A Practical Approach to Easily Monitoring and Managing IaaS Environments

Paolo Bellavista
Carlo Giannelli
Massimiliano Mattetti

07.07.2013 – MoCS '13

CIRI-ICT, Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna Italy
Center for Industrial Research on ICT of the University of Bologna

Main goals:
- promoting the technology transfer between the University and regional Small/Medium Enterprises (SMEs)
- coordinating and developing industrial research projects
- disseminating research results
- offering knowledge transfer service in the ICT field while meeting the needs of industry and sustainable development

More info at:
http://www.ciri-ict.unibo.it/en/
 Agenda

**IaaS opportunities for SME**
- Cloud Computing solutions vs. SMEs requirements

**EMMI framework**
- Architecture and features
- QoS management

**Test-bed Evaluation**
- Deployment environment
- Failover and scale-out test
IaaS opportunities for SMEs (1)

The cloud computing market stack

We have investigated the needs of our SMEs and the main emerging interest is about a possible on-premise implementation of a private cloud infrastructure.

Solutions of interest for a private cloud infrastructure
Infrastructure as a Service (IaaS)
- fully exploit available hardware capabilities
- promptly react to traffic/computational load variations
- efficiently reduce energy consumption

Costs and complexity of both software and hardware infrastructure have limited the adoption of off-the-shelf proprietary solutions

SMEs may have troubles when adapting open-source solutions to their own requirements, e.g., because of the limited know-how in cloud computing
IaaS opportunities for SMEs (3)

Our main goal: demonstrating that private cloud IaaS solutions can be easily and effortlessly adopted.

To fill the gap between open-source solutions and business requirements we developed high level functionalities for the smart management of services on top of basic mechanisms offered by IaaS platforms.
The Easy Monitoring and Management of IaaS (EMMI) framework
- based on open-source software components
- designed and implemented for the smart management of services offered on top of private clouds

Manages virtual machines running the services

Collects monitoring data of physical servers and virtual machines
EMMI Framework (2)

Four main components
Three-layer architecture

Service Specific Layer

General Purpose Layer

Open Source Tools

Cloud Monitor
Cloud Controller
Load Balancer Manager

Web Server Manager

Configure load balancer
Configure virtual machines
Configure virtual machine state

physical host and virtual machine state

Collectd
Open Stack
Apache jk

VM state
**EMMI Framework (3)**

**OpenStack** for IaaS management

**Collectd** for distributed system monitoring

---

**OpenStack**

- Compute
- Networking
- Storage

**Collectd**

- Network
- SNMP
- other plugins
- exec plugin
- rrdtool plugin
- csv plugin
Web Server as type of services managed by the framework

- Apache Tomcat 7 as application server
- Apache mod_jk for load balancing
**Cloud Controller**

- **Encapsulates** OpenStack RESTful **API**
- Offers API independent from OpenStack
- Exploits the *novaclient* sub-component provided by the OpenStack community
- The framework can be reused with other cloud platforms (e.g. Eucalyptus, CloudStack) changing the sub-module that encapsulates OpenStack APIs
Gathers data from Cloud Controller (virtual machine states, e.g., active, suspended) and nova-compute nodes (network traffic load, available RAM, free mass storage, and CPU usage) via Collectd

Provides to upper layers a uniform view of the environment

Encapsulates gathered data in a dictionary-based data model allowing to quickly retrieve a wide range of information of both physical and virtual machines
Provides the capability of **dynamically modifying the load balancer** in a programmatic way

**Interacts with Apache mod_jk via HTTP requests** to gather the state of the load balancer and to modify its configuration

** Appropriately customizes the configuration files** that are injected into virtual machines
Creates/migrates/suspends/resumes/destroys virtual machines in relation to the current state of the cloud

Exploits Cloud Monitor to periodically check the state of the cloud

Encapsulates the application logic that implements the failover and scale-out features

To be configured by IT administrators
1. Prepare a virtual machine containing the desired Web application in one of the many standard formats supported by OpenStack

2. Provide an XML document specifying the desired configuration

```xml
<clients xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="cloud_descriptor.xsd"
         xsi:schemaLocation="cloud_descriptor.xsd">
  <client id="a1">
    <level>10</level>
    <configurations>
      <configuration id="c1">
        <loadBalancer>
          <flavorName>m1.small</flavorName>
          <sshKey>ssh_key</sshKey>
          <publicIp>PUBLIC IP</publicIp>
          <workerServer>
            <imageName>Worker</imageName>
            <flavorName>m1.tiny</flavorName>
            <numDomains>1</numDomains>
            <multicastIp>228.0.0.4</multicastIp>
          </workerServer>
        </loadBalancer>
      </configuration>
    </configurations>
  </client>
</clients>
```
Starting a new virtual machine in an almost overloaded physical host may further lower the global QoS

Our scheduling algorithm considers 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
- $Read/WriteDiskThroughput_i$, the performance of read/write operations on mass storage
- $\#VM_i$, the amount of already active virtual machines
Virtual Machine Scheduler (2)

Web Server Manager selects the **least loaded physical host**, i.e., the host with highest $\text{Performance}_i$

$$\text{Performance}_i = \frac{\text{FreeRAM}_i \times w_{fr}}{t_{fr}} + \frac{\text{IdleCPU}_i \times w_{ic}}{t_{ic}} + \frac{\text{FreeDisk}_i \times w_{fd}}{t_{fd}}$$

$$- \frac{\text{ReadDiskThroughput}_i \times w_{rdt}}{t_{rdt}} - \frac{\text{WriteDiskThroughput}_i \times w_{wdt}}{t_{wdt}} - \frac{\# VM \times w_{vn}}{t_{vn}}$$

- $w$: weights in the $[0, 1]$ range
- $t$: sum of data gathered from every physical host, e.g., $t_{fr} = \sum_i \text{FreeRAM}_i$
Web Server Manager currently provides two main QoS 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 appropriately
Cloud Monitor gathers data from physical and virtual machines every 2.5s, **tagging a machine as failed if it does not receive data for 5s** (the period is reconfigurable at run-time)

**Virtual machine failure:**
- **worker**: WSM interacts with LBM to replace failed virtual machines with a new worker instance
- **balancer**: WSM instances a new balancer and restores the Apache mod_jk configuration via LBM

**Physical host failure**: each hosted virtual machine dynamically reallocated in other running hosts
Alerts to **identify if services are offered with degraded QoS**

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** (running a Tomcat worker instance) is **higher than a threshold** (default 6MB/s) for a given period (default 10s)

Once an alert has been triggered, Web Server Manager selects a host to **instantiate a new Tomcat worker on** and interacts with Load Balancer Manager to **update the mod_jk configuration**
Limit the consumption of virtual machines that could potentially harm other co-hosted services

Three alerts:

- **H-RAM**: whenever the free RAM of a host lowers below a threshold (default 15%)
- **H-DISK**: similar to H-RAM but considering the available mass storage
- **H-CPU**: whenever the CPU idle time goes below a threshold (10% default) for more than a configurable period time (15s default)
Sticky Session + Tomcat Cluster Domain to achieve horizontal scalability and failover

Two cluster domains of Tomcat workers each one with two replicas

Virtual machine running worker 1
Virtual machine running worker 2
Virtual machine running worker 3
Virtual machine running worker 4

Web Server
JK Load Balancer

Virtual machine running the load balancer
Four physical hosts
(Host A/D)

Eight virtual machines
providing three Web
servers (Server 1/3)
clustered in four groups of
two instances each

Server 1 is replicated four
times in two clusters, while
Server 2 and Server 3 two
times in one cluster each
Five physical hosts:

- HP1/2 with high performance (8GB RAM, quad-core with hyper-threading technology @3.2GHz): HP1 runs the EMMI framework and main OpenStack components monitoring and controlling the system, HP2 runs nova-network 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.
Test scenario: 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).

Machine failure simulated by abruptly disconnecting LP3 (not containing the virtual machine with mod_jk) from the network.

 Failover Test

- **Boot the virtual machine**: 20.3s
- **EMMI logic**: 0.2s
- **Identify the failure**: 5.6s

Time to instantiate a new replica (s)
Test scenario: one cluster with two replicas. The Web application consists of an HTML page with a 170KB image

Apache JMeter to emulate the HTTP workload below
Response times without EMMI framework failure in more than 40%

Average response time: 2188 ms
Dropped requests: 41.98%
Response times with EMMI framework

Average response time: 1939 ms
Dropped requests: 2.36%
Scale-out Test (4)

Workload, response times, and EMMI procedures

Scale-out Activated
New Replicas Added

Split, Croatia — 07.07.2013
Massimiliano Mattetti - MoCS '13
Our practical approach demonstrates that **private cloud solutions can be easily adopted** by SMEs in a cost effective way.

**Test results** prove the **effectiveness of our metrics**
- prompt adaptation and reconfiguration of physical hosts and virtual machines in relation to the current state of the environment

**Ongoing work**
- **additional services** on top our private cloud, e.g., private Dropbox-like store and distributed DBMS
- **federated deployment**, where remotely located cloud environments can easily interact and cooperate
Any Questions?

Thanks for your attention 😊
Questions time…

Contact info:
Massimiliano Mattetti
massimiliano.mattetti@unibo.it