Self-scaling the Cloud to meet Service Level Agreements

An open-source middleware framework solution

Antonin Chazalet, Frédéric Dang Tran,
and Marina Deslaugiers
France Telecom - Orange Labs,
28 chemin du vieux chêne,
38240 Meylan, FRANCE,
{antonin.chazalet, frederic.dangtran,
marina.deslaugiers}@orange-ftgroup.com

François Exertier, and Julien Legrand,
Bull,
1, rue de Provence,
B.P. 208
38432 Echirolles Cedex, FRANCE,
{francois.exertier, julien.legrand}@bull.net

***Abstract* - Cloud computing raises many issues about Virtualization and Service-Oriented Architecture (SOA). Topics to be addressed regarding services in Cloud computing environment include contractualization, monitoring, management, and autonomic management. Cloud computing, promotes a "pay-per-use" business model. This business model should enable to reduce costs but requires flexible services than can be adapted to load fluctuations. This work is conducted in the European CELTIC Servery cooperative research project, which deals about a telecommunication services marketplace platform. The Servery project focuses amongst others on the self-scaling capability of Telco services in a cloud environment. The self-scaling capability is achieved thanks to Service Level Agreement (SLA) monitoring and analysis (i.e., compliance checking), and to autonomic reconfiguration performed according to the analysis results. SLAs are defined for the services and for the cloud virtualized environment. In order to achieve this self-scaling capability, a specialized autonomic loop is proposed. Our proposal is well in line with the Monitor, Analyze, Plan, and Execute loop pattern defined by IBM. The proposed solution is based on the following open-source middleware: Service Level Checking, OW2 JASMINe Monitoring, OW2 JASMINe VMM. This paper presents this solution that has been implemented and validated in the context of the Servery project.**

Keywords* - *Cloud Computing; Autonomic; Self-Scaling; Service Level Agreement; Service Level Checking; Virtualization; Open-source.

I. INTRODUCTION

Today, almost all IT and Telecommunications industries are migrating to a Cloud computing approach. They expect that the Cloud computing model will optimize the usage of physical and software resources, improve flexibility and automate the management of services (i.e., Software as a Service, Platform as a Service, and Infrastructure as a Service). Cloud computing is also expected to enable data centers subcontracting from Cloud providers.

As a consequence, Cloud computing is seen as a way to reduce costs via the introduction and the use of "pay per use" contracts. It is also seen as a way to generate incomes for Cloud providers. Note that a Cloud provider can be:

- An infrastructure provider,
- A platform provider,
- And/or a software provider.

Cloud computing raises many issues. Many of them are related to Virtualization, and Service-oriented Architecture (its implementation and its deployment). These issues lie at both the hardware and/or software levels.

Nevertheless, Cloud computing also raises issues related to services contractualization, services monitoring, services management, and autonomic for the Cloud. In this paper, we address these last issues for telecommunication services offered to customers through a Cloud. These issues are critical: indeed economical concerns (i.e., the establishment and the use of the pay-per-use contracts) require the ability to contractualize services (via the use of service level agreements: SLA), to monitor and manage them, to check services contracts compliance, and to manage virtualized environments.

This paper is organized as follows. The next section provides background about Autonomic computing and Service Level Checking. Section 3 presents the related works. Section 4 outlines the autonomic approach we have followed. Section 5 focuses on the targeted Servery use case. Section 6 details our open-source solution. Section 7 describes the implementation. We present the validation and the results obtained in Servery in Section 8. Last section concludes this paper and gives directions for future works.

II. BACKGROUND

This section presents background about autonomic computing and service level checking.

A. Autonomic computing

Autonomic computing refers to computing systems (i.e., autonomic managers) that are able to manage themselves or others systems (i.e., managed resources) in accordance to management policies and objectives [1]. Thanks to automation, the complexity that human administrators are facing is moved into the autonomic managers. It allows administrators to concentrate on high-level management objectives definition and no more on the ways to achieve theses objectives. In [2], the authors define principles of autonomic computing thanks to a biological analogy with the

 ISBN: 978-1-61208-106-9

human nervous system: a human can achieve high level goals because its central nervous system allows him to avoid spending time on managing repetitive and vital background tasks such as regulating its blood pressure.

[2] specifies four main characteristics for describing systems self-management capabilities:

- Self-Configuration that aims to automate managed resources installation, and (re-)configuration.
- Self-Healing that purposes to discover, diagnose and act to prevent disruptions. Here, note that self-repair is a part of self-healing.
- Self-Protect that aims to anticipate, detect, identify and protect against threats.
- Self-Optimize that purposes to tune resources and balance workloads to maximize the use of information technology resources. Self-scaling is a subpart of self-optimization.

B. *Service Level Checking*

Generally speaking, Service Level Checking (SLC) involves a target service and a system in charge of collecting monitoring information and checking SLA compliance. More precisely, the target service offers probes, and its usage (or a derived usage) is contractualized with at least one SLA. SLA definitions are based on information that can be obtained through the services probes (directly or via calculation). The SLC system takes as input information regarding the target service as well as at least one SLA, and produces SLC results about the SLA compliance, the SLA violation, or errors that occurred during the checking or information collection steps. The target service can include software, platform and/or infrastructure, or can even be a Cloud itself (i.e., a set of software services, platform services and infrastructure services).

The SLC results can be used to inform a service administrator, to select a service provider at runtime, to launch an autonomic loop, and/or to break a contract.

III. RELATED WORKS

This related works section describes the solutions for autonomic computing proposed by equipment and IT vendors (i.e., IBM, Oracle, HP, Motorola, Cisco, Alcatel-Lucent ...). The focus is set on the use of SLA based service level checking as analyzing part (in autonomic MAPE loop) and the ability to manage virtualized environments (mandatory today in Cloud computing).

First, IBM uses policies managers as analyzers for the MAPE loop. IBM promotes the use of the Simplified Policy Language (SPL). SPL is based on Boolean algebra, arithmetic functions and collections operations. SPL also uses conditional expressions [3]. IBM Tivoli System Automation targets the reduction of the frequency and of the duration of service disruptions. It uses advanced policy-based automation to enable the high availability of applications and middleware running on a range of hardware platforms and operating systems. Note that these platforms and systems can be virtualized (or not). Tivoli's products family targets mainly availability and performance [4].

Second, Oracle provides the WebLogic Diagnostics Framework in order to detect SLA violations [5]. The Oracle Enterprise Manager 10g Grid Control can monitor services and report on service availability, performance, usage and service levels. Note that it doesn't manipulate SLA but a similar concept named Service Level Rule [6]. Oracle Enterprise Manager 11g Database Management is a solution to manage databases in 24x7. It self-tunes and self-manages databases operating w.r.t the performance, and it provides proactive management mechanisms (that involve service levels) in order to avoid downtime and/or performance degradation [7]. Oracle handles and manages virtualization through its Oracle VM Management Pack [8]. Oracle also leads research concerning PaaS and the Cloud, and provides a product called Oracle Fusion Middleware (OFM) [9]. OFM targets amongst others management automation, automated provisioning of servers, automate system adjustments as demand/requirements fluctuates. Note that unlike [1], Oracle specifies only three steps for the autonomic loop: Observe, Diagnose, and Resolve [10].

Third, autonomic architectures proposed by other equipment and IT vendors focus mainly on basic autonomic features in IT products [11]. It also shows that these remaining architectures don't use policies managers or SLA based service level checking as analyzing part, and don't manage virtualized environments.

The coming sections illustrate that our solution is well in line with the MAPE loop pattern. It uses a SLA based SLC as analyzing part and it manages virtualized environments. Moreover, unlike IBM and Oracle, it is an open-source solution: indeed, it only involves open-source middleware.

IV. APPROACH

The approach followed in this work is well in line with the Monitor, Analyze, Plan, and Execute loop pattern defined by IBM: the MAPE loop pattern (see Figure 1).

In [1], the authors defined that, similarly to a human administrator, the execution of a management task by an autonomic manager can be divided into four steps (that share knowledge):

- Monitor: The monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- Analyze: The analyze function provides the mechanisms that correlate and model complex situations (with regard to the management policy). These mechanisms enable the autonomic manager to learn about the IT environment and help predict future situations.
- Plan: The plan function provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- Execute: The execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

 ISBN: 978-1-61208-106-9

These four parts work together to provide the control loop functionality.

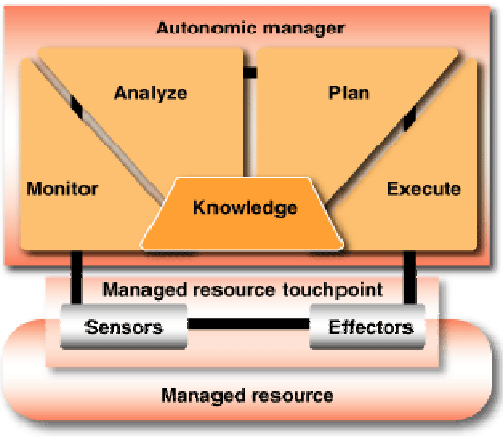

Figure 1. Autonomic loop (or MAPE/MAPE-K loop) [1].

V. THE SERVERY USE CASE

This section presents the Servery project description and the Servery self-scale-up use case.

A. *Servery research project description*

This sub-section presents the Servery research project context.

First, Servery (Service Platform for Innovative Communication Environment) is addressing the still unsolved problem of designing, developing and putting into operation efficient and innovative mobile service creation/deployment/execution platforms for networks beyond 3G [12]. One of the main goals of Servery is to propose a services marketplace platform where Telco services can be executed, and where end users can search, browse and access the executed services. Note that services published in the Servery marketplace platform can also be executed in others platforms belonging, e.g., to the telecommunication operators themselves.

The Figure 2 below shows an overview of Servery's context diagram, i.e., end users that are external actors of the system use Telco services provided by the Servery Marketplace Platform.

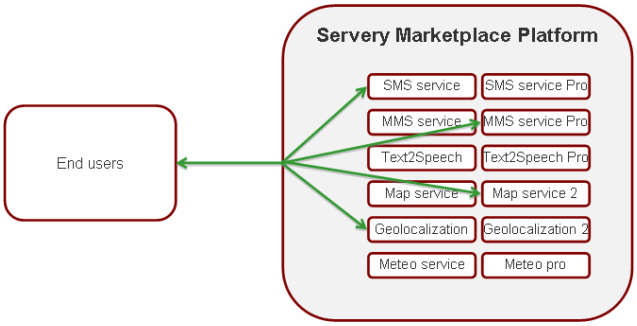

Figure 2. Servery's context diagram.

B. *Servery self-scale-up use case*

This sub-section presents the Servery self-scale-up use case. The goal of this use case is to maintain the overall QoS of the services executed in the Servery marketplace platform and of the marketplace platform itself while the user load grows up. QoS is directly (and indirectly) defined via SLAs. Here, the user load is represented by the number of end users (and consequently by the number of requests sent). The services targeted are Telco services, e.g., SMS services, e-mail services, etc.

VI. SOLUTION FOR SERVERY SELF-SCALING

As presented in the related works section, our proposition is well in line with the MAPE loop pattern defined in [1]. Our idea is to define SLAs between the administrators of the Servery marketplace platform and the marketplace platform itself. The whole MAPE loop proposed is based on these defined SLAs. It is named Servery marketplace management platform. Its monitoring and analyzing parts depend directly on the elements and metrics specified in the SLAs. As a reminder, the Servery marketplace platform is a Cloud. It means that three types of entities can be distinguished: the entities belonging to the software level, the platform entities and the infrastructure entities. SLAs defined can specify information related to these three types of entities.

More precisely, the analyzing part contains two distinct sub-parts: the SLC [13], and the JASMINe Monitoring (and its Drools module) [14]. The SLC is in charge of requesting the relevant probes and collecting the monitoring data. It is also in charge of checking the compliance of the defined SLAs with the collected monitoring data. It produces SLC results about the SLA compliance, the SLA violation, or errors occurred during the checking or information collection steps. JASMINe Monitoring takes these SLC notifications as input and checks their frequency over a configurable sliding time slot. This analysis over a sliding time slot is realized by a Drools module. Drools is a business logic integration platform which provides a unified and integrated platform for rules, workflow and event processing [15]. Using a sliding time slot analysis is interesting because it avoids to launch the planning and executing steps for non-significant/non-relevant events.

JASMINe Monitoring is also in charge of the planning part and leads the execution part. All the execution actions related to the virtual machines management is done via the mechanisms provided by JASMINe Virtual Machines Management (JASMINe VMM) [16].

The Servery marketplace platform (see Figure 3) was designed with a front-end element (i.e., an Apache HTTP Server) and at least one services execution environment (i.e., an OW2 JOnAS open-source Java EE 5 Application Server [17]). This design allows us to be able to scale-up the Servery marketplace platform and the Telco services deployed in it. In short, the Apache front-end acts as a load balancer. Note that the Apache front-end and all the JOnAS server(s) are run in virtual machines themselves run over the Xen hypervisor technology - an open source industry standard for virtualization [18].

 ISBN: 978-1-61208-106-9

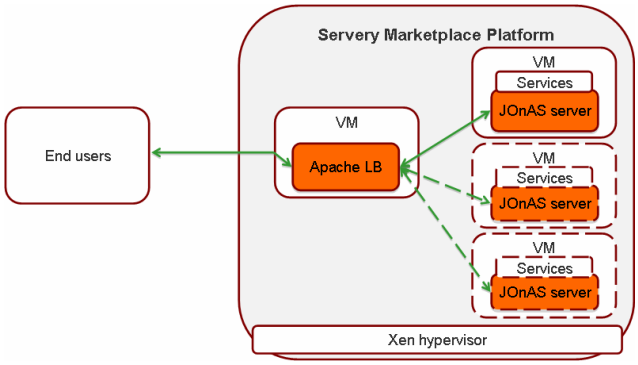

Figure 3. Servery marketplace platform architecture.

This marketplace platform design is interesting, because it enables to easily support the addition and/or removal of services execution environments. The only constraint of this design is the need to reconfigure the front-end element in order to take into account the addition and/or removal.

VII. IMPLEMENTATION

This section presents the implementation of the proposed solution. Our solution involves two high level modules: the Servery marketplace platform that is in charge of providing services to the end users, and the Servery marketplace management platform that ensures the scale-up autonomic property.

The Servery marketplace management platform involves four distinct modules:

- Service level checking is in charge of requesting the relevant probes and collecting the monitoring data from the Servery marketplace platform. It is also in charge of checking the compliance of the defined SLAs with the monitoring data collected. It produces SLC results that are sent to JASMINe monitoring. SLC is developed by France Telecom.
- JASMINe Monitoring is part of the OW2 JASMINe project. The OW2 JASMINe project aims to develop an administration tools suite dedicated to SOA middleware such as application servers (Apache, JOnAS, ...), MOM (JORAM, ...) BPM/BPEL/ESB solutions (Orchestra, Bonita, Petals, ...) in order to facilitate the system administration [19]. JASMINe Monitoring takes these SLC notifications as input and checks their frequency over a configurable sliding time slot. It is also in charge of the planning step and it leads the scale-up execution step. JASMINe Monitoring is developed by Bull.
- Cluster scaler is in charge of transmitting execution actions to JASMINe VMM. It is also in charge of the reconfiguration of the Apache Load Balancer in order to take into account the virtual machine just added. Cluster scaler is developed by Bull.
- JASMINe VMM is in charge of the management of the virtual machines created and executed over the Xen hypervisor. JASMINe VMM aims at offering a unified Java-friendly API and object model to manage virtualized servers and their associated hypervisor. In short, it provides a JMX hypervisor-agnostic façade/API in front of proprietary virtualization management protocols or APIs (such as the open-source Xen and KVM hypervisors, the VMware ESX hypervisor, the Citrix Xen Server hypervisor, and the Microsoft Hyper-V 2008 R2). JASMINe VMM is developed by France Telecom.

Note that the solution we propose is a fully open-source and Java based solution, and that all communications are done via the Java Management eXtension technology (JMX).

We now present the nominal steps executed when an autonomic scale-up is launched (see Figure 4). Here, the Servery Marketplace Platform initially contains two virtual machines (one containing the Apache LB, and one containing a JOnAS server and Telco services). End users request/interact with the (services of the) Servery marketplace platform is referred as step number 0. A nominal execution involves 6 steps (from 1 to 6):

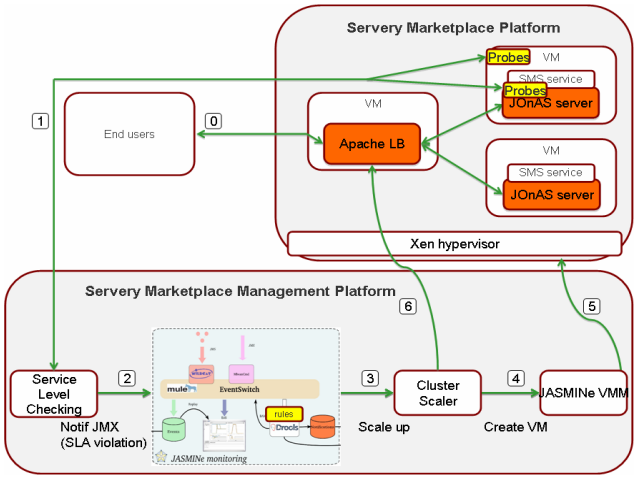

Figure 4. Overview of the proposed solution.

First, the objective of SLC is to check the compliance of the Servery Marketplace Platform (and its Telco services) with SLAs related: to the SaaS level (i.e., Telco services level), to the PaaS level (i.e., JOnAS server level), and to the IaaS level (i.e., virtual machines level). Consequently, SLC requests probes related to the Telco services, the JOnAS server and the virtual machine with regard to the contracts wanted. Amongst all the possible probes, we have chosen to focus and collect the following Telco services (SaaS) information:

- The number of requests processed during the last (configurable) time period
- The total processing time during the last period
- The average processing time during the last period

The JOnAS server information chosen was:

- The current server state (e.g., starting, running)
- The number of active HTTP sessions
- The number of services deployed/running in a server

 ISBN: 978-1-61208-106-9

The virtual machine information chosen was:

- The virtual machine CPU load
- The total memory (heap and no-heap)
- The used memory (heap and no-heap)

Second, SLC results are sent to JASMINe monitoring. JASMINe monitoring then checks the frequency of the SLC results corresponding to a violation. If the frequency of the violations is too high (e.g. more than five violations in a one minute sliding time slot), it means that a scale-up action is needed. So, JASMINe monitoring plans this scale-up action (thanks to information known about the marketplace platform) and executes it. Here, it means that JASMINe monitoring plans to introduce and configure another virtual machine containing a JOnAS server and the Telco services.

Third, the scale-up action is sent to the Cluster Scaler.

Fourth, Cluster Scaler commands the JASMINe VMM to create a new virtual machine (containing a JOnAS server and the Telco services).

Fifth, JASMINe VMM commands the Xen hypervisor in order to introduce the specified virtual machine. By introducing a virtual machine, we mean creating, instantiating and launching the virtual machine (and its content).

Sixth, Cluster Scaler is informed that the requested virtual machine has correctly been instantiated and is now in the running state. Then, Cluster Scaler reconfigures the Apache Load Balancer in order to take into account the new virtual machine (and its content) just introduced.

Finally, the load induced by the end users requests is now dispatched between the two virtual machines (containing the JOnAS Servers and the services).

VIII. VALIDATION

This section presents details, screenshots, and results about the demonstration associated to the scale-up use case.

First, our solution has been demonstrated to CELTIC and French National Research Agency (ANR) experts during the Servery project's mid-term review (the 7th of May 2010).

This live demonstration and the validation were done on three standards servers: one dedicated to the marketplace platform, one containing the marketplace management platform, and one in charge of injecting the end users load to the marketplace platform.

Over this hardware configuration, we observed that our whole MAPE loop runs approximately in 10 minutes (this is an average value coming from ten consecutive experimentations. These 10 minutes are broken down as follows:

- 1 minute is taken by SLC and JASMINe monitoring in order to monitor and detect 5 consecutive SLA violations in a 1 minute sliding time slot.
- 1 minute is taken by JASMINe monitoring for the planning of the scale-up action and the launching of the execution step.
- 1 minute is spent by JASMINe VMM in order to interact with the Xen hypervisor for introducing a new virtual machine.
- At least 6 minutes are consumed by the creation, the boot and the initialization steps of the (just introduced) virtual machine.
- Less than 1 minute is spent by Cluster Scaler to reconfigure the marketplace platform and check its state.

Note that the creation of the virtual machine can be reduced to a dozen seconds via the use of virtual machine templates; the boot and initialization steps can't be easily shortened.

Figure 5 below is a screenshot of SLC. It shows SLC results: here, one violation of the SLA tsla_id_3 has been detected).

MsgId	OccuredAt	jVM_freePhysicalMemorySize (octets)	jVM_CpuLoad	JOnAS state	appli_numberOfRequests	appli_averageProcessingTime	SlaId	SLCResult
uuid:92	2010-05-06 18:40:06.218	167325696	0.00396275	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:91	2010-05-06 18:40:01.187	167325696	0.0039753527	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:90	2010-05-06 18:39:56.14	167325696	0.0079507055	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:89	2010-05-06 18:39:51.125	167325696	0.00385505	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:88	2010-05-06 18:39:45.937	167325696	0.0059265113	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:87	2010-05-06 18:39:40.875	167325696	0.003950227	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:86	2010-05-06 18:39:35.812	167317504	0.0039635357	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:85	2010-05-06 18:39:30.765	167301120	0.0059253406	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:84	2010-05-06 18:39:25.703	167325696	0.00396275	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:83	2010-05-06 18:39:20.64	167325696	0.0059265113	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:82	2010-05-06 18:39:15.593	167325696	0.00396275	j2ee.state.running	0	-1.0	tsla_id_3	GREEN
uuid:81	2010-05-06 18:39:10.546	167309312	0.0118156755	j2ee.state.running	17	1.0	tsla_id_3	GREEN
uuid:80	2010-05-06 18:39:05.484	167309312	0.07900454	j2ee.state.running	169	1.0946746	tsla_id_3	RED
uuid:79	2010-05-06 18:39:00.406	167309312	0.053333335	j2ee.state.running	186	1.0645162	tsla_id_3	GREEN

Figure 5. Screenshot of SLC with a SLA violation.

Figure 6 is a screenshot of JASMINe VMM. It shows the marketplace platform after a self-scale-up. Three virtual machines are displayed: one containing the Apache LB (called apache) and two containing each a JOnAS server and the Telco services (called jonasWorker1 and jonasWorker3).

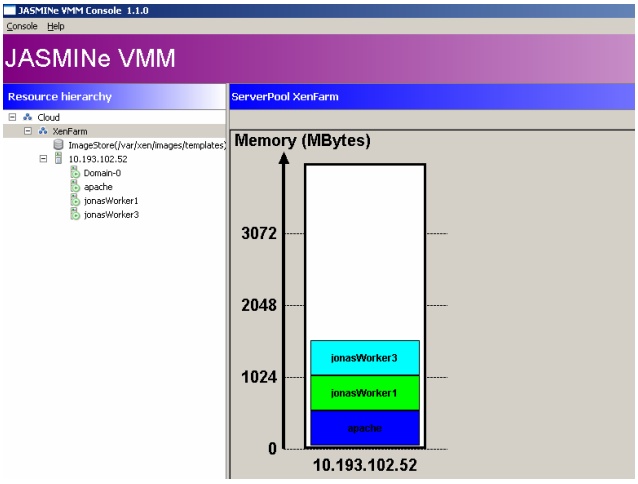

Figure 6. Screenshot of JASMINe VMM with 2 JOnAS servers.

Figure 7 below shows the number of requests, the average processing time, and the CPU load corresponding to jonasWorker1. Here, we have injected two identical loads on the Apache LB. The first load has led to a SLA violation and

the marketplace platform has been self-scaled-up. The second load has been injected after the self-scale-up action; the load is now balanced between the two jonasWorkers.

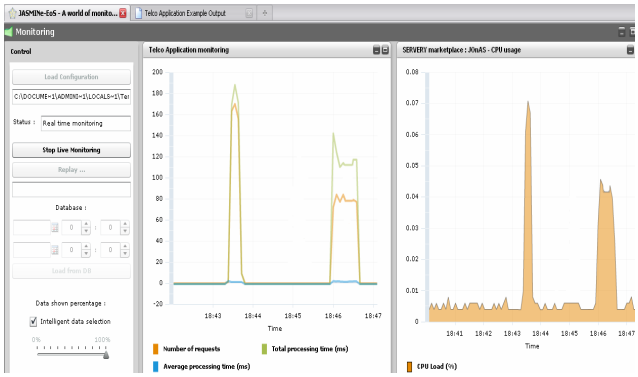

Figure 7. Screenshot of JASMINe Monitoring graphs.

IX. CONCLUSION

In this paper, we have presented an innovative open-source solution for self-scaling the cloud to meet service level agreements. Our solution has been applied to the Cloud Computing context via a self-scaling use case coming from the European CELTIC Servery cooperative research project. Applying our proposal to this use case has led us to several conclusions. First, according to the objectives, it allows to self-scale a virtualized cloud depending on the compliance with SLA. It also allows separating concerns related to the monitoring, analyzing, planning and executing steps in an industrial context and in the frame of an industrial use case.

Second, our solution is functional and efficient. It has been demonstrated in front of experts and validated.

Third, one of the important challenges we solved with this solution was to find, extend/modify, and integrate open-source middleware pieces with respect to industrial constraints raised by our R&D centers.

Last, but not least, this solution is well accepted by both France Telecom and Bull production project teams.

As future work, we consider to work on the Servery self-scale-down and self-repair use cases. We also plan to introduce several monitoring probes in the marketplace platform, to extend the SLC module in order to check more complex SLAs, and to embed it in JASMINe monitoring in order to take advantage of its monitoring mechanisms. We also plan to extend both the Drools rules for the analysis step and the planning mechanism in order to handle the two remaining use cases. We also wish to use JASMINe VMM capabilities in order to test our solution on a VMware marketplace platform.

ACKNOWLEDGMENT

We would like to thank the European CELTIC program for co-funding the SERVERY project (project number CP5-023), as well as both France Telecom - Orange Labs and Bull. Special thanks to Dr. Alexandre LEFEBVRE for his help and to Dr. Thierry COUPAYE for hosting this work.

REFERENCES

[1] IBM, "An architectural blueprint for autonomic computing", white paper, http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf, Jun. 2006, [last accessed Nov. 2010].

[2] Horn P., "Autonomic Computing: IBM's perspective on the State of Information Technology", in IBM corporation, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, Oct. 2001, [last accessed Nov. 2010].

[3] IBM, "Simplified Policy Language", http://download.boulder.ibm.com/ibmdl/pub/software/dw/autonomic/ac-spl/ac-spl-pdf.pdf, 2008, [last accessed Nov. 2010].

[4] IBM, "Virtualization Management", http://www-01.ibm.com/software/tivoli/solutions/virtualization-management/, 2010, [last accessed Nov. 2010].

[5] Oracle, "Monitoring Performance Using the WebLogic Diagnostics Framework", http://www.oracle.com/technetwork/articles/cico-wldf-091073.html, August 2009, [last accessed Nov. 2010].

[6] Oracle, "Service Management", http://download.oracle.com/docs/cd/B19306_01/em.102/b31949/service_management.htm, 2009, [last accessed Nov. 2010].

[7] Oracle, "Oracle Enterprise Management 11g Database Management", http://www.oracle.com/technetwork/oem/db-mgmt/index.html, 2010, [last accessed Nov. 2010].

[8] Oracle, "Oracle VM Management Pack", http://www.oracle.com/technetwork/oem/grid-control/ds-ovmp-131982.pdf, 2010, [last accessed Nov. 2010].

[9] Oracle, "Platform-as-a-Service Private Cloud with Oracle Fusion Middleware", Oracle White Paper, http://www.oracle.com/us/036500.pdf, October 2009, [last accessed Nov. 2010].

[10] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic Performance Diagnosis and Tuning in Oracle", 2nd Conference on Innovative Data Systems Research (CIDR), http://www.cidrdb.org/cidr2005/cidr05cd-rom.zip, pp. 84-94, 2005.

[11] Eurescom, " Autonomic Computing and Networking: The operators' vision on technologies, opportunities, risks and adoption roadmaps", http://www.eurescom.eu/~pub/deliverables/documents/P1800-series/P1855/D1/, 2009, [last accessed Nov. 2010].

[12] Servery consortium, "SERVERY Celtic project", http://projects.celtic-initiative.org/servery/, 2010, [last accessed Nov. 2010].

[13] Chazalet A., "Service Level Agreements Compliance Checking in the Cloud Computing", 5th International Conference on Software Engineering Advances (ICSEA), pp. 184-189, 2010.

[14] OW2 consortium, "JASMINe Monitoring", http://wiki.jasmine.ow2.org/xwiki/bin/view/Main/Monitoring, 2010, [last accessed Nov. 2010].

[15] JBoss Community, "Drools 5 - The Business Logic integration Platform", http://www.jboss.org/drools, 2010, [last accessed Nov. 2010].

[16] OW2 consortium, "JASMINe Virtual Machine Management", http://wiki.jasmine.ow2.org/xwiki/bin/view/Main/VMM, 2010, [last accessed Nov. 2010].

[17] OW2 consortium, "OW2 JOnAS open-source Java EE 5 Application Server", http://jonas.ow2.org/, 2010, [last accessed Nov. 2010].

[18] Citrix Systems, "The Xen Hypervisor", http://www.xen.org/, 2010, [last accessed Nov. 2010].

[19] OW2 Consortium, "JASMINe: The Smart Tool for your SOA Platform Management", http://jasmine.ow2.org/, 2010, [last accessed Nov. 2010].

 ISBN: 978-1-61208-106-9