Adding Self-scaling Capability to the Cloud to meet Service Level Agreements

An Open-source Middleware Framework Solution

Antonin Chazalet, Frédéric Dang Tran,
Marina Deslaugiers and Alexandre Lefebvre
France Telecom - Orange Labs,
{antonin.chazalet, frederic.dangtran,
marina.deslaugiers, alexandre.lefebvre}@orange.com

François Exertier and Julien Legrand,
Bull,
{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. It should enable to reduce costs but requires flexible services than can be adapted, *e.g.*, to load fluctuations. This work is conducted in the European CELTIC Servery cooperative research project, which deals about a telecommunication services marketplace platform. This project focuses amongst others on the self-scaling capability of Telco services in a cloud environment. This 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 contractualize the services and the cloud virtualized environment itself. In order to achieve the self-scaling capability, a specialized autonomic loop is proposed. Our proposal is based on the Monitor, Analyze, Plan, and Execute autonomic loop pattern (defined by IBM) that has been enhanced via the introduction, and the use of SLA. The implementation we provide is based on the following open-source middleware components: Service Level Checking, OW2 JASMINe Monitoring, OW2 JASMINe VMM. Our proposition has been validated in the context of the Servery project [1].**

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

I. INTRODUCTION

Today, almost all Information Technology (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 income for Cloud providers that can be:

- Infrastructure providers,
- Platform providers,
- And/or software providers.

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

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 the 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, Service Level Checking, and Service Level Agreement. Section 3 presents the related works. Section 4 details the autonomic approach we have followed, and the use of SLA. Section 5 focuses on the targeted Servery use cases (*i.e.*, self-scale-up, and self-scale-down). Section 6 details the open-source solution we propose. 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 work.

II. BACKGROUND

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

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 [2].

Thanks to automation, the complexity that human administrators are facing is moved into the autonomic managers. Administrators can then concentrate on defining high-level management objectives and no longer on the ways to achieve these objectives.

In [3], Horn defines principles of autonomic computing following a biological analogy with the 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 his blood pressure.

They also specify 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 in order 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 computation). The SLC system takes as input information regarding the target service as well as SLA, and produces SLC results about the SLA compliance, the SLA violation, or errors that occurred during the information collection or checking steps.

In the Cloud computing context, the target services 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, *e.g.*, the target service administrator, or the hotline support,
- To dynamically select services at runtime, depending on the compliance between SLA and the services "really" offered,
- To make decisions regarding specific preoccupations, *e.g.*, Green,
- Prior to a deployment, *e.g.*, in order to analyze the target compatibility,
- To provide analysis information to a SLA enforcement mechanism,
- In an autonomic loop,
- And/or to terminate a contract.

C. *Service Level Agreement*

A SLA (*i.e.*, a service contract or a contract associated to a service) can be used in order to specify the service offered by a service provider or the service expected by a service client. Contracts can be both about functional or non-functional aspects. A single SLA can contractualize several services, and several different SLA can contractualize the same service (*e.g.*, a SLA for autonomic preoccupations, another for Green preoccupations).

In [4], Beugnard and al. define four contract levels of increasingly negotiable properties: Syntactic level, *e.g.*, the Java interfaces, Behavioral level that requires the definition of pre and post conditions, Synchronization level, which specifies the global behavior in terms of synchronizations between method calls, and finally, Quality of Service level that specifies the expected QoS.

Web Services Agreement Specification is a specification for SLA in the Web Services domain [5]. It is proposed by the Open Grid Forum [6].

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, etc.). 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. It also uses conditional expressions [7]. 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. These platforms and systems can be virtualized (or not). Tivoli's products family targets mainly availability and performance [8].

Second, Oracle provides the WebLogic Diagnostics Framework in order to detect SLA violations [9]. The Oracle Enterprise Manager 10g Grid Control can monitor services and report on service availability, performance, usage and service levels. It doesn't manipulate SLA but a similar concept named Service Level Rule [10]. 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 [11]. Oracle handles and manages virtualization through its Oracle VM Management Pack [12]. Oracle also leads research concerning Platform As A Service (PaaS) and the

Cloud, and provides a product called Oracle Fusion Middleware (OFM) [13]. OFM targets amongst others management automation, automated provisioning of servers, automate system adjustments as demand/requirements fluctuates. Unlike [2], Oracle specifies only three steps for the autonomic loop: Observe, Diagnose, and Resolve [14].

Third, autonomic architectures proposed by other equipment and IT vendors focus mainly on basic autonomic features in IT products [15]. These remaining architectures do not use policies managers or SLA-based service level checking as analyzing part, and do not manage virtualized environments.

The coming sections illustrate that our solution is 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 components.

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 addition, we chose to design and implement a SLA-driven Analyze step. We believe that the use of SLA in the Analyze step is a pertinent choice, indeed, SLA enables the description of complex situations/states as they can be encountered in the Cloud.

A. MAPE Loop

In [2], the authors defined that, similarly to a human administrator, the execution of a management task by an autonomic manager can be divided into four parts (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.

These parts work together to provide the control loop functionality.

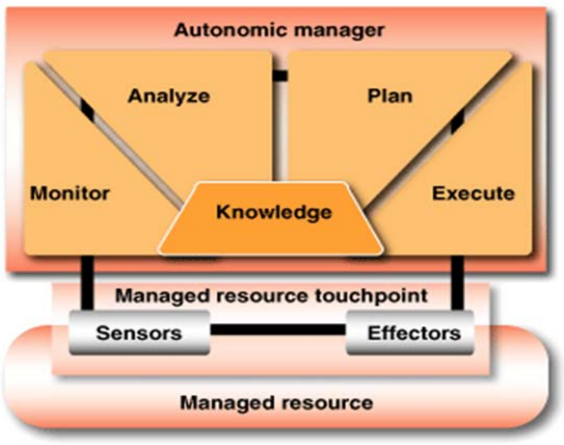


Figure 1. Autonomic loop (or MAPE/MAPE-K loop) [2].

B. Service Level Agreement

Each SLA we manipulate, must, at least, specify:

- A unique identifier,
- Its beginning and ending dates,
- A temporality/occurrence (the value of a temporality defines the length of a sliding time slot; the value of an occurrence defines a number of consecutives SLO violations),
- A non-empty set of Service Level Objectives (SLO),
- And a human-readable description.

SLO can be seen as articles/clauses within a contract. A SLO must, at least, specify:

- A unique identifier in the SLA,
- The targeted information,
- A comparison operator,
- A threshold value,
- And a human-readable description including, amongst others, the measurement unit of the targeted information.

By default, a SLA is violated when the conjunction of its SLO is false, and when its temporality/occurrence is crossed.

Regarding the four contract levels defined in [4], SLA we manipulate definitely belong to the fourth level: QoS contracts.

V. THE SERVERY USE CASE

This section presents the Servery research project and the self-scale-up, and self-scale-down use cases.

A. Servery research project description

This sub-section presents Servery's context. First, Servery (which targets a 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 [16].

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. Services published in the Servery marketplace platform can also be executed in others platforms belonging, *e.g.*, to the telecommunication operators themselves. The services targeted in Servery are stateless services.

The Figure 2 below shows the 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.

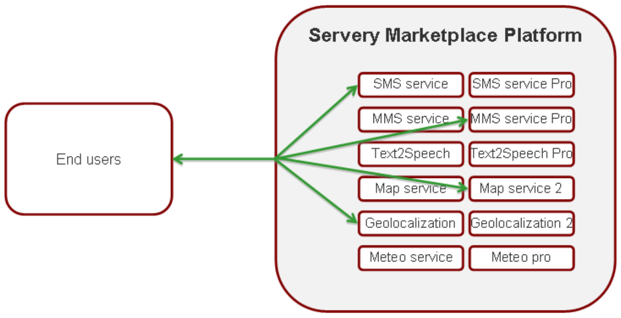


Figure 2. Servery's context diagram.

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 to the services). The services targeted are Telco services, *e.g.*, SMS services, weather forecast services, etc.

C. *Servery self-scale-down use case*

This sub-section presents the Servery self-scale-down use case. This use case's goal 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 falls down. The idea, here, is to minimize the cost of the services execution in the Servery marketplace platform, and of the marketplace platform execution itself while preserving services' QoS (at a defined level). A positive side effect of the reduction of the overall system's size is that it also enables the system to be Greener.

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 [2]. 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 [17], and the JASMINe Monitoring (and its Drools module) [18]. 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 [19]. Using a sliding time slot analysis is interesting because it avoids launching 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) [20].

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 [21]). This design enables us to be able to scale-up, and scale-down 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 [22].

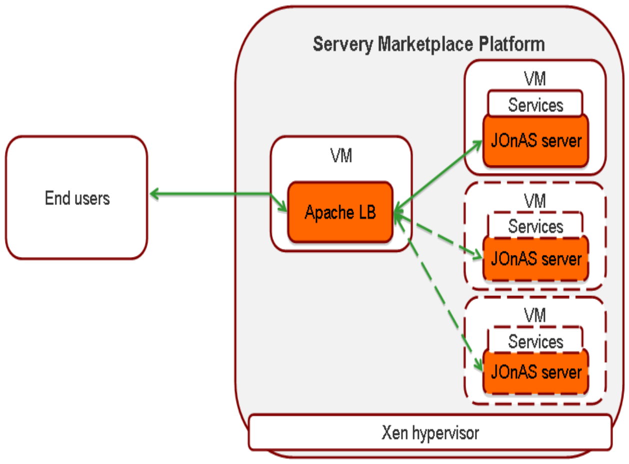


Figure 3. Servery marketplace platform architecture.

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

VII. PROTOTYPE

This section presents the proposed solution. It 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, and scale-down autonomic properties.

The Servery marketplace management platform involves four distinct modules:

- The Service level checking module 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, etc.) in order to facilitate the system administration [23]. JASMINe Monitoring takes 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 and scale-down execution steps. 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 hypervisor). 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).

A. *Self-scale-up use case*

We now present the nominal steps executed by our solution when the need of a scale-up action is detected, and when a scale-up action is then executed (see Figure 4).

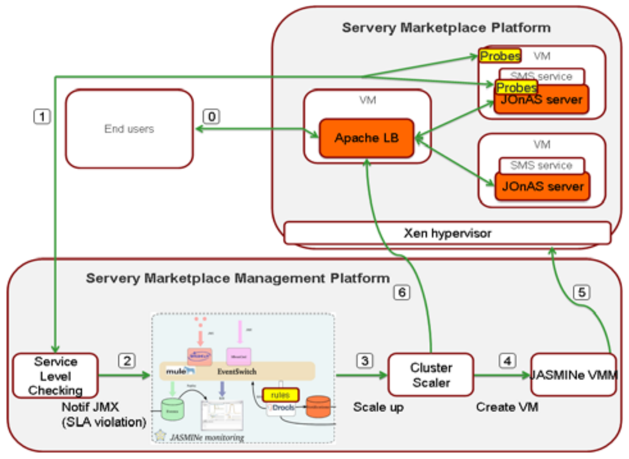


Figure 4. Overview of the self-scale-up solution.

Here, the Servery Marketplace Platform initially contains two virtual machines (one containing the Apache LB, and one containing a JOnAS server and Telco services). The fact that end users request/interact with the (services of the) Servery marketplace platform is referred to as step number 0. A nominal execution involves 6 steps (from 1 to 6).

First, the objective of SLC is to check the compliance of the Servery Marketplace Platform (and its Telco services) with SLAs related: to the Software As A Service (SaaS) level (i.e., Telco services level), to the PaaS level (i.e., JOnAS server level), and to the Infrastructure As A Service (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

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).

B. *Self-scale-down use case*

We now present the design of the nominal steps that should be executed when the need of a scale-down action is detected, and when a scale-down action is then executed (see Figure 5). Let's start with a Servery Marketplace Platform that initially involves three virtual machines (one containing the Apache LB, and two containing each a JOnAS server and the Telco services). The fact that end users request/interact with the (services of the) Servery marketplace platform is also referred here to as step number 0. A nominal scale-down execution involves 6 steps (from 1 to 6).

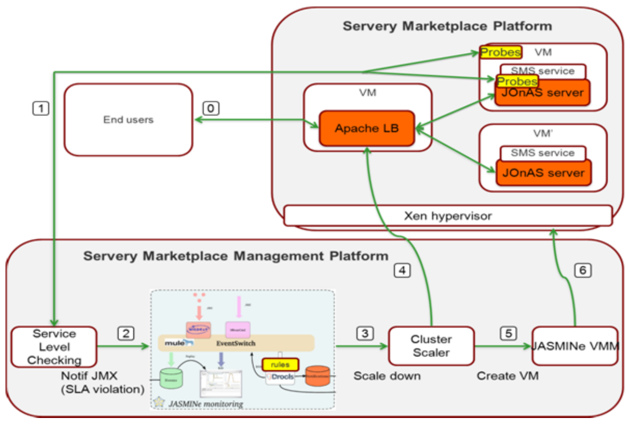


Figure 5. Overview of the self-scale-down solution.

First, SLC's objective is, here again, to check the compliance of the Servery Marketplace Platform (and its Telco services) *w.r.t.* the specified SLAs. Therefore, SLC requests the probes related to the Telco services, the JOnAS server and the virtual machine. It then produces SLC results.

These results are, then, sent to JASMINe monitoring. This latter checks the frequency of the violation. Like in the scale-up steps, we chose that more than five violations in a one minute sliding time slot means that a scale-down action is required. If such a case occurs, JASMINe monitoring then plans the scale-down action, and executes it. A virtual machine containing a JOnAS server and the Telco services will therefore be selected (let's say " VM' " in Figure 5.), and removed from the Marketplace Platform.

After that, the scale-down action is sent to the Cluster Scaler.

Fourth, Cluster Scaler reconfigures the Apache Load Balancer in order to take into account the future removal of the chosen virtual machine.

Fifth, Cluster Scaler requests the JASMINe VMM to remove the given virtual machine.

Sixth, JASMINe VMM then commands the Xen hypervisor to remove the virtual machine. By removing, we mean stopping, and deleting the virtual machine (and its content). Cluster Scaler is then informed that the requested virtual machine has correctly been removed.

Consequently, the load induced by the end users requests is now factorized/centered, here, on the remaining virtual machine.

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.

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.

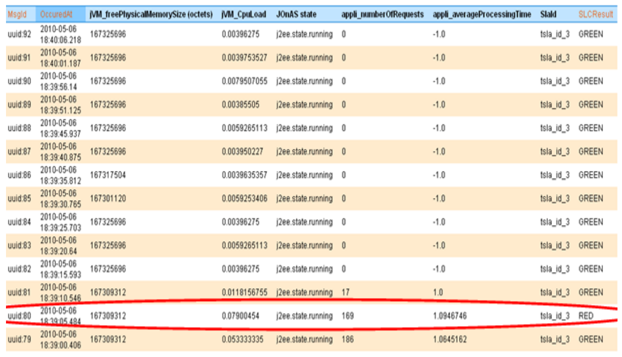
Over this hardware configuration, we observed that our whole MAPE loop runs approximately in 10 minutes (this is an average value coming from ten consecutive experimentations. These 10 minutes are broken down as follows:

- 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 execution step launching.
- 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 are complex to shorten.

Figure 6 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 6. Screenshot of SLC with a SLA violation.

Figure 7 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).

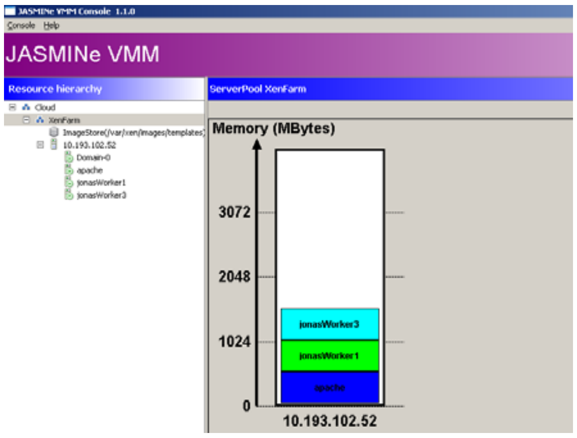


Figure 7. Screenshot of JASMINe VMM with 3 Virtual Machines.

Figure 8 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 amongst the two jonasWorkers.

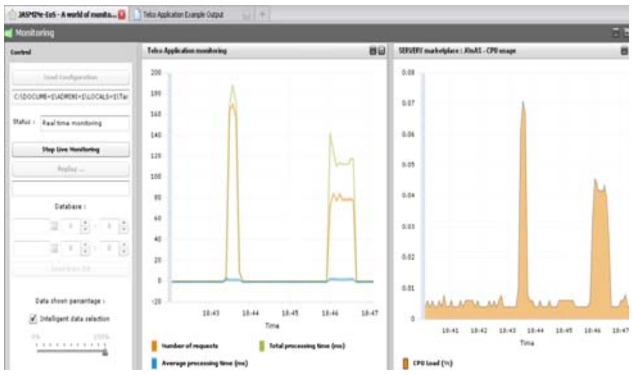

Telco Application Example Output
Monitoring
Control
Status : Real time monitoring
Stop Live Monitoring
Replay ...
Database :
Data shown percentage :
Intelligent data selection
Telco Application monitoring
Time
Number of requests
Total processing time (ms)
Average processing time (ms)
CPU load (%)
Time

Figure 8. Screenshot of JASMINe Monitoring graphs.

IX. Conclusion and Future Work

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 two self-scaling use cases coming from the European CELTIC Servery cooperative research project. Applying our proposal to these use cases has led us to several conclusions.

First, according to the objectives, it enables to self-scale a virtualized cloud depending on the compliance with SLA. It also enables separating concerns related to the monitoring, analyzing, planning and executing steps in an industrial context and in the frame of industrial use cases.

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

Third, an important challenge 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 project teams.

As future work, we plan to introduce several other 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 study how self-scale-up and self-scale-down mechanisms can coexist, as well as the self-scaling of statefull services.

We also wish to use JASMINe VMM capabilities in order to test our solution on a VMware based marketplace platform.

Finally, it would be interesting to enhance our current solution by adding self-scaling capability at a lower/finer grain, *i.e.* SaaS grain, in addition of the current PaaS grain; the OW2 Sirocco middleware can be an interesting basis to do so [24].

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. Thanks to Dr. Thierry COUPAYE for hosting this work.

REFERENCES

[1] Chazalet A., Dang Tran F., Deslaugiers M., Exertier F., and Legrand J., "Self-scaling the Cloud to meet Service Level Agreements", IARIA - Cloud Computing 2010, Lisbon, Nov. 2010, pp. 116-121.

[2] 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].

[3] 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].

[4] Beugnard A., Jezequel J.-M., Plouzeau N., and Watkins D., "Making components contract aware", IEEE Computer, vol. 32, no 7, 1999.

[5] Andrieux A., Czajkowski K., Dan A., Keahey K., Ludwig H., Kakata T., Pruyne J., Rofrano J., Tuecke S., and Xu M., "Web Services Agreement Specification (WS-Agreement)", Open Grid Forum specification, http://www.ogf.org/documents/GFD.107.pdf, 2007.

[6] Open Grid Forum, http://www.ogf.org/, [last accessed Jan. 2012].

[7] 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].

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

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

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

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

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

[13] 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].

[14] 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.

[15] 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].

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

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

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

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

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

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

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

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

[24] OW2 Consortium, "Sirocco: A Multi-Cloud Infrastructure-as-a-Service Software Platform", http://www.ow2.org/view/ActivitiesDashboard/Sirocco, 2011, [last accessed Jul. 2011].