Snooze: A Self-Organizing, Self-Healing and Self-Optimizing Cloud Management System

Demonstration

Eugen Feller*, Christine Morin*, Matthieu Simonin*, Anne-Cécile Orgerie** and Yvon Jégou*

*INRIA, Myriads project-team, Rennes, France
{*eugen.feller, christine.morin, matthieu.simonin, yvon.jegou*}*@inria.fr*
**CNRS, Myriads project-team, Rennes, France
anne-cecile.orgerie@irisa.fr

Abstract

Large-scale virtualized data centers enabling today's cloud services now require the cloud providers to implement easily configurable, fault-tolerant, and energy-efficient cloud management systems.

In this demo we present the self-organization, self-healing, and self-optimization mechanisms of Snooze, a novel cloud management system based on a hierarchical architecture. Particularly, we focus on four key features: (1) dynamic hierarchy construction; (2) hierarchy failure recovery; (3) periodic and event-based (e.g. underload) virtual machine live migration for idle time creation; (4) power management to transition idle servers into a low-power state for energy savings.

1 Introduction

Over the past years, the enormous demand for cloud services has motivated the cloud providers to deploy increasing numbers of large-scale data centers. Managing such data centers is a challenging task as it demands for cloud management system with a number of important and non-trivial to fulfill properties: (1) scalability; (2) fault tolerance; (3) energy efficiency; (4) ease of configuration. One way to enable the aforementioned properties is to implement decentralized self-organizing, self-healing, and self-optimizing cloud management systems. A number of attempts have been made in the last years to design and implement such systems with a primary focus on Virtual Machine (VM) management. Prominent examples include Eucalyptus [1], OpenStack [6], Nimbus [5], and VMware DRM [7]. However, all of them are still based on either centralized architectures and/or lack self-organization, self-healing, and self-optimization features. While the former aspect limits the scalability, the latter three aspects result in a reduced autonomy.

To tackle the introduced drawbacks of existing cloud management systems, in our previous work we have proposed Snooze [2, 3], a novel cloud management system based on a self-organizing, self-healing, and self-optimizing hierarchical architecture. Snooze is implemented from scratch in Java and comprises approximately 15 000 of highly modular code. It has been extensively evaluated on the Grid'5000 experimentation testbed using realistic applications and shown to be scalable, fault-tolerant, and energy-efficient. Since May 2012, Snooze has been distributed in open-source under the GPL v2 license at `http://snooze.inria.fr`.

In this demonstration we highlight the key features of the Snooze cloud management system. They involve ease of configuration, fault tolerance, and energy-efficiency via self-organization, self-healing, and self-optimization, respectively.

The remainder of this document is structured as follows. Section 2 introduces the Snooze architecture. Section 3 presents the demonstration scenarios. Finally, our use cases are discussed in Section 4.

2 System Architecture

We now give a brief overview of the Snooze cloud management system. The high-level Snooze system architecture is shown in Figure 1. It is partitioned in three layers: client, management, and compute. At the compute layer each physical server is managed by a system service, the so-called Local Controller (LC). To scale the system, LCs are grouped together with each of them being managed by a highly available system service at the management layer, the so-called Group Manager (GM). LCs receive VM life-cycle and server power management commands from the GM. Moreover, they monitor the resource utilization and detect server underload/overload situations. The GMs implement self-optimization mechanisms such as underload/overload mitigation, periodic VM consolidation, and power management. A Group Leader (GL) service oversees the GMs. It is elected

among the GMs during the system boot up and after a GL failure. The GL accepts VM submission requests from the clients and dispatches them among the GMs. Moreover, it integrates policies to assign LCs to GMs during system boot up and after a GM failure. Finally, a number of replicated system services, the so-called Entry Points (EPs) exist for the clients to discover the current GL. It is up to the system administrator to decide on the number of LCs, GMs, and EPs during the system setup. System services can by dynamically added/removed at runtime.

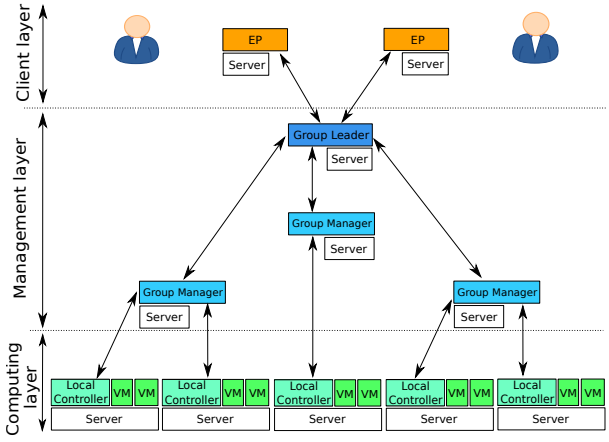

Figure 1: High-level system architecture

3 Demonstration Scenarios

We now introduce our demonstration scenarios. They are divided in three parts: self-organization, self-healing, and self-optimization. All the scenarios will be visualized using a Graphical User Interface (GUI) which provides a real-time view of the current hierarchy organization. They will be performed on a single machine by deploying all Snooze system services on the laptop.

Self-organization. To ease the system services configuration, when the system services are booted, the hierarchy needs to be dynamically constructed. This phase involves three steps: (1) GL election; (2) GM join; (3) LC join. In Snooze, GL election is implemented on top of the Apache ZooKeeper highly available coordination service [4]. This part of the demo visualizes how the GL is elected and joined (i.e. discovered and notified) by the GMs. Moreover, it shows how the LCs discover a GL, get a GM assigned, and register with the GM.

Self-healing. System services can fail at any time because of either hardware or software failures. We distinguish between three types of failures: GL, GM, and LC. In the event of a GL failure, a new GL must be elected among the GMs. In the event of GM failure, LCs previously assigned to the failed GM must be assigned to another GM. Finally, in the event of a LC failure, the LC needs to be removed from the assigned GM database. In this demo we inject GL, GM, and LC failures into the system and visualize the recovery.

Self-optimization. Finally, one of the major challenges in today's data centers is server underutilization and thus wasted energy. To remove underutilized servers, each GM integrates event-based server underload mitigation and performs periodic VM consolidation. A power management service exists to detect idle servers and transition them into a low-power state (e.g. suspend). In this part of the demo we visualize how Snooze leverages the aforementioned mechanisms to conserve energy.

4 Use Cases

We now discuss three Snooze use cases: (1) scalable data center management; (2) research testbed for VM management algorithms; (3) building block for Platform-as-a-Service (PaaS) cloud stacks.

Scalable data center management. The hierarchical architecture allows to manage production data centers hosting a large number of physical and virtual servers.

Research testbed for VM management algorithms. The modular system design enables researchers working on VM management algorithms (e.g. underload mitigation, consolidation) to easily integrate and test their algorithms in a realistic environment.

Building block for PaaS cloud stacks. The resource utilization data in Snooze, can be leveraged by PaaS cloud stacks to enable dynamic addition and removal of VMs based on their load and application objectives.

References

[1] Eucalyptus. `http://www.eucalyptus.com/`, 2013.

[2] FELLER, E., RILLING, L., AND MORIN, C. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2012), CCGRID '12, pp. 482–489.

[3] FELLER, E., ROHR, C., MARGERY, D., AND MORIN, C. Energy Management in IaaS Clouds: A Holistic Approach. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing* (2012), CLOUD '12, pp. 204–212.

[4] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Annual Technical Conference* (2010), USENIX ATC'10, pp. 11–11.

[5] KEAHEY, K. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *Cloud Computing and Its Applications 2008 (CCA-08)* (Chicago, IL, Oct. 2008).

[6] THE OPENSTACK PROJECT. OpenStack: The Open Source Cloud Operating System. `http://www.openstack.org/`, 2013.

[7] VMWARE. Distributed Resource Management: Design, Implementation and Lessons Learned. `http://labs.vmware.com/publications/gulati-vmtj-spring2012`, 2012.