# An Approach to Service Deployment to the Service Cloud

Juha Puttonen, Andrei Lobov, José L. Martinez Lastra
Department of Production Engineering
Tampere University of Technology
Tampere, Finland
{juha.puttonen,andrei.lobov,jose.lastra}@tut.fi

*Abstract*—**Computing clouds facilitate rapid and effortless resource allocation. In particular, Infrastructure-as-a-Service clouds allow clients to dynamically lease virtual machines that behave similarly to physical servers. However, executing an application by directly using computing cloud resources is complicated and typically involves similar steps as installing and executing an application on a physical machine. Moreover, starting numerous application instances on a single virtual machine may result in poor performance. Thus, we propose developing a web service that acts as a mediator between the leased cloud resources and the cloud users and facilitates the use of the resources. When the mediator web service is used, an application can be started in a computing cloud effortlessly by invoking simple web service operations. Furthermore, in the case of several applications, the workload can automatically be distributed between several virtual machines, resulting in higher performance.**

***Keywords- cloud computing; web services***

## I. Introduction

Computing processes require hardware resources, such as processing power and data storage capacity. Traditionally, the resources have existed on physical server machines. Hence, organizations have had to invest in the purchase of the hardware as well as allocate resources in installing and maintaining the systems. Moreover, the need for computing resources tends to considerably fluctuate, causing the expensive systems to be frequently idle. Adjusting the computing resources to match the current needs is typically expensive using traditional methods. Cloud computing provides a solution to the problem by allowing organizations to lease computing resources and only pay for the amount that they actually use [1].

### *A. Previous Work*

There are different types of cloud computing. In Infrastructure-as-a-Service, IaaS, the leased resources are complete virtualised systems [2]. More specifically, the resource units leased from IaaS clouds are virtual machines [3], which behave identically to actual servers connected to the internet. However, they are created through virtualisation from actual servers. Other types of cloud computing include Software-as-a-Service, SaaS, and Platform-as-a-Service, PaaS. In SaaS, software vendors make their applications accessible over the Internet, while in PaaS the cloud systems provide platforms that allow software vendors to implement their applications. Then, the end users can access the applications over the Internet similarly to SaaS [4].

Public IaaS clouds are typically commercial enterprises from which virtual machines can be leased at certain prices [3]. The Amazon Elastic Compute Cloud, Amazon EC2 [5], is a notable example of public IaaS clouds. However, organizations can also create private clouds that are used internally and non-commercially [3]. The main purpose of private clouds is to share existing resources, rather than provide additional resources. On the other hand, private clouds may also use the resources of public clouds, and the combinations are called hybrid clouds [3].

Cloud computing toolkits, such as Eucalyptus [6] allow the creation of private clouds. While there are no standard computing cloud interfaces, the private clouds created using the Eucalyptus software framework conform to the Amazon EC2 cloud interface and can be used with the same client tools [6].

For example, companies consisting of several departments can benefit from the effortless resource allocation enabled by private IaaS clouds. If each department were allocated physical servers, those servers would be idle for considerable periods of time. While reallocating physical servers to different departments might cause considerable amount of additional work, it is straightforward to dynamically start virtual machines and attach virtual storage volumes to the virtual machines with all the necessary software and data.

In our work on semantic web services orchestration [7], we have proposed a set of web services providing a web service orchestration framework. The orchestrated web services can be hosted in resource-constrained embedded devices. In the orchestration framework, the performance issues related to memory and CPU resources can be overcome by outsourcing some of the resource-demanding functions to the cloud. Considering service oriented architecture (SOA), it would be natural for the applications deployed in the cloud to provide web service interfaces. Fortunately, computing clouds can facilitate the dynamic deployment of web services, such as those forming our web service orchestration framework. Some web services may be needed only for a limited time, after which the computing resources reserved by them should be released. Moreover, deploying the services on physical server machines might require considerable effort in configuring and installing the hardware and software. The use of cloud computing is a more feasible approach, as it allows the dynamic creation of virtual machines for hosting the web services, thus reducing the number of actual computer systems required and the amount of idle resources.

### B. *Problem Formulation*

While IaaS clouds offer more flexibility and portability than SaaS and PaaS solutions, the workload in starting an application using an IaaS cloud is considerable [8]. Although many IaaS clouds support similar interfaces, starting an application in an IaaS computing cloud may be difficult due to the low-level nature of IaaS clouds. Indeed, a client must first select the appropriate virtual machine images to use, and communication with a virtual machine instance is typically performed by logging in to the instance with a terminal program. Hence, automating the use of the virtual machine in, for example, starting new applications, would be difficult. Therefore, we propose developing a web service that is deployed on a virtual machine in an IaaS cloud and facilitates the use of the leased computing resources. The web service interface provides operations that allow starting and terminating applications.

### C. *Outline of the Paper*

The structure of this paper is as follows. Section II introduces the proposed new approach on IaaS cloud resource utilisation. Section III demonstrates the application of the approach and evaluates its performance aspects. Finally, Section IV contains conclusions and issues to be targeted in future research.

## II. Main Results

We have developed a web service that facilitates using computing cloud resources. We have named the web service Cloud Gateway because it acts as a mediator between a computing cloud and the cloud users. Specifically, one instance of the Cloud Gateway service is started on each virtual machine leased from an IaaS cloud. The service instances then enable a user to effortlessly execute applications on the virtual machines. Moreover, the Cloud Gateway services can form networks spanning several virtual machines that may reside in separate computing clouds. Thus, when a Cloud Gateway is low on computing resources, it can delegate a request for starting a new application to another Cloud Gateway instance hosted by a less burdened virtual machine.

### A. *Adding and Executing Applications*

Cloud Gateway provides operations for adding and removing applications to and from its application library as well as starting and terminating instances of the applications. Cloud Gateway assigns a unique string identifier to each application and to each started application instance. The most important operations in the Cloud Gateway service interface are described in Table I.

We have particularly considered the case where each application, when executed, creates and starts a web service compliant with the DPWS specification [9]. Thus, in the sequel, these types of applications are called server applications.

To facilitate the effortless transfer and execution of the server applications, they must be packaged into executable Java archive (JAR) files. Hence, Cloud Gateway can download the applications as single files. Furthermore, the applications can be executed on any platform that has a sufficiently new Java runtime environment installed.

An application can be executed by invoking the *StartApplication* operation and passing the application identifier as an input. Optionally, a list of command-line arguments may be specified to override the default arguments. As a response, *StartApplication* returns the identifier assigned to the new application instance or 'FAILURE' if starting the application failed.

Command-line arguments may contain keywords that Cloud Gateway expands before executing the corresponding application. Keywords are identified by enclosing them between '$#' and '#$'. For example, Cloud Gateway replaces each occurrence of the string '$#HOST#$' with the host machine network address.

A typical server application needs at least several seconds to deploy a set of web services. Web services compliant to the WS-Discovery specification [10] send *Hello* messages when they enter a network. Hence, Cloud Gateway listens to *Hello* messages originating from the host machine. Whenever Cloud Gateway receives such a message, it considers sends a *ServiceStarted* notification to all subscribed clients. The notifications allow the clients to determine server application start-up times.

Cloud Gateway allows starting multiple instances of each application. A running application can be terminated by executing the *TerminateApplication* operation. Since Cloud Gateway is able to terminate applications only in a forcible manner, the terminated applications must prepare for the abrupt termination of the underlying Java virtual machine and perform the necessary activities at such an event. For example, DPWS-compliant web services should broadcast WS-Discovery Bye messages when leaving a network. The Bye messages allow clients to automatically detect when the web services become unavailable.

### B. *Resource Consumption*

Because the amount of applications that can be started with Cloud Gateway depend on the amount of hardware resources available to the virtual machine, Cloud Gateway must use some metrics for determining the amount of free resources. Furthermore, it must compare the determined resource levels to threshold values indicating the maximum allowed resource utilisation. Cloud Gateway accepts a request to start an application only if the determined resource utilisation levels are below the maximum allowed levels.

The metrics that most clearly define the resource utilisation of a virtual machine are the random access memory (RAM) and central processing unit (CPU) usage. In Linux systems, the percentage of RAM used can be measured fairly effortlessly by examining the contents of the virtual */proc* file system. The CPU usage level is more problematic to determine, but it can be derived from the system load average, which can also be determined from the

*/proc* file system. The load average represents the number of processes that are either in execution or queuing for CPU time. Hence, the higher the value, the more burdened the CPU is. If the load average is equal to the number of CPUs, CPU utilisation is optimal [11]. To calculate a value for the CPU utilisation level, Cloud Gateway divides the system load average with the number of CPUs.

If either the determined RAM or CPU usage value is higher than the corresponding threshold value, Cloud Gateway rejects any requests to start a new application. The threshold values can be specified by invoking the *SetThreshold* operation.

### *C. Cloud Gateway Networks*

The system resources of a virtual machine will inevitably be exhausted if several application instances are executed on the machine. Therefore, Cloud Gateways residing on separate machines can form networks to balance the load between several machines. For this purpose, the Cloud Gateway service interface includes the operations *RegisterCloudGateway* and *DeregisterCloudGateway*, which allow registering and deregistering partner Cloud Gateways that will be used in workload balancing. The *StartApplicationInNet* operation will execute the application locally on the host machine only if the resource utilisation is within allowed boundaries. Otherwise, the *StartApplicationInNet* operation is recursively invoked on the partner Cloud Gateways to find one that is able to service the request. Similarly, the *SetThresholdInNet* is a recursive version of the *SetThreshold* operation.

The sequence diagram in Figure 1 represents a typical use scenario of Cloud Gateway. The *Client* object in Figure 1 can be an autonomous software agent or a software tool operated by an end user. In the beginning of the example sequence, the client registers *Cloud Gateway 2* is registered to *Cloud Gateway 1* to form a Cloud Gateway network. Then, the client registers a new server application to *Cloud Gateway 1*. Once *Cloud Gateway 1* has downloaded the application, the client executes it in the cloud by invoking the *StartApplicationInNet* operation. Because *Cloud Gateway 1* is low on computing resources, it delegates the request to *Cloud Gateway* 2, which then executes the application, effectively deploying a new web service. The response to the original *StartApplicationInNet* request includes the endpoint URI of the selected Cloud Gateway instance. Finally, the client requests *Cloud Gateway 2* to terminate the server application to release the computing resources for future use.

If a Cloud Gateway selects another service instance in the network to execute an application, it must first ensure that the other instance possesses a copy of the application and obtain the application identifier by invoking its *AddApplication* operation. This is illustrated by point 10 in the sequence diagram.

## III. An Application Example

We have tested our approach both with a private cloud created using the Eucalyptus [6] software framework and with the Amazon EC2. This section will first present the experiment setup, and then describe the test results.

### *A. The Experiment Setup*

We have set up a private cloud consisting of only one computing cluster composed of a single desktop running a

TABLE I. The Cloud Gateway service includes operations for managing the available applications as well as executing and terminating them.

| Operation | Inputs | Outputs |
|---|---|---|
| AddApplication | **location** – the URL from which the JAR file can be read<br>**parameters** – the default command-line arguments | The identifier assigned to the application or 'FAILURE' if reading the JAR file from the specified URL fails. |
| RemoveApplication | **id** – the application identifier | 'SUCCESS' or 'FAILURE' if no application with the specified identifier exists, or if a running instance of the application exists. |
| StartApplicationInNet | **id** – the identifier of the application to start<br>**parameters** – the list of command-line arguments, if empty, the default arguments will be used | The identifier assigned to the new application instance or 'FAILURE' if starting the application failed.<br>The endpoint URI of the Cloud Gateway that started the application. |
| TerminateApplication | **id** – the identifier of the application instance to terminate | 'SUCCESS' if the application was running, otherwise 'FAILURE'. |
| ListApplications | - | The list of uploaded applications. The identifier, JAR file name and default arguments are listed for each application. |
| ListAll | - | The list of all application instances. The instance identifier, application identifier, command-line arguments and state (running or terminated) are listed for each instance. |
| SetThresholdInNet | **MemoryThreshold** – a floating point value between 0 and 1<br>**CPUThreshold** – a non-negative floating-point value | 'FAILURE' if the threshold values are outside the allowed ranges, otherwise 'SUCCESS'. |
| RegisterCloudGateway | **URI** – the endpoint URI of the Cloud Gateway instance to register | 'FAILURE' if the Cloud Gateway service had already been registered, otherwise 'SUCCESS'. |
| DeregisterCloudGateway | **URI** – the endpoint URI of the Cloud Gateway instance to deregister | 'SUCCESS' if the Cloud Gateway service had been registered, otherwise 'FAILURE'. |
| GetResourceUsage | - | Numeric values indicating the amount of free memory, total memory, the number of CPUs, the system load average as well as the current memory and CPU utilisation thresholds. |

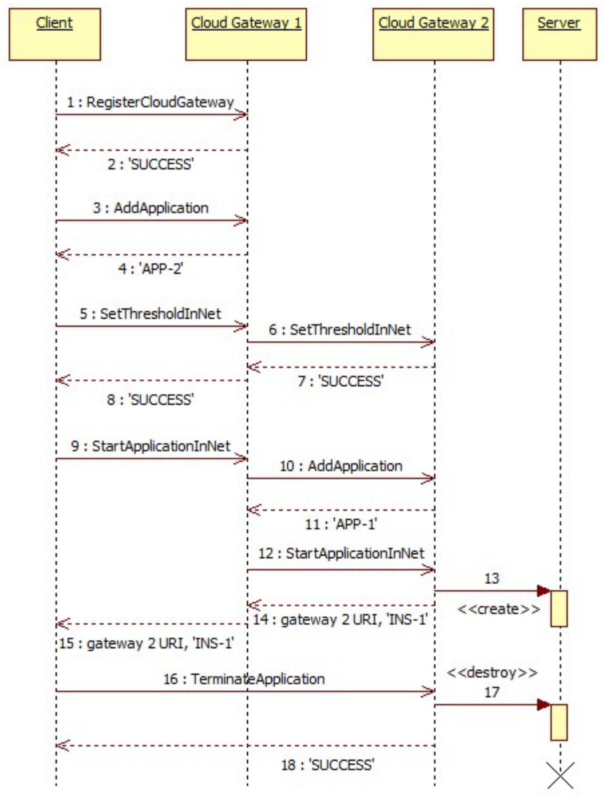

Figure 1. A typical use scenario of Cloud Gateway includes starting a web service and terminating it after use to conserve resources.

Linux operating system and Eucalyptus version 1.6.1. The restricted computing cloud limits, for example, the number of virtual machines that may be created; we have experimented with a maximum of two parallel virtual machine instances. However, even such a limited setting suffices for testing the proposed approach.

We have modified a virtual machine image so that it includes all the necessary software components for starting the Cloud Gateway server application. To upload the image to the cloud and to create virtual machines from it, we have used the Euca2ools command line utilities.

For interacting with the Cloud Gateway services, we have used our own application called Service Explorer. It includes a simple graphical user interface that allows, for example, inspecting web services and invoking their operations. In our experiments, we have executed Service Explorer on a laptop connected to the same local network as the desktop hosting the private computing cloud. Thus, Service Explorer is able to automatically detect the web services started on the virtual machines.

Each virtual machine executes a separate copy of the Cloud Gateway server application. We have deployed only two virtual machine instances in the private cloud. The experiment topology is depicted in Figure 2.

### B. *Performance Measurement*

To measure the performance of the Cloud Gateway service, we have developed a test application with a simple

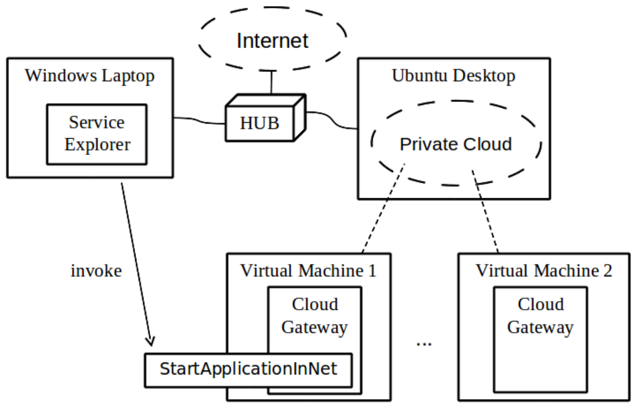

Figure 2. The test arrangement includes two physical machines, one of which hosts a private cloud containing virtual machines (VMs).

user interface. The test application assumes the place of Service Explorer in Figure 2. The purpose of the test application is to measure the time required for deploying several independent web services in a computing cloud. The test application first invokes the *AddApplication* method to register an application and then sequentially invokes the *StartApplicationInNet* operation to execute the application a number of times specified by the user. After each *StartApplicationInNet* request, the test application waits for the Cloud Gateway to send a *ServiceStarted* notification before sending the next request. The user interface includes text fields for specifying the JAR file URL and the number of times to execute the JAR with Cloud Gateway. In addition, the performance test application allows specifying threshold values, which it requests Cloud Gateway to use by invoking the *SetThresholdInNet* operation.

While experimenting with different threshold values, we have noticed that if only a RAM usage threshold were used, it should be set to at most 0.98 because the operating system never appears to let the RAM utilisation reach 99 percent but retains a small amount of memory as work space and compensates the missing memory with swap file usage. For example, with one gigabyte of RAM, the memory utilization typically reaches 98 percent after Cloud Gateway has started 28 conveyor service server applications, after which the proportion of used RAM fluctuates only marginally. However, the increased page file usage burdens the CPU, resulting in very poor performance. To prevent the CPU load from excessively increasing, a threshold value for CPU utilisation should be specified.

In one of our test runs, setting the RAM threshold to 0.99, and CPU utilisation threshold to 6.0 resulted in 42 applications being started before the CPU threshold was exceeded. Cloud Gateway started the applications in 217 seconds. However, the maximum number of applications and the start-up delay vary between different test runs. Given that the virtual machine hosting Cloud Gateway comprises only one (virtual) CPU, the load factor of 6.0 indicates that, on average, only five processes are queuing for CPU time. However, the web services, including Cloud Gateway, running on the machine seemed unable to respond to requests within the communication time out durations. Logging in to the virtual machine revealed that the load average had exceeded 60. Afterwards, the virtual machine became unreachable. Apparently, since the load average is computed over the previous minute [11], it is difficult to use it as a measure of the workload of the machine at a specific instant. On the other hand, the applications may temporarily have to queue for processing time at start-up, while later they will require less computation power.

On a virtual machine with only 256 megabytes of RAM, the memory utilisation exceeds 98 percent after Cloud Gateway has only started six server applications, and the overall delay is 22 seconds. When the memory threshold is set to 99 percent and CPU threshold is set to 6.0, Cloud Gateway starts nine server applications, but finally the virtual machine becomes unreachable.

The application start-up delay begins to increase steeply after Cloud Gateway has started a certain number of applications. This is obviously caused by the virtual machine having to compensate the lack of physical memory with page file usage. Moreover, the responsiveness of the applications running on a virtual machine is very poor when the machine is executing several applications simultaneously.

We have also experimented running the Cloud Gateway service on remote virtual machines leased from the Amazon EC2 cloud. In the experiments, each virtual machine hosting a Cloud Gateway service has been allocated 1.7 gigabytes of RAM. Table II shows the test results using a single virtual machine in the EC2 cloud. The table shows the memory threshold, number of started instances and overall start-up times. It also lists the reasons why Cloud Gateway stopped starting new server applications.

The last row in Table II represents a test scenario where the CPU threshold was set to 100. In this case, the client connection to the virtual machine abruptly terminated while starting the 94th application instance, apparently due to the excessive workload on the virtual machine.

## C. *Performance Measurement in a Network Setting*

We have performed performance tests also in a setting of two Cloud Gateways running on separate virtual machines in our private computing cloud. Each of the virtual machines is allocated one gigabyte of RAM and five gigabytes of disk space. The Cloud Gateway Performance Test application communicates directly only with the main instance. However, it invokes the *RegisterCloudGateway* operation on the main instance to add the auxiliary instance to the Cloud Gateway network. The memory and CPU thresholds set in the user interface are submitted to each Cloud Gateway in the network.

In the scenario of two Cloud Gateway instances, the main instance will serve the first application requests. However,

TABLE II. THE NUMBER OF CONVEYOR SERVICE APPLICATIONS THAT CAN BE STARTED ON A VIRTUAL MACHINE WITH 1.7 GB OF RAM.

| Memory threshold | Reason for termination | Number of instances | Total duration (s) |
|---|---|---|---|
| 0.9 | memory | 28 | 138 |
| 0.98 | memory | 33 | 173 |
| 1 | CPU > 50 | 85 | 652 |
| 1 | failure | 93 | 1379 |

once it exceeds the memory threshold, the main instance starts delegating incoming application start requests to the auxiliary instance.

For example, in one of the test runs, the memory threshold of the two Cloud Gateways was set to 98 percent, while the CPU threshold was set to five. Finally, the performance test application started requesting the main Cloud Gateway instance to start instances of the conveyor service server application. The main instance exceeded the memory threshold after starting the 28th server application and started delegating the requests to the auxiliary instance on the other virtual machine. The auxiliary instance was able to start 27 applications before exceeding the memory threshold. Hence, a total of 55 application instances were started, and the total duration was approximately 210 seconds.

## D. *Inter-Cloud Experiment Scenario*

To experiment web service orchestration across different computing clouds, we have performed an experiment with two remote virtual machines and one local virtual machine. The remote virtual machines are leased from the Amazon EC2 cloud, while the local virtual machine is running in our private computing cloud. Each virtual machine hosts one Orchestration Engine web service and three virtual conveyor web services.

The experiment consists of a cycle that starts when the Orchestration Engine on virtual machine 1 is requested to execute a BPEL process orchestrating the three conveyor web services. In the end of the process, the Orchestration Engine on virtual machine 1 requests the Orchestration Engine on virtual machine 2 to execute a similar process, which is represented by step 4 in Figure 3. Then, the Orchestration Engine on virtual machine 3 executes a similar BPEL process, which finally requests the Orchestration Engine on virtual machine 1 to again execute the BPEL process (step 12). Hence, the cycle continues indefinitely, so that only one Orchestration Engine is executing a BPEL process at a time.

To measure cycle durations, a client application monitors the Orchestration Engine service on the local virtual machine. Each time the Orchestration Engine sends a

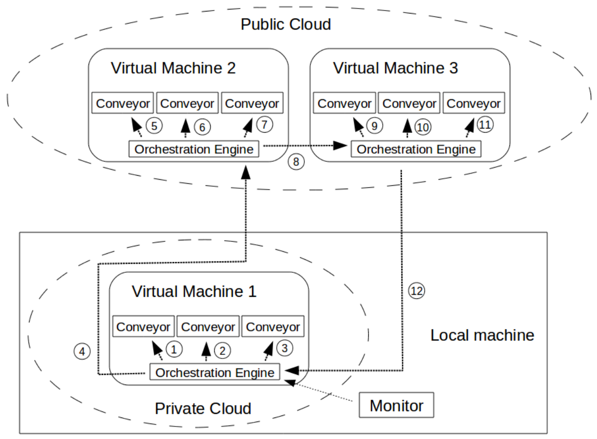

Figure 3. The private cloud is hosted on a local machine. Each virtual machine hosts four web services.

notification signalling that it has begun executing a BPEL process, the client application records the duration of the elapsed interval since the previous notification. It also determines the minimum, maximum and average interval length. The experiment topology is depicted in Figure 3. We have studied the use of BPEL in web service orchestration in [12] and described the Orchestration Engine web service in [7] and [13].

Table III contains the experienced minimum, maximum and average intervals in an experiment consisting of 20 cycles. To obtain a reference point, we repeated the experiment so that all of the web services were running on the local machine. As Table III shows, the average cycle duration is approximately two seconds longer when using computing clouds. This constitutes less than five percent of the average cycle time. The minor performance degradation is presumably caused by the network traffic between the web services on different virtual machines. However, network traffic is unavoidable when the Orchestration Engine services execute on different machines.

## IV. Conclusion And Future Work

In this paper, we have proposed a web service that facilitates the use computing cloud resources. The method allows the use of computing cloud resources without knowledge on the cloud interface or internal composition. In particular, we have shown that a client can use the cloud resources by invoking simple web service operations without having to directly interact with the leased virtual machines.

A current limitation of the proposed approach is that cloud resources are somewhat inefficiently used. While it is possible to create a network of Cloud Gateway services running on separate virtual machines, the machines must be leased in a static manner, before launching the corresponding Cloud Gateways. Cloud Gateway could be enhanced so that it dynamically created new virtual machines as the utilisation of the existing ones reached a certain level.

We have carried out experiments to evaluate the performance of the proposed approach. While the automated execution of applications is effortless, the resource limits of the underlying virtual machine are eventually reached as the number of executed applications increases. Moreover, exhausting the resources over a certain point tends to considerably decrease application responsiveness. On the other hand, we have shown that forming networks of several Cloud Gateway services allows automatically balancing workload between several virtual machines.

Cloud Gateway measures the percentage of used memory and the system load average to avoid the overuse of computing resources. The method appears effective in preventing severe performance degradation when starting several applications. However, currently, Cloud Gateway is unable to estimate the amount of resources an application will consume once it has been started. Therefore, future research should target the implementation of a mechanism for evaluating the runtime resource consumption of the started applications.

In addition, we have experimented with web service Orchestration spanning separate computing clouds. The results suggest that using computing cloud resources causes no considerable performance drawbacks. However, deploying several web services on separate virtual machines requires manual work. Future research should investigate the use of Cloud Gateway in automating this task.

TABLE III. The duration of 20 cycles is quite similar regardless of whether cloud resources are used instead of local resources.

| | **Minimum (ms)** | **Maximum (ms)** | **Average (ms)** |
|---|---|---|---|
| 1 local VM, 2 remote VMs | 47471 | 49106 | 47961 |
| Only local web services | 45519 | 46787 | 45682 |

## References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A Break in the Clouds: Towards a Cloud Definition,“ ACM SIGCOMM Computer Communication Review, Vol. 39, Issue 1, pp. 50–55 (2009).

[2] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, “Sky Computing,” Internet Computing, IEEE, Vol. 13, Issue 5, pp. 43–51 (2009).

[3] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual Infrastructure Management in Private and Hybrid Clouds,” Internet Computing, IEEE, Vol. 13, Issue 5, pp. 14–22 (2009).

[4] G. Lawton, “Developing Software Online with Platform-as-a-Service Technology,” Computer, IEEE, Vol. 41, Issue 6, pp. 13–15 (2008).

[5] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, Referenced on 23.08.2010.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, Y. Lamia, and D. Zagorodnov, “The Eucalyptus Open-Source Cloud-Computing System,” Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 124–131 (2009).

[7] J. Puttonen, A. Lobov, M. A. Cavia Soto, and J. L. Martinez Lastra, “A Semantic Web Services-Based Approach for Production Systems Control,” The Cognitive Factory special issue of the journal of Advanced Engineering Informatics, September 2010, in press.

[8] A. Sheth and A. Ranabahu, “Semantic Modeling for Cloud Computing, Part I,” Internet Computing, IEEE, Vol. 14, Issue 3 (2010).

[9] Devices Profile for Web Services Version 1.1, http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html, Referenced on 23.08.2010.

[10] Web Services Dynamic Discovery (WS-Discovery), http://schemas.xmlsoap.org/ws/2005/04/discovery/, Referenced on 23.08.2010.

[11] R. Walker, “Examining Load Average,” Linux Journal, Issue 152, Dec. 2006, http://www.linuxjournal.com/article/9001, Referenced on 20.09.2010.

[12] J. Puttonen, A. Lobov, and J.L. Martinez Lastra, “An Application of BPEL for Service Orchestration in an Industrial Environment”, IEEE International Conference on Emerging Technologies and Factory Automation, pp. 530–547 (2008).

[13] A. Lobov, F. Ubis Lopez, V. Villaseñor Herrera, J. Puttonen, and J. L. Martinez Lastra, “Semantic Web Services Framework for Manufacturing Industries”, IEEE International Conference on Robotics and Biomimetics, pp. 2104–2108 (2009).