A Profile-based Approach to Just-in-time Scalability for Cloud Applications

YANG Jie, QIU Jie, LI Ying
IBM China Research Laboratory, Beijing 100094, P.R.China
yangyjie@cn.ibm.com, jieqiu@cn.ibm.com, lying@cn.ibm.com

Abstract

Cloud platforms offer resource utilization as on-demand service, which lays the foundation for applications to scale during runtime. However, just-in-time scalability is not achieved by simply deploying applications to cloud platforms. Existing approaches require developers to rewrite their applications to leverage the on-demand resource utilization, thus bind applications to specific cloud infrastructure. In this paper, profiles are used to capture experts' knowledge of scaling different types of applications. The profile-based approach automates the deployment and scaling of applications in cloud. Just-in-time scalability is achieved without binding to specific cloud infrastructure. A real case is used to demonstrate the process and feasibility of this profile-based approach.

1. Introduction

More and more cloud applications are anticipating just-in-time scalability due to the unpredictability of their usage workload. Typical examples include the famous Facebook application, "Friends for Sale", which induces 300% monthly growth rate that even surprises its founder [1]. Some social networking sites even went through a higher growth rate during some specific phases [2]. Just-in-time scalability helps to avoid possible application down time when the runtime infrastructure of these applications could not accommodate the usage workloads. By "just-in-time scalability", we mean the ability for a cloud application to expand or shrink its architecture at runtime to adapt it to the dynamic changing workloads. Just-in-time scalability comes with not only dynamic resource consuming (i.e. an application occupies more resource only when its workload increases; alternatively, the occupied resource shrinks when the workload decreases), but also runtime architecture scaling (i.e. components constituting an application increases or decreases with proper configurations to ensure consistency and efficiency).

Traditionally, the scalability of cloud applications is achieved by deploying them to scalable application servers (such as IBM WebSphere Application Server, Tomcat, and so on), which offer scalability by server-specific clustering mechanisms. This approach is not satisfiable for desired just-in-time scalability, because the cluster size must be determined before the deployment of applications and the cluster size is difficult to change during runtime. The fixed cluster size also implies a fixed cost for application providers. They have to tolerate the high cost for the whole cluster when the workload of their applications is light.

The emerging cloud platforms offer opportunities to overcome the above limitation. Typical cloud platforms include IBM Research Computing Cloud (RC2) [3], Amazon Elastic Computing Cloud (EC2) [4], Google App Engine [5], Microsoft Azure [6]. Different cloud platforms share some common characteristics, including on-demand resource utilization, and utility-based payment [7]. By "on-demand resource utilization", applications in cloud could apply for more resource only when needed, and reclaim excess resource when workloads decrease. By "utility-based payment", applications in cloud only pay more when more resource is applied and used. This saves cost especially for SMBs and start-ups.

Although cloud platforms lay a solid foundation for scaling applications during runtime in term of on-demand resource utilization, the just-in-time scalability of applications is not achieved by simply deploying them to cloud platforms. There are still challenges confronting deployers.

Firstly, deployers have to be familiar with how to apply for computing resource from a cloud. Different cloud platforms offer different APIs. There is no guarantee that cloud platforms are compatible with each other. Therefore, deployers have to select a cloud platform first, followed by becoming skilled users of it.

Secondly, cloud applications scale dynamically, which implies that the utilization of cloud resource should be automated. For application servers with different scaling mechanisms, different codes are required.

Thirdly, as time goes, more and more cloud vendors appear; if cloud applications do not want to be bound to a specific cloud platform, the above efforts have to be repeated for different cloud platforms.

In this paper, we propose a profile-based approach to developing just-in-time scalability for cloud applications. In this approach, experts' knowledge of scaling application servers in cloud is captured in platform-neutral profiles. Profiles enable the automation of the setup and scaling of application servers in cloud. As a

978-0-7695-3840-2/09 $26.00 © 2009 IEEE
DOI 10.1109/CLOUD.2009.87

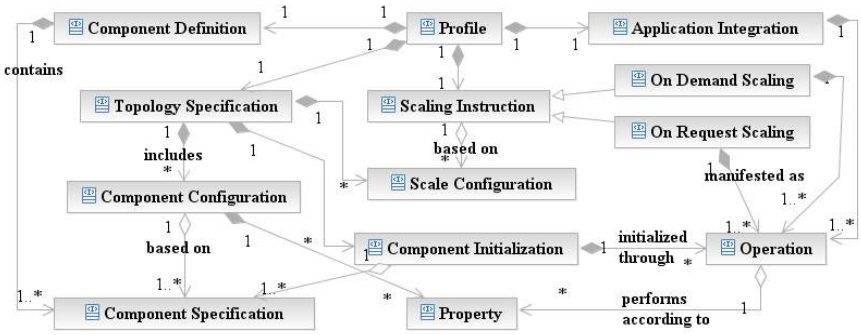

Figure 1 Profile model in UML

result, cloud applications could achieve just-in-time scalability by being deployed to scalable computing environments that are based on profiles. Besides, profiles shield application providers from underlying technologies of cloud platforms. The migration cost is thus minimized greatly.

The organization of this paper is as follows. Section 2 introduces what profiles are and details what constitutes a profile. Section 3 introduces profile-based deployment, including its process and its supporting framework, profile driver. Section 4 exemplifies profile-based deployment with a real cloud application. Section 5 is related work. Section 6 concludes this paper.

2. Profile

A profile is an embodiment of knowledge and best practice of a commonly adopted computing environment with inherent just-in-time scalability. The main purpose of a profile is to automate the instantiation of a computing environment that inherits the just-in-time scalability from the profile by straightforward configuration. Therefore, a profile declares what constitutes a computing environment, and how a computing environment is configured to enable just-in-time scalability.

The profile model (which is captured in UML as shown in Figure 1) consists of four main constituents, namely component definition, topology specification, scaling instruction, and application integration, respectively. In this section, a real profile, named ApacheTomcat, is used to exemplify how a profile is defined. This profile automates the creation of a Tomcat cluster with just-in-time scalability.

2.1 Component definition

This part of profile defines software components involved in computing environments that are created by a profile. There are two types of components. One is virtual components defined via images, the other is physical components. They form the basis of automatic setup of software constituting a computing environment.

Each virtual component is defined by an image file and the configuration file for the image. The image records the execution environment, including the operation system, the minimal and maximal memory, the hard disk capacity, required software as well as configurations from best practices. The configuration file is used to automate the provisioning of a virtual image. It contains information required for provisioning an image, including which virtual technology the images is based, where the image is located, the host operation system for the image, and so on.

Each physical component is defined as a URL and a configuration file. The URL denotes the place where all required software is; the configuration file specifies the execution environment that is required for installing software in a physical machine.

Take the ApacheTomcat profile as an example, there are two components involved: Apache [8] of version 2.0 and Tomcat [9] of version 5.5. Both components are provided in XEN images[1], which require Suse Linux Enterprise Server 10.

2.2 Topology specification

This part of profile defines how to construct a scalable topology using software components declared in the "Component Definition" part of the profile. A topology is formed by connecting software components in a proper way. Connections among software components are manifested as configurations of software components.

For example, the topology of a scalable Tomcat cluster consists of one Apache server as load balancer and several Tomcat instances to handle incoming requests. To set up a Tomcat cluster, one needs to install and configure the Apache and every Tomcat properly, including installing Apache and Tomcat instances, installing mod_jk connector to enable communication among Apache and Tomcat instances, configuring Tomcat instances as proper types of workers to handle incoming

[1] No physical components are involved in ApacheTomcat profile.

requests, creating URL mappings so that Apache knows what requests should be load balanced.

Traditionally, the whole set up process is performed manually. In our approach, each step in the set up process is abstracted and described as an "Operation" in the part of "Component Initialization" (Figure 1). Its definition enables the automation of the whole process. More detail on "Operation" is provided in Section 2.5

2.3 Scaling instruction

This part of profile defines when and how a topology scales. Scaling at proper time could lower down the high resource consumption that may lead to the crash of the existing topology if incoming requests continue increasing, or increase the utilization ratio of existing computing resource by retrieving some software components when incoming requests decrease.

There are two kinds of scaling, one is "On Request Scaling", and the other is "On Demand Scaling". "On Request Scaling" is triggered manually, while "On Demand Scaling" is triggered automatically when predefined runtime conditions are satisfied.

For example, an administrator could choose to increase the amount of Tomcat instances in a cluster when he or she observes a rapid increase of incoming requests. Alternatively, an administrator could set up some guiding conditions, such as the average CPU usage of all Tomcat instances in the cluster is over 70%. Based on these guiding conditions, the administrator could set up auto-scaling rules, such as when the above condition is satisfied, a new Tomcat instance should be started and added into the cluster.

No matter what, proper provisioning as well as configurations is required to ensure the validity of a topology. They are encapsulated as "Operation"s that could be performed automatically during runtime ("jomo:operations" in Figure 2). More detail on "Operation" is provided in Section 2.5.

To specify when scaling actions could be performed, an approach based on finite state machine (FSM) is adopted (Figure 2).

The FSM-based approach assumes that profile creators are domain experts who know very well when and how a computing environment scale. Therefore, they could identify all states that are keys to scaling. For example, for purpose of scaling Tomcat clusters, the list of possible Tomcat instance states includes "Not Available", "Started", "Stopped", "Is Starting", "Is Stopping", "Is Scaling". Based on component states, state transition rules could be specified for each scaling actions. For example, to scale out a Tomcat cluster, it is safe to add new Tomcat instances when existing Tomcat instances are all started or stopped (the "jomo:State" in "jomo:when" in Figure 2). Furthermore, scaling actions should not be performed concurrently to ensure consistency. Therefore, when a scaling action starts, states of existing Tomcat instances should be changed into "Is Scaling" temporarily ("jomo:lockState" in Figure 2). Their states would be recovered when scaling actions are finished successfully ("jomo:renewState" in Figure 2).

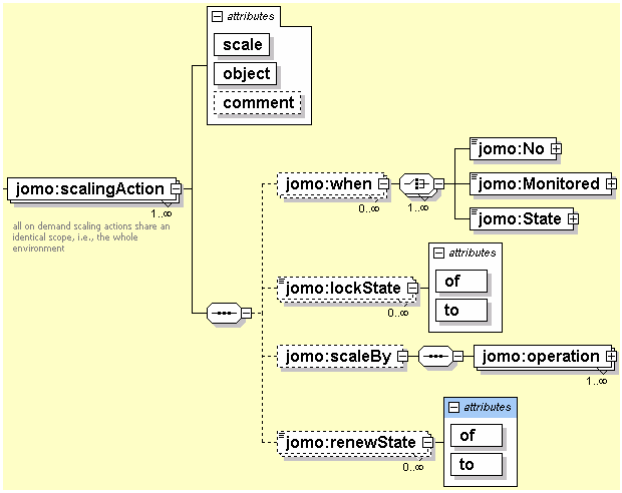

Figure 2. FSM-based scaling action definition

2.4 Application integration

This part of profile defines how to deploy application packages so that the application could leverage the just-in-time scalability offered by the scalable topology. Not surprisingly, a set of "Operations" are defined to automate the application integration.

For example, to deploy web applications to a Tomcat cluster with Apache as the load balancer, the deployer must upload web applications to the root directory of every Tomcat instances, and set up correct URL mappings in Apache.

2.5 Operation

In our profile, an "Operation" is based on an executable script, which may be a shell script in Linux, or a bat script in Windows. The purpose of "Operation" definition is to enable the automation of executing a series of scripts on different machines that are usually executed manually. Therefore, besides the path to an executable script, an "Operation" specifies the following information.

- Success indicator

If scripts are executed manually, it is easy for an expert to find out whether the script is executed successfully. However, when scripts are expected to be run automatically and in proper order, there must be indicators that manifest the successful execution of a script. By default, we could check the list of running

processes in the system to see whether a script has been executed successfully, if its execution results in a new process. Unfortunately, this is not always true for all scripts. Even worse, sometimes the appearance of a new process does not imply that the execution of the script is finished.

Considering that a script would print out hint messages during its execution most of the time[2], a string that is extracted thoroughly from the output of a script is used to denote the successful execution of a script. This approach works well so far.

- Execution location

By default, it is assumed that a script is executed locally in its containing component. However, it is also common to execute a script remotely during the provisioning or configuration of a software system. Therefore, an "Operation" can optionally specify the target component in which a script is executed.

- Execution order

By default, an "Operation" is independent of others; different "Operation"s could be executed concurrently for efficiency. In cases when an "Operation" is dependent on others, the dependencies are claimed explicitly.

In a word, an "Operation" specifies "*when* a specified script is executed *somewhere* successfully according to *what* criterion". By interpreting "Operation"s, profile driver (in section 3.2) automates the execution of a series of scripts.

2.6 Summary

Profiles formulate not only how to set up a finely-tuned computing environment, but also how to make use of cloud platforms to scale the computing environment flexibly. Our experience so far shows that the profile model is general enough to describe most typical application servers, including typical J2EE clusters, and hadoop cluster that is relatively new.

Definitely, profiles are created by domain experts who own the best practice of building up a computing environment with just-in-time scalability. The more profiles, the more chances for deployers to build an expected environment. XML-based profiles are easy to create, because many mature XML editors are out there; domain experts could use their favorite XML editors to create profiles based on the profile schema [10]. To further facilitate the creation of profiles, we design a specific profile editor that helps domain experts to (a) create a new profile based on existing ones; (b) arrange the execution order of different operations; (c) upload profiles to a profile repository, and so on. Details of profile creation are omitted due to space limit.

[2] Sometimes, the verbose mode of a script should be enabled with proper parameters to output hint messages.

Once a profile is created, it could be reused again and again. No matter how complex a profile may be, what deployers need to care is simply "key-value" style configurations. Profiles shield deployers from being lost in massive and trivial details by (a) providing a template for building a high quality computing environment; (b) embedding configurations that have been proved by best practices; (c) declaring scaling actions that hide technical details of just-in-time scalability provided by underlying cloud platforms; (d) enabling the automation of deployment for just-in-time scalability.

Guided by profiles, profile driver (in section 3.2) automates the deployment of applications to various clouds according to deployers' configuration. After the deployment, cloud applications achieve just-in-time scalability automatically without coupling with technologies of underlying cloud platforms. The profile-based deployment is presented in the next section.

3. Profile-based deployment

3.1 Deployment process

The purpose of profile-based deployment is to deploy applications to computing environments in cloud to achieve just-in-time scalability. The deployment consists of three phases (Figure 3).

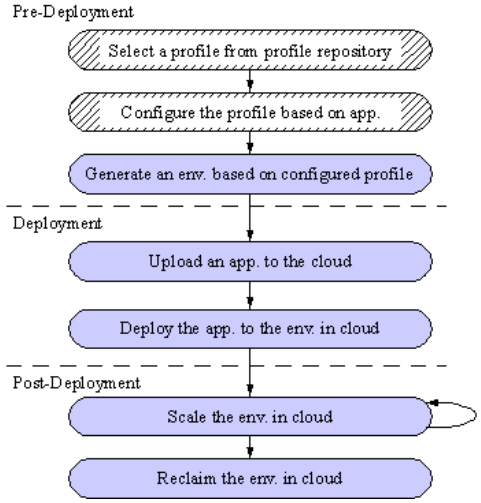

Figure 3. Profile-based deployment process

In the phase of pre-deployment, a profile-based computing environment is created in cloud. From deployers' perspective, all they need to do is to select and configure a profile. The profile selection may be based on the application's type, the killer applications of profiles (which indicate the popularity and maturity of profiles), and so on. Once a profile is chosen, the deployer could configure the profile to suit their needs. Configurable items are specific to profiles. Possible configurations

include the initial size of the computing environment (e.g. the initial size of a Tomcat cluster), the network ports that an application wishes to use, and the thresholds of scaling actions. Based on deployers' input (a profile with proper configurations), a computing environment in cloud would be generated and started automatically.

In the deployment phase, deployers simply upload applications to the cloud. Uploaded applications would be stored in some places that are unknown to deployers. After that, actual deployment starts according to operations defined in the "Application Integration" part of profiles (Figure 1). When applications are deployed and started successfully, access points to applications are returned to deployers.

Access points to applications could be disclosed to application developers or application users for testing purpose or business purpose. Then, in the post-deployment phase, the computing environments with deployed applications may scale in or out according to runtime workloads. No matter on-request scaling triggered by deployers[3], or on-demand scaling triggered by runtime situations, the change of computing environments as well as re-deployment of applications are automated under the guide of profiles. Finally, auto-generated computing environments could be reclaimed when it is decided to cease deployed applications.

The profile-based deployment is automated except the parts where deployers are involved (as indicated by rectangles in shadow in Figure 3). The whole process is enabled by our core implementation, profile driver.

3.2. Profile driver

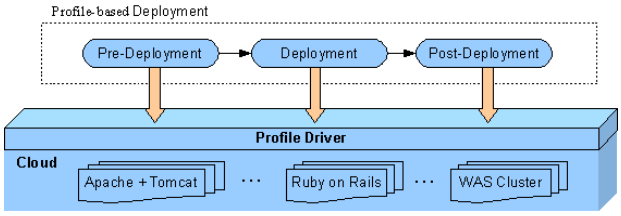

Figure 4. Role of profile driver

Profile driver is a key component in our project named Jomo.[4] Jomo aims to build a "cloud middleware" with various cloud services (e.g. queue service, log service, etc.); it hides the complexity of underlying cloud platforms, and lowers the barrier to leverage resource in clouds in a cost-effective way. In this paper, we focus on profile driver, which is a soft layer between profile-based deployment and underlying cloud platforms (Figure 4). All operations during the deployment only interact with profile driver to achieve just-in-time scalability. In other words, profile driver shields deployers from details of cloud platforms that offer dynamic resource management.

The responsibilities of profile driver include:

R1) Maintaining available profiles;
R2) Creating computing environments based on configured profiles;
R3) Leveraging computing nodes in cloud, which may come from different cloud platforms;
R4) Provisioning software (including virtual images) to computing nodes in cloud;
R5) Executing scripts on remote machines;
R6) Monitoring computing nodes;
R7) Health checking components in the profile driver;
R8) Synchronizing and maintaining critical data, including the list of computing environments in cloud, history monitored data, and so on.

To handle these responsibilities, a distributed architecture (Figure 5) is designed, which follows the following principles:

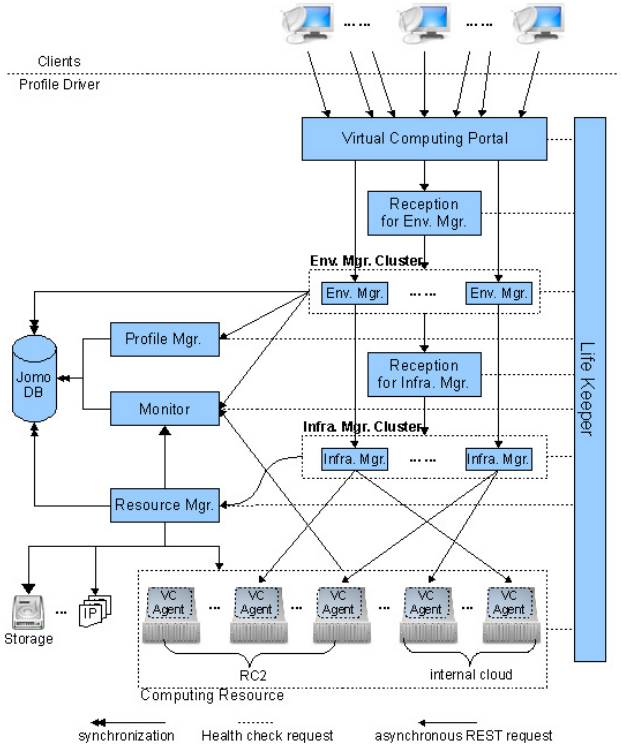

Figure 5. Architecture of profile driver

- Loose coupling

Each component in profile driver is self-contained. Components interact with each other through file-based configuration and asynchronous REST requests [11].

For example, "Profile Mgr." maintains profiles in "Jomo DB", which handles responsibility R1. "Env. Mgr." takes configured profiles as input and creates

[3] In the post-deployment phase, the involved roles are those who are responsible for maintaining applications. Here "deployer" is used to represent such roles.

[4] Jomo is the short for Jomo glangma, the Chinese name of the Everest, the highest mountain in the world.

computing environments accordingly (R2). "Infra. Mgr." operates computing nodes involved in computing environments, including installing software, configuring virtual machines. It handles responsibilities R4 and R5. "Resource Mgr." maintains all available resource from various cloud platforms, including computing nodes, storage nodes, as well as IP resource, etc[5]. It handles responsibility R3.

- Easy to scale

Profile driver aims to manage several thousands of physical machines. To adapt to such scale, both "Env. Mgr." and "Infra. Mgr." are organized as clusters (Figure 5), which could scale up by adding new nodes at runtime.

Due to the dynamic nature of "Env. Mgr.", the "Reception for Env. Mgr" is introduced. When the "Virtual Computing Portal" needs to communicate with an "Env. Mgr." for the first time, the "Virtual Computing Portal" would turn to the "Reception for Env. Mgr.", which returns an "Env. Mgr." based on a predefined load balancing algorithm and workloads of each "Env. Mgr.". After that, the "Virtual Computing Portal" would send requests from the same user to the designated "Env.Mgr.", until the designated "Env. Mgr." fails for some reason. In that case, the "Virtual Computing Portal" would turn to the "Reception for Env. Mgr." again for another "Env. Mgr." to precede the requests. Similarly, when an "Env. Mgr." needs to communicate with an "Infra. Mgr." for the first time, the "Env. Mgr." asks the "Reception for Infra. Mgr." for an available "Infra. Mgr.".

There are other components that should be scalable, including the "Monitor", the "Resource Mgr.", and the "Life Keeper". For space limit, the scalable design of these components is not presented in Figure 5.

- No single point of failure

Both "Env. Mgr." and "Infra. Mgr." are organized as clusters. When one "Env. Mgr" fails, requests would be processed by other "Env. Mgr" in the cluster. The end users will not be bothered, because data processed by "Env. Mgr" is synchronized with "Jomo DB"; each "Env. Mgr" always fetches latest data from "Jomo DB" to handle incoming requests. Similarly, other available "Infra. Mgr." would take the place of failed "Infra. Mgr.". The data used by "Infra.Mgr" comes from "Resource Mgr.", which synchronizes data with "Jomo DB" too.

Other components in profile driver have one hot backup (that is not shown in Figure 5) each to ensure high availability.

- Cloud platform agnostic

Profile driver is not bound to any specific cloud platform. Instead, profile driver manages computing resource from different cloud platforms in a consistent way. For each computing node that is under the control of profile driver, a "VC Agent" is deployed. After that, profile driver only communicates with the "VC Agent" to manage the computing node remotely.

Currently, profile drive manages computing nodes from both an internal cloud and RC2 [1]. Managing computing nodes from different clouds leads to such issues as different QoS for computing nodes, the communication between nodes in different clouds. They are beyond the scope of this paper and omitted thereof.

4. Case study

In this section, a real cloud application, named Slide River (refer to [12] for a brief introduction), is used to demonstrate how easy it is to achieve just-in-time scalability by profile-based deployment through the "Virtual Computing Portal" of profile driver (Figure 5).

Slide River is an online presentation (slide) repository, which allows sharing, searching, and editing presentations easily. It requires just-in-time scalability naturally, because as presentations in Slide River become more, so do its online users. However, during mid-nights, the online users would decrease sharply.

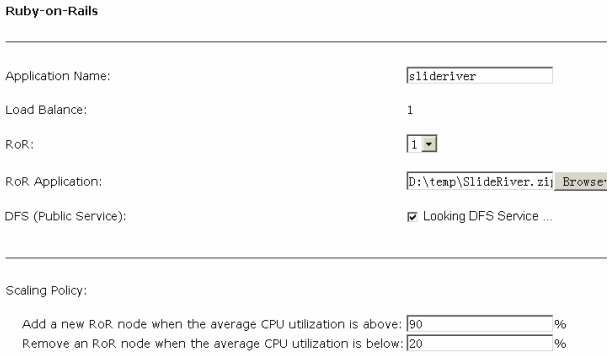

Figure 6. Profile configuration for Slide River

To achieve just-in-time scalability for Slide River, the deployer only needs to access Jomo portal to finish a profile-based deployment in three to five minutes.

Slide River is based on Ruby on Rails (RoR) [13]. Naturally, the deployer selects the RoR profile from the profile list. What follows is the configuration of RoR profile for Slide River (Figure 6). The deployer only needs to specify (1) the initial count of RoR nodes (in this case, it is set to "1" for demonstration purpose); (2) the local path to an RoR application (in this case, it is a path to the zip file of Slide River); and (3) the scaling threshold for on demand scaling (in this case, they are set to 90% and 20%, respectively)[6].

[5] This paper focuses on computing scalability; therefore, some parts of architecture involving resource other than computing resource are omitted.

[6] When DFS service is checked, the deployer would have to perform additional configurations in the next page, which is beyond the scope of this paper.

When the configuration finishes, an RoR execution platform with one load balance and one RoR node is created in our internal cloud. After that, the Slide River is uploaded and deployed automatically. As soon as Slide River starts, the access point to Slide River is returned. At this time, users could log into Slide River to view, edit or share presentations on line.

During runtime, the deployer could use the same portal to perform on request scaling (Figure 7). According to the RoR profile, on request scaling permits adding more than one RoR node at one time. Therefore, a dialog pops up, waiting for the deployer's selection.

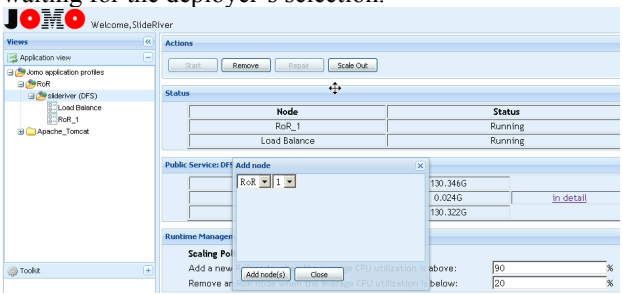

Figure 7. Scaling Slide River on request

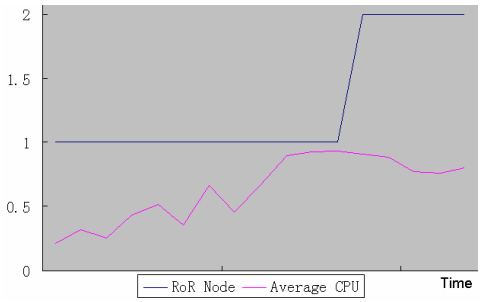

Figure 8. Scaling Slide River on demand

To demonstrate on demand scaling based on configuration, we ran a multi-threaded client that ejects random requests to Slide River continuously. These requests may be viewing or uploading randomly selected presentations, which simulate online users' behaviors. As concurrent requests increase, so does the average CPU utilization ratio of RoR nodes. When the average CPU utilization ratio exceeds the threshold of 90%, a new RoR node is provisioned and started. After that, the average CPU utilization ratio starts to decrease (Figure 8). It is also noted that there is a short time lag between the first time the CPU utilization ratio exceeds 90% and the first time the CPU utilization ratio falls below 90%, because it takes some time to add a new RoR node dynamically.

5. Related work

Just-in-time scalability of cloud applications could be achieved from facilities in different levels.

The lowest level is API from specific cloud platforms.

Amazon Elastic Compute Cloud (EC2)'s simple web service interface allows applications to obtain and configure capacity with minimal friction. It offers complete control of computing resources and lets applications run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing applications to quickly scale capacity, both up and down, as their computing requirements change [4].

Microsoft Azure is a platform for running Windows applications and storing their data in the cloud [6]. These applications must be built on the .NET Framework. In Windows Azure, each application has a configuration file. By changing the information in this file manually or programmatically, an application's owner can scale the application by setting the number of instances that Windows Azure should run [14].

Google App Engine is a scalable, fault-tolerant web application environment that lets developers run their own applications on Google's infrastructure [5]. It leverages Google's expertise in building web-scale services. App Engine applications are easy to build, easy to maintain, and easy to scale as their traffic and data storage needs to grow. Developers simply upload applications and they are ready to serve, with no servers to maintain. The biggest limitation comes from its limited programming model (only Python 2.5 is supported) and the close ties to Google infrastructure.

Achieving just-in-time scalability by using API-level facilities allows developers to control the scaling behavior of their applications. However, developers have to master a new programming model. Accordingly, applications are restricted by specific programming model and bound to specific cloud infrastructure.

To facilitate the setup of scalable cloud computing environment without binding to specific cloud infrastructure, RightScale [15] offers a higher level approach to achieving just-in-time scalability for cloud applications. It provides a design environment that developers can use to set up and manage server clusters on cloud computing providers such as Amazon EC2. Its predefined templates reduce the time and technology staff needed to set up Web applications and Web sites on cloud platforms. However, at the most fundamental level, RightScale is a configuration and monitoring dashboard for controlling servers and images on cloud platforms [16]. To setup an auto-scaling deployment environment, it still requires application developers to make sure that the inputs for templates are properly configured and that applications will boot correctly [17]. In a word, RightScale is not a total solution for achieving just-in-time scalability.

Some Related work offers total solutions for specific type of applications. By adopting such total solutions,

developers need not modify their applications to achieve flexible scalability.

Aptana cloud [18] supports scalable production environments for Jaxer and PHP. With the help of Aptana Studio, developers could upload finished projects to the cloud and scale the execution environment during runtime. The scalability is achieved by adding or reducing hardware resource (memory and CPU).

Similarly, Joyent [19] offers scalability based on Joyent Accelerators with high end configurations. Joyent Accelerators are virtualized servers (8+ cores, 32+GB RAM) with vast amounts of NAS storage. They are deployed within an ecosystem of the fastest networking and routing fabric (Force 10) and the best hardware load balancers (F5 networks) [19].

However, neither Aptana nor Joyent support on-demand scalability as defined in this paper; applications deployed by Aptana and Joyent are not capable of scaling automatically based on predefined rules.

Profile-based deployment in this paper is a total solution based on XEN [20](Other virtualization technologies may be supported in the future). Scaling knowledge of different application servers is recorded in profiles, which makes it easy for our approach to support new types of applications. For example, to support the emerging Map-Reduce applications [21], our approach only needs to create a Hadoop profile[7] that manages the scaling of Hadoop cluster during runtime [22]. When the Hadoop profile is added into the profile repository, profile driver supports Map-Reduce applications thereafter.

6. Conclusion

Cloud platforms permit on-demand resource utilization, which forms the basis of enabling just-in-time scalability for cloud applications. To facilitate the development of just-in-time scalability, a profile-based approach is proposed. In this approach, profiles capture experts' knowledge on scaling applications dynamically. Guided by profiles, profile driver automates the setup and scaling of execution environments, which ensure just-in-time scalability of cloud applications. A real project is used to demonstrate the feasibility and efficiency of our approach. In the near future, more profiles would be created, and the approach would be integrated with more cloud services.

10. References

[1] http://highscalability.com/friends-sale-architecture-300-million-page-view-month-facebook-ror-app

[2] Facebook: the "social media" revolution, http://www.docstoc.com/docs/294669/Facebook-Case-Study

[3] Robin Pinning, "Research Computing Cloud: A User Focused Infrastructure for Research Computing", JISC-Advanced Tools and Technologies for Collaborative Research Workshop, University of Manchester, Nov. 2008

[4] Amazon.com, Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/

[5] Google, Google App Engine, http://code.google.com/appengine/

[6] Microsoft, Microsoft Azure Services Platform, http://www.microsoft.com/azure/default.mspx

[7] Vouk, M.A., "Cloud computing - Issues, research and implementations", 30th International Conference on IEEE Information Technology Interfaces, June 2008, pp. 31-40.

[8] Apache Software Foundation, Apache HTTP Server, http://httpd.apache.org/

[9] Apache Software Foundation, Apache Tomcat, http://tomcat.apache.org/

[10] World Wide Web Consortium (W3C), XML schema, http://www.w3.org/XML/Schema

[11] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000

[12] Slide River, http://marketing.openoffice.org/ooocon2008/programme/wednesday_3001.odp

[13] David Heinemeier Hansson, Ruby on Rails (RoR), http://rubyonrails.org/

[14] David Chappell, *Introducing the Azure Services Platform: an early look at Windows Azure, .NET Services, SQL Services, and Live Services,* http://download.microsoft.com/download/e/4/3/e43bb484-3b52-4fa8-a9f9-ec60a32954bc/Azure_Services_Platform.docx

[15] RightScale Inc., Web-based Cloud Computing Management Platform by RightScale, http://www.rightscale. com/

[16] Jeff Cogswell, *RightScale Eases Developing on Amazon EC2,* October 21, 2008. http://www.eweek.com/c/a/Application-Development/RightScale-Eases-Developing-on-Amazon-EC2/3/

[17] RightScale Support Wiki, Autoscaling Setup, http://wiki.rightscale.com/3._How-To_Guides/How_do_I_ set_up_Autoscaling%3F

[18] Aptana, http://www.aptana.com/studio

[19] Joyent, http://www.joyent.com/

[20] Citrix Systems, Inc., XEN, http://www.xen.org/

[21] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", 6th Symposium on Operating System Design and Implementation, San Francisco, CA, Dec. 2004.

[22] Apache Software Foundation, Hadoop, http://hadoop.apache.org/core/

[7] The Hadoop profile is finished and starts to serve internal projects during the writing of this paper.