Simplified cloud-oriented virtual machine management with MLN

Kyrre Begnum
kyrre.begnum@iu.hio.no
Oslo University College, Norway

March 7, 2010

Abstract

System administrators are faced with the challenge of making their existing systems power-efficient and scalable. Although Cloud Computing is offered as a solution to this challenge by many, we argue that having multiple interfaces and cloud providers can result in more complexity than before. This paper addresses cloud computing from a user perspective. We show how complex scenarios, such as an on-demand render farm and scaling web-service, can be achieved utilizing clouds but at the same time keeping the same management interface as for local virtual machines. Further, we demonstrate that by enabling the virtual machine to have its policy locally instead of in the underlying framework, it can move between otherwise incompatible cloud providers and sites in order to achieve its goals more efficiently.

1 Introduction

Infrastructure as a service (IaaS) is becoming attractive for both researchers and technicians with the emergence of products and tools surrounding on-demand deployment of virtual machines. The main principle is that you can get computing resources (Xen-based virtual machines in the case of Amazon EC2) without investing in or setting up hardware, thereby reducing cost and power.

What most understand as "Cloud Computing" compared to a conventional IaaS product is the existence of an API to manage virtual instances directly. Instead of a permanent setup of instances which are rented over an contracted period of time, the users create new instances themselves and tear them down when they are obsolete. Cloud Computing is therefore more transient and flexible than a conventional IaaS. This property of minute-to-minute influence over the number of resources used, fits very well with the ideas of dynamically adaption of services. Virtualization, Iaas combined with an API, work as an enabler for the development of new types of infrastructures.

Being a system administrator in this era means facing a new set of management challenges. Before, it was about simply keeping systems and services running and getting enough permanent resources to survive a usage spike. Today, we are expected to *program* behavior into our systems so that they adapt and behave according to complex policies. This new paradigm means having to give our systems self-* properties, like self-configuration and self-scaling. A likely scenario is to leverage an increase in distributed server resources for temporary extra computing power or proximity to customers. One of the main challenges in achieving this from a local perspective, is the lack of tools which address both in-house virtual machine management and cloud computing through the same management interface. This is not merely a problem of learning yet another tool, but that of integrating the clouds instances into existing, local infrastructure systems, such as monitoring, user management and backup. We argue that cloud computing should be integrated into the existing management tools and practices at an organization and not having to split up management operations to span multiple provider-centric interfaces.

Consider a small company producing CGI (computer-generated imagery) material which may not afford to buy and maintain their own render farm. Hiring a render farm for fixed periods of time would be the alternative albeit somewhat inflexible if they fail to utilize the farm for the entire period they paid for. Cloud computing may offer a better solution where the render farm exists in a cloud such as Amazon EC2 and can be booted up only when needed. The local technicians can maintain a local version of the cluster, running entirely as virtual machines on a moderately powered computer, providing a testing ground for their software. Once they need to render a large project, the cluster is pushed into the cloud and booted up as a far more powerful version of itself. When the rendering is finished, it can be taken down again and there are no more costs associated with it.

Another example is that of a company with local web services experiencing a sudden heavy load due to increased attention from customers. Buying and installing new hardware in order to cope with the pressure has numerous drawbacks: The time to get them up will be too long and once they are up they were a wasted expense if the traffic goes down to its normal level again. Further, although they can expand their infrastructure, there might be little they can do with the capacity of their internet connection. Again, with cloud computing they could boot up web-server instances in different clouds and use load-balancing on the DNS service to direct the customers. Once the traffic dies down again, the instances can be taken down. If the load settles on a high level, the company now has time to invest in more hardware and network equipment based on their expected profits.

There are some observations to be made about these scenarios. The most important one is that the companies all have local infrastructure and the capability to run virtual machines. Further, they are cloud-aware and have the means to effectively manage instances both locally and in a cloud if necessary.

As a result, they have designed their service to work with support from cloud providers when needed. Seeing cloud computing as an *addition* rather than only a replacement for local infrastructure offers many more scenarios like these.

This paper describes an effort to address the problem of cloud-aware local management by modifying the virtual machine management tool MLN to integrate cloud architectures and thereby enable seamless integration of cloud instances into local management. We showcase its usefulness today through two scenarios, an on-demand render cluster and a scaling web-scenario. Possibilities for self-management are demonstrated in a third scenario to highlight the future potential of virtualization and cloud computing.

2 Background

Amazon's elastic computing cloud (EC2) is a Xen-based IaaS product which has become a popular example of cloud computing. Users can upload custom virtual machine images and boot instances of them according to five different hardware profiles. There are several cost-metrics involved in calculating the running costs for a virtual machine, based on added services such as permanent storage partitions and network traffic. However, the most important observation is that cost is dominated by instance uptime and not CPU usage per se. The more powerful the instance, the higher the cost associated with keeping it up and running. A cost-effective utilization of Amazon's product means making sure that the instances only running when they are needed. This highlights the new paradigm of system administration where *behavior* becomes the focus rather than uptime. When the behavior is resource scaling, the common approach is to find an algorithm for resource scaling based on the usage levels. The work done in [15] is a good example of this, even though they implemented their cluster in a data center and booted up physical machines instead of virtual machines. In [16], the cost / value function is investigated in a resource scaling context. Here, the load balancer would seek to optimize the resource level entirely based on the cost for each resource unit compared with the reported value from the users. Other, traditional variables such as connection rate and system load are not used, thereby reducing the need for elaborate monitoring and representation of knowledge.

An available API to the EC2 cloud enables other developers to build value-added products based on Amazon's infrastructure. Tools such as the firefox plugin Elasticfox or sites such as RightScale.com provide different entry-points to the same cloud computing product. The drawback of these tools is that they only deal with virtual machines in the cloud and not local virtual machines. Further, they are built as graphical interfaces which encumbers automation.

An open source alternative to Amazon EC2 is Eucalyptus, a project located at the University Of California[1]. Eucalyptus is appealing because it supports the same API as Amazon EC2 and can work as a plug-in replacement for Amazon's commercial offering. This may be more attractive to research institutions, where sharing resources for experiments has been widespread. One example of

such practice is PlanetLab.

One particular feature of Amazon EC2 and Eucalyptus alike is that the changes made to a running virtual machine are not stored in the filesystem which was originally uploaded. Rebooting the instance will lead to all changes being lost. From a system administration point of view, this creates extra challenges as systems are rarely satic. Examples of activities which change the system are user management (adding, deleting, changing passwords and permissions) and log files. Re-uploading a large filesystem just because one changed the permission to a file is cumbersome as well as not being able to investigate the log-files after a crash. One is quickly faced with the need to create a local staging area in order to fine-tune the virtual machine before it is moved into the cloud. This makes it increasinlgy important to combine local virtual machie management tools aware of clouds and facilitate this process. Amazon's attempt at mitigating lack of permanent storage is to use so-called EBS volumes, which are block devices which offer permanent storage and can be attached to running virtual machines. Although this in theory makes it possible to have persistent data, it is more difficult in practice due to a complicated naming scheme of volumes and instances which leaves most tools unable to automate much in this regard.

An example of IaaS being used in order to enhance local infrastructure can be found in the VMplant project [13]. Here, virtual machines running on a remote infrastructure could be connected to the local area network using tunneling. The goal was the ability to dynamically expand the number of local servers. Utilizing clod computing in order to optimize resource allocation to the current demand has seen great interest as of late. Examples of such can be found in [9, 10, 11]

2.1 MLN

MLN (Manage Large Networks) is an open source tool designed for management of large numbers of virtual machines. A concept of groups of virtual machines called "projects", enable atomic management operations such as building, starting and stopping entire clusters and networks. An expandable plugin framework to allow additions to MLNs configuration language and features.[14]

MLN uses a configuration language to specify how the virtual machines are to be set up. In this language both VM properties such as memory and disk size are specified along with internal system properties such as users, passwords and startup parameters. MLN supports Xen, User-Mode Linux and VMware Server and has previously been shown to work well in scenarios where large numbers of virtual machines are deployed with complex network topologies and system configurations, such as high-performance computing and virtual labs in education. [12, 5, 6]

Typical management tools in this field are GUI-based, with little possibility to group together virtual machines or correlate configuration parameters accross them. Further, scripting of complex operations, such as moving groups of virtual machines from one location to another based an initiative taken by the virtual machine itself is not possible with standard vendor management tools. In MLN,

simplified management is achieved through atomic management commands. The user can build, start, stop and upgrade large and complex scenarios, consisting og multiple, logically grouped, virtual machines spread over several physical servers, using only single commands. This type of management fits well with scenarios like the ones described above and allows for scripts and higher level tools to utilize these commands in order to achieve their goals.

2.2 Integrating Amazon EC2 and Eucalyptus

Both Amazon EC2 and Eucalyptus support by-and-large the same API for virtual machine management. For MLN, integrating support for these two cloud providers is done through the development of a plugin, which can be maintained apart from the main MLN codebase. This plugin seamlessly expands the syntax of the MLN configuration language, allowing users to include Amazon EC2 or Eucalyptus properties in the virtual machines they design. The following is a simple project consisting of only one virtual machine which is to run inside the Amazon EC2 cloud. The `ec2`-block inside the host specification is what is handled by the plugin and results in this virtual machine being built in the cloud.

```
global {
       project ec2example
}
host webserver {
       xen
       ec2 {
           type c1.medium
           volumes {
              2G hda1 /var/log/apache ext3 defaults
           }
       }
       network eth0 {
           address dhcp
       }
       template web.ext3
       free_space 2000M
}
```

Notice how volumes can be specified in an integrated manner, allowing MLN to coordinate which volumes belong where, letting the virtual machine always be associated with that particular volume. Associating volumes with virtual machines is of particular importance when projects of numerous virtual machines are run, where each virtual machine has its own volume. Keeping track of the volumes so that the same volume is always associated with the same virtual machine is something which is not possible with even Amazon EC2's own management console.

The management commands stay the same, which enables the user to keep managing virtual machines the exact same way regardless if they are local or in a cloud. It is even possible with projects containing both local virtual machines and cloud instances and subsequent management (starting, stopping, building, removing) of these as an atomic unit:

```
mln build -f projectfile.mln
mln < start |stop > -p projectname
```

In order to cope with long-term management of projects, MLN provides an "upgrade" command which allows modifications to running virtual machines. When upgrading, a new version of the project configuration file is submitted and MLN will attempt to enact the changes based on the difference from the previous version. The type of modification can be hardware-oriented, such as increasing disk size or memory. It can also be used to migrate the virtual machine from one server to another, so-called live migration, where it is supported. It is this upgrade mechanism which allows system administrators to migrate virtual machines which are running locally into Amazon EC2 or Eucalyptus. When doing so, the new version of the project needs to have an `ec2`-block inside each virtual machine which is to be migrated to a cloud. Another possibility is to add virtual machines to a running project, which leads to the possibility of a scaling site where the number of servers can be increased on demand. The command to upgrade an existing project is: `mln upgrade -S -f newprojectfile.mln`

3 Case: On-demand render farm

In this scenario, we want to demonstrate how MLN can be used to create and manage a large cluster of virtual machines which can reside either locally or in a cloud. The case, as highlighted above,is that of a company in need for a render farm for short periods of time. The technicians maintain a local renderfarm running on a moderately powered computer, such as a modern desktop machine. The following project represents the design of the render farm in its local version:

```
global {
       project render
       autoenum {
                numhosts 5
                superclass rendernode
                hostprefix rend
       }
}
superclass common {
           xen
           memory 512
           free_space 1000M
           network eth0 {
              address dhcp
           }
           template rendernode.ext3
}
superclass rendernode {
     superclass common
     startup {
             /scripts/rendernode.pl start
     }

}
host frontend {
     superclass common
     template manager.ext3
     startup {
             /scripts/frontnode.pl start
     }
}
```

Note that some additional parameters which would have been useful, such as users and passwords have been omitted for clarity. The project consists of a

frontend virtual machine and a number of render nodes which all inherit from the `rendernode` superclass. The number of render nodes (in this example 5) is defined in the `autoenum` block, which removes the complexity of adding more nodes by acting as a for-loop at parse time[6]. The project file itself will not grow based on the number of nodes, so it could just as well be 32 nodes without changing any of the management steps. The virtual machines are built from templates, which are ready-made filesystems containing the render software and libraries. The frontend node contains queueing software in order to partition the job and manage the nodes. Building and starting the cluster is done by issuing management directives on the project-level:

```
mln build -f render.mln
mln start -p render
```

Until now the render farm has been running locally while the technicians have adjusted the software to work properly with their local applications. With a low memory setting per node, this could run as local virtual machines on a moderate desktop computer using the Xen hypervisor[2]. When it is time to run the render farm as a more powerful cluster, a cloud provider will act as extra infrastructure for the required period. The next step for the technician will be to move the cluster into the cloud. This is achieved by using the MLN upgrade command. First, a new version of the project file is written with the following addition to the superclass `common`:

```
ec2 {
     type c1.xlarge
     volumes {
        2G hda1 /var/log
     }
}
```

By adding the `ec2`-block to the superclass, all virtual machines inheriting from it will now be instances running in a cloud environment. This could be either Amazon's EC2 framework, as the name suggests, or an Eucalyptus cloud provided by another party. There are two directives in this block. The first is a type assignment, which will boot this VM as the most powerful hardware allocation in terms of CPU; 8 cores and 7GB of memory. The `volumes` block will assign each instance a permanent partition which will be mounted on the `/var/log` folder for logging. This is useful since the EC2 frameworks do not offer permanent storage of changes made on the virtual machines while they are running. The changes to the project are enacted using the following command:

```
mln upgrade -f render_ec2.mln
```

MLN will now bundle and transfer the images necessary to boot the project to the cloud environment and automate the typical steps of registering the images, creating the permanent volumes and assigning volumes to each instance. Note, that EC2 frameworks can boot several virtual machines from the same image, so in this case we only need to transfer two compressed images, one for a render node and one for the frontend. This holds regardless of the size of the render farm. Once the upgrade process is completed, the project can be started with the same command as above on the same machine. Having 16 render nodes of the largest type, would provide a 128 CPU cluster, which they in terms of management can boot up and shut down as they like just like a group of local virtual machines.

It is also possible to run a secondary local copy of the cluster for updates and then push it out again into the cloud for actual production. This can ease lack of permanent storage of the root filesystems for both EC2 and Eucalyptus.

4 Case: Scaling web service

The ability to dynamically de- or increase it's number of resources, be it for performance objectives or power-management, has been the focus of many in the wake of virtualization's attention the last years. For most organizations today, however, the technology is unavailable to them unless they invest heavily in specialized products. Further, automatically scaling is not trivial, as it has the potential to cost excessive amounts if the algorithm reacts to a false positives or deliberate denial-of-service attacks.

In this scenario, MLN is used to add or remove nodes in a cloud belonging to a web service. The service consists of a load balancer with the ability to dynamically add new nodes to be used as webservers. The system administrator can decide how many webservers to use through MLN upgrade command to the project.

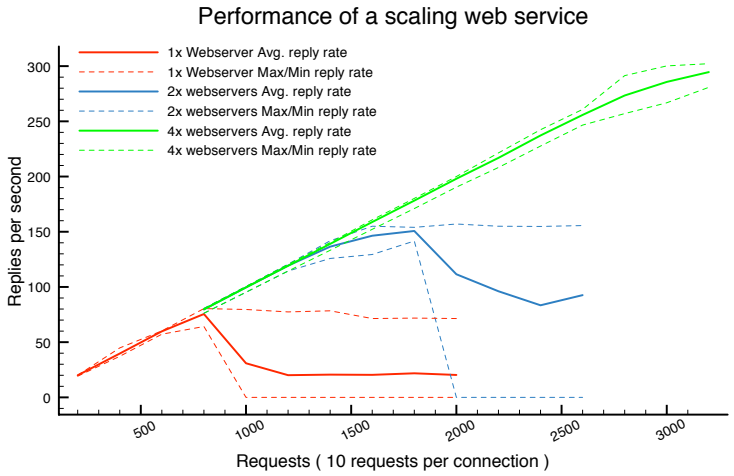

Figure 1: The performance of a scaling website when using 1, 2 and 4 webservers running on the Amazon EC2 framework

Using a design similar to the render farm, a project consisting of the frontend loadbalancer and a number of webservers is declared. Different versions of that project would only differ in the number of webservers. Changing between them could then be enacted using the MLN upgrade command. Each of these commands would result in the specific number of webservers being set and can be run in arbitrary order:

```
mln upgrade -S -f webservice_x1.mln
mln upgrade -S -f webservice_x2.mln
```

```
mln upgrade -S -f webservice_x4.mln
```

The figure 1 shows the performance of the website based on the number of webservers running. The website was running on the Amazon EC2 cloud and the loadbalancer used was perlbal. The performance data was gathered using httperf. We see that the ability to withstand a high request rate increases with the number of backend webservers. The maximum number of connections per second for a single webserver seems to be around 900. As the number of webservers is increased, we see that the performance w

One would ask why it would be desirable to keep the website running with only one webserver when it could run with four. The answer is in essence what cloud computing is about. A system administrator, with proper monitoring, will be able to keep the site at a low-cost, low-power performance level during periods of *known* inactivity, like night-time. During the day, a scheduled job could increase the number of servers to two and four servers could be kept as a short peak-hours remedy. What is important in this scenario, is that local management commands can be used to manage cloud-based systems like we would traditionally script system behavior, and that the mechanism to control cloud-based systems is separated from the policy (or algorithm), enabling local solutions.

An example of a local solution would be the implementation of automated scaling when MLN is combined with decision making component. In the following case, the load-balancing frontend continuously monitors the incoming rate of connections. It is asssumed that the performance of a single webserver is known from benchmarks and monitoring. This *quota* can then be used to deduct the desired number of webservers which should be running relative to the current rate. The avreage of a sliding window of the last 60 seconds is used for decision making.

The frontend adjusts the number of webservers through the MLN command and sends new versions of the project to the infrastructure it is running on. The average rate is compared to the number of webservers every 5 seconds, however a grace time of 15 seconds is enforced after a change is initiated in order to allow the previous command to finish. In the experiment the web-farm runs in Amazon EC2. The incoming rate of connections varies between 0 to 130 per second in steps, similar to the work in [15]. A maximum of 7 backend webservers is allowed. The webpage requires the webserver to read a file and sort its contents of 4096 bytes, simulating a site with a majority of static content but some server-side processing. A client timout of 5 seconds is used to signify an *acceptable* service level. Each webserver is found to be capable of sustaining a maximum of ca 30 connections per second. The qouta is set to 25 per second in order to allow a little over 80% utilization of each webserver.

The traffic is generated through a series of httperf sessions, varying both the rate and length of each session. The rate changes quickly from 0 to 130 over a few minutes before it declines more slowly down to 20. This part investigates the ability for the load blancer to adjust itself and follow a slow-moving trend. A short burst at the end for only 30 seconds will test how the front-end copes

with sudden and short-lived bursts. The entire experiment is approximately 18 minutes long.

Data is collected both on the front-end and the traffic generator. Important values to observe is the number of lost connections during times of pressure and before the cluster has had time to adapt. Further the rate at which changes can be enacted will impact the overall result. Figure 2 shows the results of one iteration of the experiment.

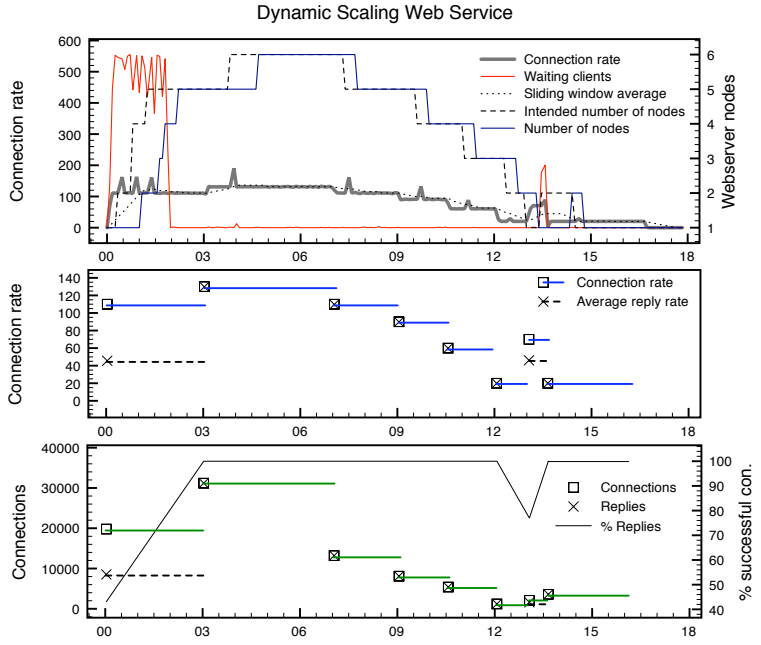

Figure 2: The plots show the dynamic scaling of webservers in a webfarm based on the incoming rate of connections. Each server has a qouta of 25 conenctions per second. The decision is made based on a sliding window og 60 seconds.

The experiment was repeated 20 times. Each time a total of 84600 connections were made to the webservice. An average of 87% were successful connections. Most of the lost connections happen during the two large surges, one at the beginning and at 13 minutes where the connection loss was average 47% and 78% respectively.

The flexibility of the service depends on the rate at which instances can be added or removed from the cluster. The results show that the loss of service is only present while we wait for the new instances to become operational. The experiments showed low variation from the average of 53.5 seconds from the decision was made to increase with a single virtual machine until that virtual machine was registered with the load balancer. Interestingely, in the cases where

two isntances were added in the same decision, they could be booted up at the same time. Now the average was only 61 seconds until both instances were in use. This indicates that the major factor is the boot time of the virtual machine.

5 Self-management with local policies

This part demonstrates how cloud computing can play a role in development of advanced system policies based on ideas from agent-based behavior. A virtual machine, compared with a physical and conventional system, has the ability to be modified in ways which resemble that of traditional agents. They can increase in computing power and memory while powered on or even move between locations either while running or powered off. Virtualization provides the flexibility for agent behavior. What is needed is the individual VMs ability to enact behavior based on its own local goals.

The traditional approach of managing virtual machines is to build a framework which handles and monitors all the virtual machines combined with an algorithm for balancing the virtual machines across the physical machines. However, this approach has some drawbacks when seen from the perspective of the user:

- Cloud providers are unaware of each other, while the users more likely will have virtual machines in multiple clouds in the future. It is difficult to get homogeneous treatment of your virtual machine across the different clouds, which complicates coordinated management of all resources.
- Load balancing algorithms are very basic and limited to typical services like web applications. The customers can not decide which factors to prioritize, like accepting poor performance in order to keep costs down.
- Interfacing with a cloud is often done through a graphical application intended for humans. It is difficult to program your own algorithm on top which would give your virtual machines a specialized policy across multiple cloud providers.

The decoupling of high-level management algorithms from the infrastructure is a necessary step in order to let the virtual machines manage them selves. We will next show two examples where a virtual machine will be aware of multiple infrastructure providers and make decisions locally as to where it should be located based on its own policy. Further, we show that this can be achieved with conventional tools common in the field of configuration management.

5.1 Service optimization through user proximity

In [3] we let the virtual machine run a web-service and monitor the origin of its users. We introduced three location where the virtual machine could move freely based on its own local decision. The location were one in the United States and two in Norway. In order to achieve the best service to the majority

of its current users, the virtual machine decided where to stay based on a short history of incoming user requests. By introducing curve-based behavior at each location relative to local working hours, we observed that the virtual machine moved back and forth between Norway and The United States.

The virtual machine was able to change its own location by having a local version of its own MLN project and by the ability to communicate changes down to the framework using the MLN client and issuing an upgrade with a new version of its project. The framework itself is passive and does not do any load balancing or optimization itself. The decision is made entirely inside of the virtual machine, based on its own policy. The configuration management tool Cfengine[7, 8], was used to monitor the clients, analyze the usage and make the decision to migrate. Dynamic DNS is used to offer the same address to the clients regardless of the virtual machines current location.

The plot below shows the result from migrating a virtual machine between two norwegian cities, Oslo and Gjøvik, with induced difference in time-zones. The virtual machine uses a simple policy to decide where to stay: The desired location has more than 60% of the total current clients and their trend is rising in the last 15 minutes. The dotted line represents the threshold of 60% while the line around it depicts the current percentage of the location with the most users. The two sine curves show the activity from the two cities, normalized to fit the plot. The round-trip time, as seen from Gjøvik, is shown in the bottom line. The results show that the simple policy migrates the machine to Oslo at the beginning of their working hours. The RTT-value is lowest as seen from Oslo. Gradually, the activity increases at Gjøvik, and when Oslo is past its peak, the majority shifts and the virtual machine moves to Gjøvik. We see then that the RTT-value becomes low for Gjøvik for the remainder of their working day.

It is important to note, that the input data is something only the virtual machine can see at all times. It would be possible to see the traffic from the framework, but if Oslo and Gjøvik would represent two different providers, chances are small they would share monitoring data.

The policy used to migrate is simple but demonstrates our point that policies inside of the virtual machines can mimic those usually found in frameworks with basic resource balancing. We see from the RTT value that at some migrations, it takes time for the virtual machine to *settle* at its new location. This is because we had a level of noise in our load generators, which could impact the majority when it was still very even. However, more advanced policies with better trend analysis can take advantage of the same mechanism to use MLN in order to enforce its local policy.

In an addition to free movement between the cities, the virtual was given the opportunity to add extra local disk devices when it desired to. This was utilized in order for the virtual machine to backup its data whenever it was at a location it considered to be trustworthy. This type of behavior fits well with the design of popular clouds today. Amazon EC2 offers storage volumes called EBS volumes, which mimic the exact same behavior. The EBS volumes are localized and are therefore only available to instances in the same availability zone.

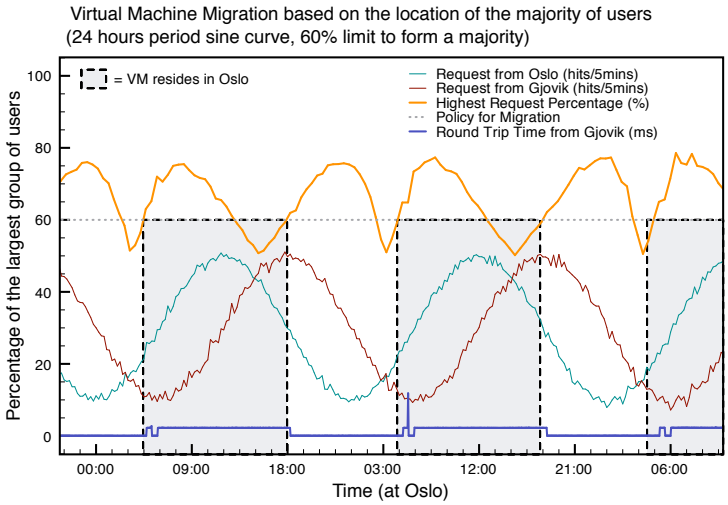

Figure 3: The virtual machine monitors the activity from the two locations (two sine curves) and decides to migrate to the location with a majority of clients.

5.2 Cloud-assisted Queue processing

In this example, we will also consider a virtual machine with a local policy. The virtual machine reads jobs from a queue and processes them in order. The jobs vary in size and could be mostly cpu-intensive, like mathematical computations or graphical rendering. The virtual machine resides at the company which issues the jobs. If the queue should become too long or a number of large jobs should reach a threshold, the virtual machine will choose to migrate into the Amazon EC2 cloud in order to become a more powerful virtual machine. The queue processing will speed up and when the number of jobs is small again, the virtual machine moves back to a physical machine at the company.

In our experiment[4], we assume a queue to be made up of different jobs of a certain type, which takes about half the time it takes to run on a local setup when processed in the cloud. It would be possible to find a length of queue for which it is more time efficient to migrate and run in the cloud. If the average time taken to migrate from the local installation to the cloud M_t, then there exists a time L_t which is the time taken for the queue to be processed locally for which $L_t = C_t + M_t$ where C_t is the time to process the queue in the cloud. For migration to be an incentive,

$$L_t >> C_t + M_t.$$

Eg. We have a queue contain units of jobs, which take 10 minutes each to complete locally, but 5 minutes complete in the cloud. Our average migration time from the local installation to the Amazon EC2 cloud was about 15 minutes,

it means that there would be equilibrium when the length of queue (n) is

$$n(10) = n(5) + 15$$

This means that for an incentive for migration, $n > 3$. If $n = 10$ for example, it would take $10(10) = 100$ minutes to process the queue locally, however it would take $10(5) + 15 = 65$ minutes to process the same queue by doing a migration into the cloud first. This behavior was implemented on a virtual machine, using MLN to migrate itself into the Amazon cloud every time the queue was longer than 3. In order to control the behavior further, we introduced a maximum 6 hour allowance to be in the cloud in one day. Interestingly, migration back from Amazon is near instantaneous, since no changes are stored anyway in the cloud, we can simply boot the filesystem we sent into the cloud when we migrated last.

6 Discussion

MLN provides a familiar interface to multiple scenarios, spanning two important cloud technologies in addition to provide local virtual machine management. We believe that local, user-oriented management of virtual machines is more beneficial than todays framework-oriented approach. It is unlikely that an organization which utilizes clouds will only be a tenant at one provider, but will instead have different providers in order to gain their individual pricing advantage. A familiar tool for management will enable seamless adoption of new cloud providers without learning their particular interface.

Another important factor, is that of monitoring and detecting phase changes on services. Data centers are often criticized for having under-utilized servers which waste power. This is true, but one fails to see that for one very important moment, the servers can become fully utilized. Detecting such a change of phase from idle to active, is a prerequisite for dynamically scaling services. We have shown that such algorithms can be implemented inside of the virtual machines themselves instead of having a resource-balancing infrastructure. With MLN, the task of implementing an algorithm, be it schedule bsed or dynamic, is that of scripting rather than heavy software developement.

The most obvious case where this is useful is when moving from one provider to another. It is unlikely that they will offer the same algorithms or interface for resource management. Bundling it inside of the virtual machines and moving them means keeping the operations logic across the providers. However, there are problems with our approach as well. If the virtual machine was to be "ignorant" of other virtual machines, how should they organize themselves? If, on the other hand, all virtual machines would know about all other virtual machines, we would in effect duplicate what would be almost the same analysis on all the nodes, which could waste resources.

The implementation of a scaling web-services show that MLN can function as an enabler for advanced behavior. The scaling algorithm used was basic but was able to adapt and increase in the rate of incoming conections based on prior knowledge. Using a-priori knowledge of a individual nodes performance for

determining the quota per node might seem static and contrast to approaches that rely on machine learning. However, in a system administration context, the service administrator may already have priori knowledge from when the service was tested during its developement and deployment before it went into production. It is therefore reasonable to assume that performance bondaries are known in many contexts. It is outside of the scope of this work to fine-tune the scaling algorithm used for optimal performance. Instead, the point is that different algorithms can be implemented and exchanged with transparency to the instance management layer.

The field of mobile agents comes to mind in the later cases. The virtual machines behavior can be characterized as that of simple, reactive agents. The idea is intriguing, as this opens the door to use many of the established concepts of group-consensus, trust, local belief and market-place dynamics. The virtual machine as not simply an operating system instance, but rather a mobile agent with the capabilities to move about and fulfill its (or their) mission is interesting. At the same time, this agent's mission is to do real work, such as running an optimal service for its organization by being closest to its users. This brings together the field of artificial intelligence and system administration in a new way, which should be explored further. It is our concern, that over-focusing on building algorithms for frameworks and leaving the virtual machines passive may overshadow the potential of this agent-based behavior.

7 Conclusion

This paper addresses the management of virtual machines from a local system administrators perspective where IaaS-type frameworks such as Amazon EC2 and Eucalyptus are utilized for increased performance and flexibility. We show that advanced management concepts can be achieved using MLN and familiar tools in the system administration community without the need to use special web-based management consoles. Our scenarios showcase cloud-oriented management which combines both local virtual machines and cloud instances. An on-demand render farm and scaling website represent what companies would be interested in realizing today. Lastly, we consider the effect of putting the decision-making capabilities of a dynamic service inside of the virtual machine, enabling it to behave in a manner more likely to mobile agents.

Aknowledgements

The author would like to thank Lu Xing and Nii Apleh Lartey for their contributions to this work.

References

[1] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov: The Eucalyptus Open-source

Cloud-computing System. Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid, Shanghai, China. pp.124-131, ISBN: 978-0-7695-3622-4, 2009, ACM Press

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield: Xen and the art of virtualization: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003, 1-58113-757-5, 164–177, ACM Press

[3] Lu Xing: A Self-management Approach to Service Optimization and System Integrity through Multi-agent Systems. Master Thesis, University of Oslo, 2008

[4] N. Apleh Lartey Virtual Machine Initiated Operations Logic for Resource Management Master Thesis, University of Oslo, 2009

[5] K. Begnum and K. Koymans and A. Krap and J. Sechrest: Using virtual machines in system and network administration education. Proceedings of the System Administration and Network Engineering Conference (SANE),2004

[6] K. Begnum, M. Disney, AE. Frisch and I. Mevaag: Decision support for virtual machine re-provisioning in production environments. Proceedings of the 21st conference on Large Installation System Administration Conference, ISBN:978-1-59327-152-7, p. 1-10, USENIX Association, 2007

[7] M. Burgess, A site configuration engine, COMPUTING SYSTEMS Volume: 8 Issue: 3 Pages: 309-337 Published: SUM 1995

[8] Burgess M, Ralston R Distributed resource administration using Cfengine SOFTWARE-PRACTICE & EXPERIENCE Volume: 27 Issue: 9 Pages: 1083-1101 Published: SEP 1997

[9] VM3: Measuring, modeling and managing VM shared resources Author(s): Iyer R, Illikkal R, Tickoo O, et al. Source: COMPUTER NETWORKS Volume: 53 Issue: 17 Special Issue: Sp. Iss. SI Pages: 2873-2887 Published: DEC 3 2009

[10] Sandpiper: Black-box and gray-box resource management for virtual machines Author(s): Wood T, Shenoy P, Venkataramani A, et al. Source: COMPUTER NETWORKS Volume: 53 Issue: 17 Special Issue: Sp. Iss. SI Pages: 2923-2938 Published: DEC 3 2009

[11] Harnessing Cloud Technologies for a Virtualized Distributed Computing Infrastructure Author(s): di Costanzo A, de Assuncao MD, Buyya R Source: IEEE INTERNET COMPUTING Volume: 13 Issue: 5 Pages: 24-33 Published: SEP-OCT 2009

[12] K. Begnum: Manage Large Networks of virtual machines. Proceedings of the 20th Large installation system administration conference, p. 16 - 28. 2006, USENIX Association

[13] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes and R. J. Figueiredo: VM-Plants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, 2004 p. 7. isbn: 0-7695-2153-3, IEEE Computer Society

[14] K. Begnum: Towards Autonomic Management in System Administration. PhD Thesis, University of Oslo, 2008 issn: 1501-7710, Unipup

[15] M. Steinder, I. Whalley, J. Hanson and J. Kephart: Coordinated Management of Power Usage and Runtime Performance NOMSIEEE (2008), p. 387-394

[16] A. Couch and M. Chiarini: Dynamics of resource closure operators Lecture Notes In Computer Science; Vol. 5637 Proc. Autonomous Infrastructure Management and Security 2009 P. 28 - 41, ISBN:978-3-642-02626-3