine with size "small" (2048MB RAM, 2 vCPU) and public IP address "publicIP", while virtual machines providing only the Web server must be of size "tiny" (512MB RAM, 1 vCPU). Once the virtual machine is ready and the XML file configured, the EMMI framework autonomously manages the virtual machine, also by reconfiguring the load balancer appropriately.

```
<?xml version='1.0' encoding='ASCII'?>
<clients    xmlns:xsi    =    http://www.w3.org/2001
/XMLSchema-instance xsi:noNamespaceSchemaLocation =
"cloud_descriptor.xsd"     xsi:schemaLocation     =
"cloud_conf_file/cloud_descriptor.xsd">
  <client id="a1">
    <priority>10</priority >
    <configurations>
      <configuration id="c1">
        <loadBalancer>
          <flavorName>m1.small</flavorName>
          <sshKey>ssh_key</sshKey>
           <publicIp>publicIP</publicIp>
          <webApp>myApp/*</webApp>
          <workerServer>
            <imageName>WebServer</imageName>
            <flavorName>m1.tiny</flavorName>
            <numDomains>1</numDomains>
            <multicastIp>228.0.0.4</multicastIp>
          </workerServer>
        </loadBalancer>
      </configuration>
    </configurations>
  </client>
</clients>
```

Figure 3. Example of EMMI XML configuration document.

B. Web Server Manager QoS Features

Our Web Server Manager currently provides two main features:

- failover, periodically checking the state of physical and virtual machines and instantiating new virtual machines in case of failures;
- scale-out, verifying the QoS level and adding new virtual machine instances in case it deteriorates too much.

To support the failover feature, Cloud Monitor gathers data from physical and virtual machines every 2.5s, tagging a machine as failed if it does not receive data for 10s (the period is reconfigurable at run-time). In this case, Web Server Manager interacts with Load Balancer Manager to remove failed unavailable virtual machines from the set of active workers; thus, Apache jk replicas forward HTTP requests only to available workers.

While identifying failed machines is relatively straightforward, selecting where instantiating new virtual machines deserves more attention. In this case, the framework not only has to gather data to provide a clear picture of the current state of physical and virtual machines, but has also to support a mechanism to decide which physical host is the most suitable one to run a new virtual machine. In fact, starting a new virtual machine in an almost overloaded physical host may further lower the global QoS. To this purpose, Web Server Manager considers the following parameters for each physical host *i*:

- FreeRAM_i, the available RAM;
- IdleCPU_i, the percentage of CPU idle time;
- FreeDisk_i, the available mass storage;
- $\text{Read/WriteDiskThroughput}_i$, the performance of read/write operations on mass storage;
- $\#\text{VM}_i$, the amount of already active virtual machines.

Then, it selects the least loaded physical host, i.e., the host with highest $Performance_i$ value computed as

$$Performance_i = \frac{FreeRAM_i \times w_{fr}}{t_{fr}} + \frac{IdleCPU_i \times w_{ic}}{t_{ic}} + \frac{FreeDisk_i \times w_{fd}}{t_{fd}} - \frac{ReadDiskThroughput_i \times w_{rdt}}{t_{rdt}} - \frac{WriteDiskThroughput_i \times w_{wdt}}{t_{wdt}} - \frac{\#VM \times w_{vn}}{t_{vn}}$$

where w parameters are weights in the [0, 1] range that IT administrators can modify at run-time and t parameters are the summation of data gathered from every physical host, e.g., $t_{fr} = \sum_i FreeRAM_i$.

Finally, Web Server Manager invokes the `newVM()` method of Cloud Manager to instantiate the new virtual machine and interacts with Load Balancer Manager to configure Apache jk.

To support the scale-out feature, it is also required to identify whether services are offered with sufficiently high QoS. To this purpose, Web Server Manager activates two different alerts:

a) VM-CPU, triggered whenever the CPU of a virtual machine is higher than a threshold (default 85%) for a given period time (default 10s);
b) VM-Traffic, triggered whenever the outgoing traffic throughput of a virtual machine is higher than a threshold (default 10MB/s) for a given period (default 10s).

Note that VM-CPU and VM-Traffic alerts are typically triggered in case virtual machines are not able to satisfy incom-

ing requests. Based on virtual machine CPU load and traffic throughput, we can infer the state of the service inside the virtual machines themselves. Thus, to a certain degree of coarse-grained approximation, it is possible to gather information about the state of Web servers without directly interacting with them, making easier the deployment of new services on top of the cloud platform (we do not need to modify virtual machine images offering the service).

Once an alert has been triggered, Web Server Manager selects a host to instantiate a new virtual machine on and reconfigures Load Balancer Manager as already described in the failover feature. In addition, we plan also to support scale-up, increasing hardware capabilities of running virtual machines instead of instantiating new ones. However, scale-up is not suitable to quickly react to unexpected traffic bursts, since it requires to shut down the targeted virtual machine, modify its hardware characteristics, and then reboot it, thus further lowering the overall available resources for a brief time period. Instead, we aim at gathering daily statistics of virtual machine loads and, in case specific traffic patterns are detected, to proactively resize hardware capabilities of virtual machines.

In addition, Web Server Manager monitors physical hosts to identify possible overloads, trying to limit the consumption of virtual machines that could potentially harm other co-hosted services. In particular, it considers the following three alerts:

- H-RAM: whenever the free RAM of a host lowers below a threshold (default 15%), it tries to migrate one of its virtual machines with lowest priority to another host, possibly suspending it if the migration is not possible, e.g., there is not enough free RAM in other physical hosts;
- H-Disk: similar to H-RAM but considering the available mass storage;
- H-CPU; similar to H-RAM but activated whenever the CPU idle time goes below a threshold (10% default) for more than a configurable period time (15s default).

In this manner we are able to identify possible issues even if virtual machines do not trigger scale-out alerts.

IV. Test-bed Evaluation

We have experimentally validated our EMMI framework in a test-bed composed of five physical hosts (Figure 4):

- HP1/2 with high performance (8GB RAM, quad-core with hyper-threading technology @3.2GHz). HP1 runs the EMMI framework and main OpenStack components in charge of monitoring and controlling the system, HP2 runs nova-network, thus performing as unique entry-point from the Internet;
- LP1/2/3 with low performance (4GB RAM, dual-core @2.6GHz), running the Collectd client and OpenStack nova-compute component to host virtual machine instances. Virtual machines running only the server have 512MB of RAM and 1 vCPU, with also Apache jk 2048MB of RAM and 2 vCPU.

We have tested the failover feature when deploying two different services, each one in a cluster, and each one with two replicas (Server 1 on LP1 and LP2, Server 2 on LP2 and LP3). We have abruptly disconnected LP3 (not containing the virtual machine with Apache jk) from the network: the EMMI framework keeps about 43.2s to instantiate a new replica of Server 2 on LP1: 5.2s to identify LP3 failure, 0.3s to select the new host, 0.1s to reconfigure Apache jk, and the rest to boot the virtual machine. It is worth noting that most of the reconfiguration latency is due to failure identification and virtual machine booting, while the delay due to EMMI framework management is considerably limited.

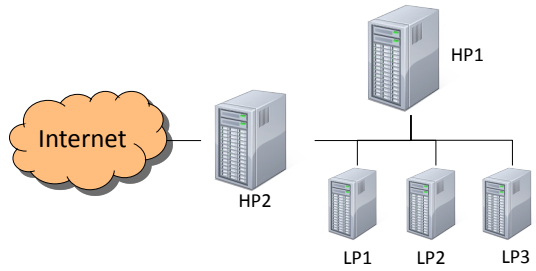

Figure 4. Test-bed deployment.

We have also tested the scale-out feature, exploiting Apache JMeter [5] to emulate multiple clients performing HTTP requests at the same time. Web servers reply to HTTP requests sending an HTML page with an image of 170KB, roughly half the average page size of the Web [6]. JMeter has been configured to perform up to 400 request/s during 330s, following the pattern in Figure 5. When not activating the scale-out feature, response times quickly degrade at an average of 10.2s (see Figure 6); moreover, only 86.2% of the requests are correctly served. Instead, when the scale-out feature is active, 98.2% of requests are successfully served and the average response time is 43% lower (about 5.8s). Finally, note that some requests are queued and correctly served at the end of the experiment.

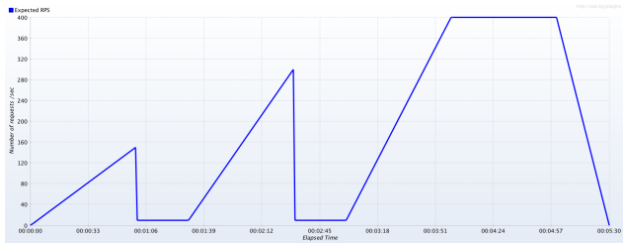

Figure 5. Traffic pattern of emulated HTTP requests (x-axis test time from 0 to 330s, y-axis request/s from 0 to 400).

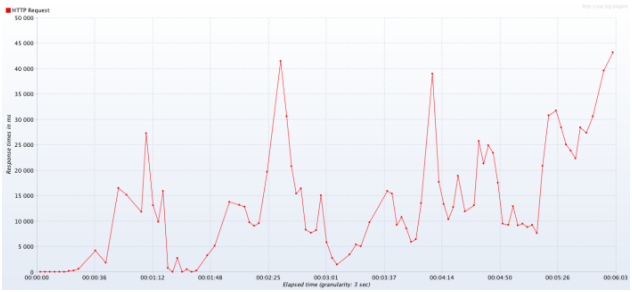

Figure 6. Achieved performance without scale-out (x-axis test time from 0 to 330s, y-axis response time from 0s to 50s).

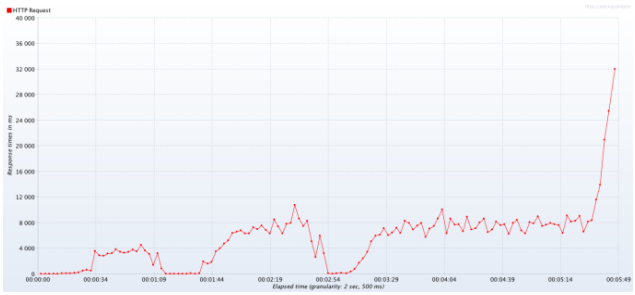

Figure 7. Achieved performance with scale-out (x-axis test time from 0 to 330s, y-axis response time from 0s to 40s).

V. Related Work

In the last years, industrial and academia researchers have demonstrated growing interest in cloud computing. While the cloud computing market is currently dominated by solutions targeted for large companies [7, 8], the relatively mature state of open-source cloud platforms (such as OpenStack) is pushing for alternative solutions fitting SME requirements. For instance, [9] proposes a solution to support SMEs in the adoption of hybrid cloud environments, by coupling benefits from both private and public cloud models. Instead, [10] focuses on the deployment of applications on top of OpenStack, monitoring and controlling the environment to satisfy management objectives. The proposed solution has been tested on both Web-based and CPU-intensive applications, demonstrating to be able to reduce resource consumption. However, it does not support the dynamic management of clusters based on virtual machines and its QoS assessment is mainly based on CPU consumption. Note that the importance of easily supporting monitoring, scaling, and load balancing in private cloud environments based on open-source solutions is also demonstrated by current efforts of the Eucalyptus initiative [11] to provide such features in their upcoming 3.3 release.

Other solutions represent interesting evolutions of the cloud computing model, fitting SME expectations of easy management of storage and network resources. For example, [12] also considers the network layer, supporting the dynamic re/configuration of virtual networks; in this manner, SME IT departments can improve the security of virtualized environments, e.g., by isolating virtual machines related to different services. Instead, [13] is focused on virtual storage capabilities of OpenStack, with particular attention to scalability and performance of the virtual storage infrastructure. We are currently evolving the EMMI framework to additionally integrate network and storage virtualized resources, made easier by the modular and reconfigurable design of our framework.

VI. Conclusions

The proposed EMMI framework represents a practical approach to easily adopt the IaaS model based on open-source tools, i.e., the OpenStack platform, the Collectd monitoring framework, and the Apache jk load balancer. Based on EMMI, we believe that even SME IT departments with limited know-how on cloud computing can adopt the IaaS model in a cost effective way, both to fully exploit available hardware equipment and efficiently support the adaptive reconfiguration of software components. In addition, the EMMI modular architecture allows easily integrating novel cloud computing platforms and managing heterogeneous services, thus widening its scope beyond the provisioning of Web-based applications. Finally, the reported performance results demonstrate the effectiveness of our metrics in quickly adapting and reconfiguring physical hosts and virtual machines in relation to the current state of the environment.

Also pushed by the promising performance results achieved so far, we are further developing the EMMI framework to widen the set of its supported features. In particular, we are currently developing Dropbox-like and distributed DBMS services on top of EMMI. In addition, we plan to develop a federated deployment environment, available for experimentations of SMEs, where remotely located cloud environments can easily interact and cooperate.

Acknowledgment

Funded by POR FESR Emilia-Romagna 2007-2013.

References

[1] "OpenStack, Open source software for building private and public clouds", http://www.openstack.org/ (last visited on March 2013).

[2] "libvirt virtualization API", http://libvirt.org/ (last visited on March 2013).

[3] "Collectd – The system statistics collection daemon", http://collectd.org/ (last visited on March 2013).

[4] "Apache jk Load Balancer", http://tomcat.apache.org/connectors-doc/generic_howto/loadbalancers.html (last visited on March 2013).

[5] "JMeter", http://jmeter.apache.org/ (last visited on March 2013).

[6] S. Ramachandran, "Web metrics: Size and number of resources", https://developers.google.com/speed/articles/web-metrics (last visited on March 2013)

[7] "IBMSmartCloud: Advanced, enterprise-ready IaaS", http://www.ibm.com/cloud-computing/us/en/iaas.html (last visited on March 2013).

[8] "Oracle Infrastructure as a Service: Infrastructure as a Service On Premise with Capacity on Demand", http://www.oracle.com/us/products/engineered-systems/iaas/overview/index.html (last visited on March 2013).

[9] S. Van Hoecke, T. Waterbley, J. Devos, T. Deneut, J. De Gelas, "Efficient management of hybrid clouds", Second Int. Conf. on Cloud Computing, GRIDs, and Virtualization, pp. 167-172, 2011.

[10] F. Wuhib, R. Stadler, H. Lindgren, "Dynamic Resource Allocation with Management Objectives: Implementation for an OpenStack Cloud", Int. Conf. on Network and Service Manag. (CNSM), 2012.

[11] "Eucalyptus Cloud", http://www.eucalyptus.com/ (last visited on March 2013).

[12] P. Ruth, A. Mandal, Y. Xin, I. Baldine, C. Heerman, J. Chase, "Dynamic network provisioning for data intensive applications in the cloud", 8th IEEE Int. Conf. on E-Science, pp. 1-2, 2012.

[13] S. Toor, R. Toebbicke, M. Zotes Resines, S. Holmgren, "Investigating an Open Source Cloud Storage Infrastructure for CERN-specific Data Analysis", 7th IEEE Int. Conf. on Networking, Architecture and Storage (NAS), pp. 84-88, 2012.