

Improved Virtual Machine Cloud Provisioning

Minh Huynh, Chao Jin, Blair Bethwaite, David Abramson
Faculty of Information Technology
Monash University, VIC, Australia
{minh.huynh, chao.jin, blair.bethwaite, david.abramson}
@monash.edu

Philip Papadopoulos, Luca Clementi
California Institute for Telecommunications and
Information Technology
CA, U.S.A.
phil@sdsc.edu, lclementi@ucsd.edu

Abstract—The process of provisioning Virtual Machines (VM) into the cloud requires users to conduct a series of (often manual) operations to configure and deploy their VM instances. Commercial clouds often discard all changes made to a particular running VM instance at shutdown. This means that every time system software or application requirements change, the entire re-configuration process must be repeated. A variety of system configuration techniques, like Cfengine, Puppet and Chef, can be employed to automate this process and make it less error prone. However, they cannot efficiently expand local existing infrastructures into the Cloud. This paper describes one method to automate this process by taking advantage of the Rocks Cluster Toolkit (a cluster management and deployment suite) to enable users to author and test compatible VM images in a local environment and then upload fully-configured VMs to a commercial cloud. It explores techniques to improve the performance of the entire process: the speed of transferring VM images from a local environment to a working Amazon EC2 instance. By utilizing UDT (a UDP based application level data transport protocol) and some features of Amazon’s Elastic Block Store, our approach reduces a VMs provisioning time by 35% with a very modest increase in cost. Moreover, it allows users to make an informed decision about which provisioning technique should be used based upon a performance/cost criterion.

Keywords—Cloud Computing; Cloud Provisioning; Virtual machine; Amazon EC2; Rocks

I. INTRODUCTION

Cloud computing provides computing resources with on-demand access and is believed to be an important step towards the long-held dream of utility computing [1]. In particular, computing resources and/or services are delivered to users via the Internet. Of the common Cloud service models, IaaS (Infrastructure as a Service) is the most generic model that provides computational, storage and network resources, usually via Virtual Machines (VMs) [1] hosted on the providers hardware. It allows users to completely control VM instances created in the cloud with root-level access.

With the IaaS model, the typical process of executing application requests consists of two steps. Step 1, called VM Provisioning [2], requires creating VM instances to host each application request according to the specific characteristic of

each request. Step 2 requires mapping and distributing Virtual Machines to physical resources. Presently, the second step is totally controlled by Cloud vendors, like Amazon EC2 (Elastic Compute Cloud) [3], and therefore the details are totally hidden from users. In contrast, the current process of provisioning VMs requires users to conduct a series of manual operations, from creation and instantiation of VM images to configuration and ongoing management.

To provision VMs, users may start from using the VM templates residing in the cloud. For instance, Amazon supports a VM image publication system that provides users more than 7,000 VM images [4] with a variety of platform (operating system and application software) configurations. However, without an efficient classification system, looking for a VM image exactly matching a specific requirement is not easy [4]. Before safely reusing a published VM image, it should be examined to guarantee that it is free of worms, vulnerable software and undesired or insecure software. Frequently, users still need to install additional software on the selected VM images.

A straightforward approach for provisioning VMs is to build images from scratch according to users’ specific requirements. This provides acceptable safety, but requires copying the VM images from users’ local host into the cloud. Currently, a typical Cloud vendor, such as Amazon EC2, splits the VM image file into small chunks before transferring them into the cloud across the Internet. This process frequently consumes a significant amount of time due to the performance bottleneck caused by inefficiently transferring a number of small files.

In addition, configuring virtual machines and installing the required software is not trivial. Commercial clouds, like Amazon EC2, often discard all changes made to a particular running VM instance at shutdown. Moreover, when users need to update their application requirements, they need decide what action should be taken: creating brand new VM images or reconfiguring an existing one. Users must have knowledge of the performance and cost of each method before they can make an informed decision.

In this paper we present a method which improves the process of provisioning VMs into Amazon EC2, from locally building VM images to configuring software within the cloud, by taking advantage of the Rocks toolkit [4]. In particular, this paper presents the following contributions:

- The VM image is transferred as a whole file into an Amazon Elastic Block Store (EBS) volume by using UDT [5] rather than transporting a number of small chunks into Amazon Simple Storage Service (S3);

- The process of reconfiguring and redeploying VM images is automated by using Rocks; and
- The time of provisioning VM images has been reduced by 35% with a very modest increase in cost.

The remainder of this paper is organized as follows. Section II gives an overview of related work and section III reviews Rocks. Section IV describes our solution. Section V presents its design and implementation. Section VI evaluates its performance. Section VII presents the conclusion.

II. BACKGROUND AND RELATED WORK

Amazon's EC2 is one of the most widely used and mimicked IaaS models of cloud computing. It currently provides 17 types of VM instances, equipped with different memory space, CPU cores, disk capacity and in some cases, special hardware. An Amazon Machine Image (AMI) is used by EC2 to create a virtual machine. It can be stored in the S3, which is an object storage system. Amazon EBS is a newer type of storage specifically designed for EC2 instances.

Substantial numbers of research projects [6][7] focus on the dynamic provisioning of computing resources within the cloud. Many of them investigate extending the capacity of a private cluster with on-demand resources drawn from the cloud. A number of research projects [7][8][9][10] have investigated provisioning techniques for particular clouds. Genaud, et al. [11] evaluated four sets of provisioning strategies. However, most of these projects depend on using the existing tools provided by the cloud vendors to provision VM images, which may cause performance bottlenecks [Section VI.A]. Cost is an important parameter with provisioning. However, the cost of different provisioning techniques was not investigated.

Configuration Management (CM) tools, like Cfengine [12], Puppet [13] and Chef [14], are used to automate system administration for VM instances running in the cloud. These tools allow the user to describe system resources, their states, the relationship between installation packages, and the order of installation using specific configuration languages. Cfengine is an IT infrastructure automation framework written in C that helps administrators to build, manage and maintain IT infrastructure throughout its lifecycle. Puppet is a Ruby-based CM tool and was created with the aim to simplify the complexity of programming configurations. It has been integrated with a number of clouds. Chef is similar to Puppet and its command line interface, Knife, is used to handle VM instances for Amazon EC2.

All these three tools require a starting minimal OS installed and then on top of that they can install the required software stack and perform the necessary action to properly configure it, although none of them can deploy a system on a bare metal hardware. So even when using one of these tools the user still needs to find a suitable starting AMI from the one available on EC2. In comparison to the above work, our approach focuses on improving the whole process of provisioning VMs into the Amazon cloud by taking advantage of Rocks. Simultaneously, we help users to understand the tradeoff of performance and cost for various provisioning technologies.

III. OVERVIEW OF ROCKS

Rocks [4] provides a framework based on Kickstart [15] to describe the configuration of Linux machines. It manages a system descriptor for installing software. A directed graph is utilized to represent the dependency relationship between installation of software packages and configurations. Each vertex in the graph represents the configuration of a specific service, such as a software package or service configuration, and the dependency between services is represented by directed edges connecting vertices.

The Rocks toolkit allows systems to be built based on selected components, called *Rolls* [16]. Each Roll contains a configuration sub-graph which defines package name, version, and patches. The system graph is defined by composing a set of Rolls selected by users. There are many Rolls available, such as the base Rocks OS Roll, the Xen Roll, and the Python Roll. When Rocks is used to set up a cluster, the system graph is traversed for each physical node to define the set of packages and configuration for each machine [4]. The traversal output is interpreted by the system installer (Anaconda Kickstart for Linux or Jumpstart for Solaris) to build each system image. A logical system image or a type of machine is called an 'appliance' by Rocks. For example, a *lite appliance* is a machine with the core Linux kernel and minimal installed software. Many appliances are specified in Rocks such as the *compute appliance* and a *development server appliance*.

In 2007, an add-on roll was introduced that enables customers to build VM images using the identical graph framework that defines physical machines [4]. With the aid of the Xen Roll, and the EC2 Roll, Rocks has been used to extend clusters into Amazon EC2 [4], and simplify the process of complex cluster management. For instance, to set up a virtual Linux-based cluster for scientific computing, the head node needs to install approximately 1,500 packages, with more than 200 configuration changes, while the compute node needs to install around 850 packages, with more than 100 after-installation configuration changes[4]. The whole process of software installation and configuration is automated by using Rocks.

IV. ARCHITECTURAL SOLUTION

Most existing techniques for provisioning VMs into Amazon EC2 rely on the standard tools provided by Amazon, which causes performance bottlenecks [Section VI.A]. Reconfiguring VM instances that are running in the cloud depends on a separate set of tools supported by third party providers, such as Cfengine, Puppet and Chef. Here we aim to provide a solution to support the whole process of VM provisioning and improve its performance.

We take advantage of the programmable and extensible configuration systems of the Rocks toolkit to achieve this goal. Compared to the existing process of using the Rocks toolkit to provision VMs, two improvements have been made. 1) The file of VM image is transferred into the cloud using UDT rather than the tools provided by EC2. 2) The process of reconfiguration and redeployment is automated by Rocks. The whole process is illustrated in Figure 1.

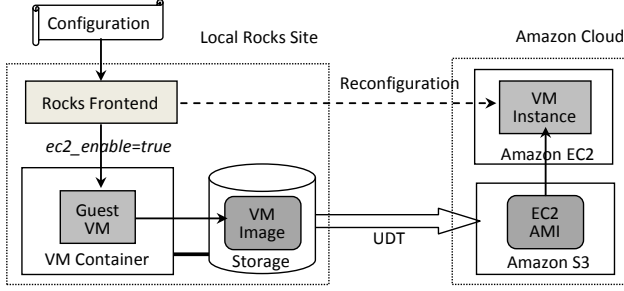


Figure 1. Our process of provisioning virtual machines into Amazon EC2.

Users need to edit a Rocks configuration graph according to the software stack required to install and send it to the Rocks Frontend. It is actually an XML file and existing XML files of configuration graph can be used as templates for adding required rolls or removing old ones. An example configuration file for creating an EC2 compatible VM image for a *lite appliance* is illustrated in Figure 2. Its *ec2_enabled* attribute is set *true* to make the VM compatible with EC2.

The graph configuration is processed by the Rocks Frontend to perform installation automatically, and to create the Guest VM within the VM container. The VM image is compatible with Amazon EC2, and is maintained by local storage before it is transferred into EC2.

Unlike the tools provide by Amazon EC2, we do not split the image file into small chunks. Instead, the image file is transferred as a whole into an EC2 instance by using UDT. The received image is initially kept in an EBS volume[Section V.A]. Subsequently, it is stored in S3 and registered with a unique AMI (Amazon Machine Image) identifier. Any number of EC2 instances could be instantiated using the registered image. Updating their configurations is handled by the Rocks Frontend.

```
<?xml version="1.0" standalone="no"?>
<graph>
  <description>Rocks Lightweight Appliance on EC2.</description>
  <!-- Graph Ordering for Kickstart Generation -->
  <order head="python-development">
    <tail>lite-base</tail>
  </order>
  <!-- Graph OO Framework -->
  <edge from="condor-server">
    <to>ec2-condor-server</to>
  </edge>
  <edge from="lite-client">
    <to>lite-base</to>
    <to>411-client</to>
    <to>node-sets</to>
    <to>pxeboot</to>
    <to>networking-client</to>
    <to>firewall</to>
  </edge>
  <edge from="lite-client" to="condor-client" arch="i386,x86_64"
cond="Condor_Client"/>
  <edge from="lite-client" cond="ec2_enable">
    <to>ec2-client</to>
    <to>ec2-common</to>
    <to>ec2-partition</to>
  </edge>
</graph>
```

Figure 2. An example of Rocks configuration file.

A. Improving the Performance of Image Bundling

The tools provided by Amazon EC2 requires creating the full bundle, splitting the file of VM image into a number of 10MB encrypted chunks and then uploading them. The chunks are transported by using a single active transfer stream over HTTP. The whole process is inefficient, because

a typical VM image creates several hundred small chunks, which causes additional storage and network overhead when copying them to the cloud [17]. These factors significantly increase the time of transferring VM images into EC2 over the Internet. Therefore, rather than copying small chunks with HTTP, we choose to transfer the whole file with an efficient protocol to ship VM images.

Several options are available for efficiently transferring large files across the Internet. GridFTP [18] has been widely used as a data transfer protocol to efficiently ship a large volume of data for grid computing. It has been deployed in Amazon EC2 [19]. However, users must install its client and server software and set up accounts [18]. These requirements do not fit our needs.

UDT [5] provides a UDP-based approach that can transfer data at a much higher speed than TCP. It is able to break the data transfer bottleneck across the Internet. In particular, UDT transfers bulk data with guaranteed reliability by using its own mechanisms of reliability control and congestion control. In addition, UDT is easy to deploy. Therefore, we selected UDT to transfer the large VM image file into the cloud by taking advantage its high-speed networks and parallel transmission.

B. Reconfiguration and Redeployment

When users need to deal with changes in the computation environment, such as installing new software or reconfiguring environment variables, it can be very ineffective to rebuild the VM image and transfer it again into the cloud. In contrast, our approach uses Rocks to update the running VMs incrementally.

In particular, when a running VM instance in the cloud needs to be altered, we update its Rocks configuration file. The Rocks toolkit examines the new configuration file and creates a new Kickstart file for updating the configuration of the VM instance, including software and environment.

Within EC2, each running instance receives the new Kickstart file. By comparing it with the current one, it can detect the additionally required installation and configurations. It then performs the installation and applies configuration, thereby keeping itself up-to-date.

V. DESIGN AND IMPLEMENTATION

This section presents the details of utilizing UDT to improve the performance of transferring VM images into Amazon EC2 via EBS volumes and automating the process of re-configuration.

A. Improving the Performance of Transferring VM Images

In order to utilize UDT to transfer the entire VM image as a whole file, an Image Sender and Receiver were created. The Image Sender is located in the local host and controlled by the Rocks Frontend, while the Image Receiver is located within the cloud. The Receiver Instance is a normal Amazon AMI with UDT installed as a client and an EBS volume attached to store the incoming image.

To enable the Image Sender to communicate with the Image Receiver, the Receiver Instance is created with modified security rules that open some predefined UDP ports

that allow UDT to transfer data. The Image Receiver listens to the predefined port number, while the Image Sender initiates the transfer. When the image transfer completes, both the Image Sender and Image Receiver instances can be terminated. The target EBS volume then contains the VM image from which a snapshot is created, stored in S3 and registered as a new AMI.

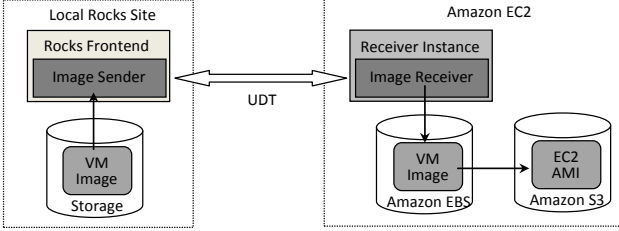


Figure 3. Transferring VM images into Amazon EC2 by using UDT.

Transferring VMs into EC2 using UDT is coordinated by the Rocks Frontend, as illustrated by Figure 3.

Initially, the following steps are conducted within EC2:

- 1) Boot up a Receiver Instance.
- 2) Attach an EBS volume to the Receiver Instance and the volume is formatted with *ext3*.
- 3) Start running the UDT Receiver in the Receiver Instance and bind it to a predefined UDP port.

After the above steps complete, the Rocks Frontend starts transferring the VM image with the following steps:

- 4) Loop mount a Virtual Machine image of Rock machine to a local directory.
- 5) Start running the UDT Sender to transfer entire directory to the EBS volume.

Simultaneously, within the cloud, the following steps are conducted to aid the transfer:

- 6) Modify *fstab* so that it can be booted by EC2 (the root partition of the image is mapped to */dev/sda1* by EC2).
- 7) Detach the EBS volume; take a snapshot from that volume and register an AMI from the snapshot.

B. Improving Reconfiguration and Redeployment

To simplify the process of reconfiguration, initially a *lite appliance* machine is uploaded. During redeployment, it is upgraded according to the specific reconfiguration requests. This process is controlled by the Rocks Frontend to transform an old appliance type into a new one.

After a *lite appliance* is transferred into the cloud, only the following steps are conducted for each reconfiguration:

- 1) Users need to specify all the XML files that define the software stack and post configurations for the reconfiguration and redeployment.
- 2) Boot up a *lite appliance* instance in EC2. When it starts, the Frontend sends it the corresponding Kickstart file after traversing the system graph configured in step 1.
- 3) The new Kickstart file is used to transform the *lite appliance* into a new type.

After receiving the new Kickstart file, the *lite appliance* instance compares it with its current one. After that, it detects the required installation and figures out the remaining tasks, which could mean installing new software or modifying the

existing configurations. Step 3 consists of two minor steps. The first one is to detect which packages need to be installed by comparing the packages that are currently installed and the new packages defined in the new Kickstart. The second step performs post installation. It is needed to setup environment variables and perform post installation required by new packages.

VI. PERFORMANCE EVALUATION

This section examines the performance and cost of our provisioning method. We built VM images, uploaded them into Amazon EC2, and redeployed them by using Rocks from two sites: UCSD (University of California, San Diego) and Monash University in Australia. Each site consists of 2 servers running a Frontend and a VM Container respectively.

UCSD: Each server is equipped with a Quad core CPU of clock speed 1.6 Ghz with 4GB RAM.

Monash University: 2 Dell PowerEdge 1950 servers were used. Each server is equipped with 2 Quad-Core Intel Xeon E5310 1.6GHz processors and 8 GB RAM.

The Amazon US East (N. Virginia) region was used in our experiments with 3 EC2 instance types: Small, High-CPU Medium and High-CPU Extra Large.

The time and cost of building VMs and uploading them into the cloud were measured respectively for 3 methods. The *standard* approach is the existing method in Rocks, which utilizes the tools provided by Amazon EC2. With *option 1*, Rocks used UDT to transfer VM images, as described in Section V.A. With *option 2*, a Rocks *lite appliance* was built and then uploaded into the cloud and subsequently its instance was upgraded to a required configuration by installing additional software, as described in Section V.B. Each experiment was repeated 5 times in both sites and the averaged values are presented.

The cost is calculated for 24 hours usage of the Amazon cloud. It consists of 3 parts: leasing on-demand EC2 instances, transferring data into EC2 and storing VM images. The pricing of Amazon S3 [20] and EC2 [21] at 30, Jan., 2013 is used to calculate the cost.

A. Performance and Cost of Provisioning VMs

We built a VM image of the Rocks *compute appliance* for Amazon small instance, which is installed with software to perform common computations, including parallel processing using MPI, and Java. The size of the *compute appliance* image is about 3.9 GB. The performance and cost of the 3 provisioning methods were measured while adjusting the number of uploaded instances. The results are illustrated in Figure 4. The size of the *lite appliance* image used by *option 2* is 2.2 GB.

The time of building VMs and uploading them into EC2 was examined. Compared with the standard approach, *option 1* requires 35% less time while the number of VM instances increases from 1 to 16, because using UDT to transfer the image file as a whole saved a significant amount of time. The performance of *option 2* is always worse than the other 2 methods. Although the time of transferring images with *option 2* is the smallest, installing required software within EC2 is slow because the additional software is fetched across

the Internet from the Rocks Frontend. In addition, as the number of instances increases, this bottlenecks are worsens.

The size of the single VM image stored in S3 is same for each method no matter how many instances are desired. Therefore, the storage cost of each provisioning method has no difference. However, option 1 and 2 are more expensive than the *standard* approach, because they require starting a Receiver Instance in EC2 to receive the VM images. Additionally, the Receiver Instance needs to reserve enough storage space in EBS to keep received VM images. The cost of transferring data into S3 and EBS can almost be ignored. When more instances are required, the cost of each provisioning method increases accordingly.

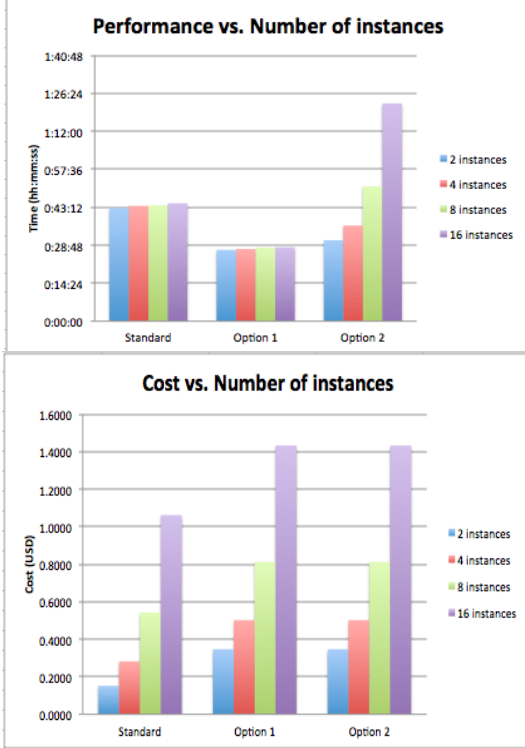


Figure 4. Comparing the performance and cost of provisioning VMs into EC2 with different approaches.

B. Effect of Image Size and Hardware Type

In this section, we investigate the impact of VM image size and EC2 hardware configuration on the performance and cost of each provisioning technique.

To examine the impact of image size, three types of VM images were used for Amazon Small instance: the *lite appliance* VM image (2.2 GB), the *compute appliance* image (3.9 GB) and the *development server appliance* image (4.2 GB). To evaluate the impact of different hardware configurations, the *compute appliance* was built on three different EC2 instance types: Small, High-CPU Medium and High-CPU Extra Large.

The time for building images and uploading them into EC2 was measured for each method with each configuration. The results are presented in TABLE I. The corresponding cost required for each configuration is shown in TABLE II.

Provisioning a *lite appliance* is much faster than provisioning a *compute* or *development server appliance*. This applies to all provisioning techniques, because the smaller size of the image requires less time to upload. The key factor that affects the performance of *option 2* is the number and size of installed packages. Provisioning a *development server appliance* takes the most time because it installs more packages than the other two appliances.

TABLE I. PERFORMANCE WITH VARIOUS IMAGES AND HARDWARE

Performance (m:s)	Standard	Option 1	Option 2
Lite Appliance	23:07	14:23	14:26
Compute Appliance	42:03	26:24	26:48
Development Server	50:19	31:34	34:29
Small Instance	42:03	26:24	26:58
High-CPU Medium	41:50	25:37	25:57
High-CPU Extra Large	40:29	25:03	25:12

TABLE II. COST WITH VARIOUS IMAGES AND HARDWARE

Cost (US\$)	Standard	Option 1	Option 2
Lite Appliance	0.0817	0.2530	0.2530
Compute Appliance	0.0870	0.2734	0.2734
Development Server	0.0879	0.2761	0.2761
Small Instance	0.0817	0.2530	0.2530
High-CPU Medium	0.1870	0.4695	0.4695
High-CPU Extra Large	0.6829	1.4621	1.4621

Using a more powerful EC2 instance type does not significantly change the performance of any provisioning techniques, because provisioning is I/O bounded and the magnitude of data involved is constant for each technique. However, using a high I/O performance instance (such as High-CPU Extra Large) is much more expensive than a Small Instance according to TABLE II.

C. Reconfiguring VMs

This section evaluates the performance of each provisioning technique when updating the software installed on EC2 instances. The following 2 cases were investigated:

- 1) Update-only: update configurations without installing any software. A *lite appliance* image was used in this case.
- 2) Install-new: install new software, specifically Java and Vim, then perform post-configurations in the appliances.

To reconfigure a VM, the *standard* and *option 1* approaches require building the entire machine image and uploading it into the cloud. In contrast, with *option 2*, those running instances just request a new Kickstart file and then perform additional installation or conduct post-configurations as required.

The results are illustrated in Figure 5. It shows that *option 2* requires the smallest time to reconfigure the running instances in EC2. With *option 2*, only small files of a few KBs were transferred in our test. Moreover, the *standard* and *option 1* approach require an extra cost to instantiate the EC2 instances with the new VM image and a cost for storing it.

As a conclusion, *option 2* is a suitable solution to reconfigure a VM instance when the total size of additional software is comparatively small. When there are a large number of instances to be reconfigured and the size of the additional software is big, the performance of option 2 is

dramatically decreased as explained in section VI.A. In this case *option 1* is a more suitable solution.

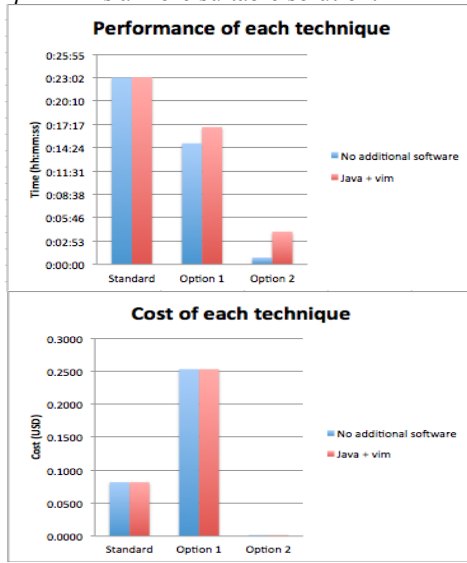


Figure 5. Comparing the performance and cost of reconfiguring EC2 with different approaches.

VII. CONCLUSION

VM provisioning is an important step in cloud computing. Most existing solutions have performance bottlenecks with cost constraints. We describe a method to take advantage of Rocks to manage the whole process, including reconfiguration and redeployment.

The performance of transferring VM images into the cloud has been improved significantly by shipping the whole image file using UDT. In comparison to using the tool provided by Amazon, our method saves 35% of the image bundling, uploading and registration time.

When updating the VM instances in the cloud, the time and cost saved by our reconfiguration method depends on the size of installed software. It can be used to prevent shipping multiple VM images for different types of VM instances. For example, creating a *compute appliance* and *server appliance* by reconfiguring a *lite appliance* in EC2 is able to save a significant amount of overhead in comparison to bundling them from scratch. The overall cost and performance of our method are evaluated and compared with the *standard* approach used in Rocks. This provides users with guidance when making a decision about which provisioning technique suits them based on performance/cost criteria.

ACKNOWLEDGMENT

This project is supported by the Australian Research Council and the MURPA (Monash Undergraduate Research Projects Abroad) program, which is funded by Monash University. We acknowledge the team members at CalIT2 and the team members of MeSSAGE Lab at Monash University for their contribution to this project. Minh Huynh

performed the work in partial fulfillment of his Bachelor of Software Engineering research project.

REFERENCES

- [1] Armbrust, M., et al. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, Feb 2009.
- [2] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in Proc. of the 10th IEEE/ACM Intl. Conference on Grid Computing (GRID'09), 2009.
- [3] Amazon, "Amazon EC2 User Guide," [Accessed: 10 January, 2012].
- [4] P.M. Papadopoulos, "Extending clusters to Amazon EC2 using the Rocks toolkit," Intl. Journal High Performance Computing Applications, vol. 25, no. 3, 2011, pp. 317-327.
- [5] Y. Gu and R.L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," Computer Networks, vol. 51, no. 7, 2007, pp. 1777-1799.
- [6] C. Vazquez, et al., "Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers," Proc. Grid and Pervasive Computing Conference, 2009, pp. 113-120.
- [7] M. Ming, et al., "Cloud auto-scaling with deadline and budget constraints," Proc. 11th IEEE/ACM Intl. Conference on Grid Computing (GRID), 2010.
- [8] S. Ostermann, et al., "Resource Management for Hybrid Grid and Cloud Computing, Cloud Computing," Computer Communications and Networks, 2010, pp. 179-194.
- [9] P. Marshall, et al., "Elastic Site: Using Clouds to Elastically Extend Site Resources," Proc. of 10th IEEE/ACM Intl. Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 43-52.
- [10] D. Candeia, et al., "Investigating Business-Driven Cloudburst Schedulers for E-Science Bag-of-Tasks Applications," Proc. 2nd IEEE Intl. Conference of CloudCom, 2010.
- [11] S. Genaud and J. Gossa, "Cost-Wait Trade-Offs in Client-Side Resource Provisioning with Elastic Clouds," Proc. IEEE Intl. Conference on Cloud Computing (CLOUD), 2011.
- [12] M. Burgess, "Cfengine a site configuration engine," USENIX Computing Systems, vol. 8, 1995.
- [13] T.D. Nielsen, et al., "Private Cloud Configuration with MetaConfig," Proc. 14th IEEE Intl. Conference on Cloud Computing, Italy, 2011.
- [14] "Chef," <http://www.opscode.com/chef/> [Accessed: 10 January, 2012]
- [15] Redhat, "Redhat Installation Guide," May, 2012.
- [16] G. Bruno, et al., "Rolls: modifying a standard system installer to support user-customizable cluster frontend appliances," Proc. IEEE Intl. Conference on Cluster Computing, , 2004, pp. 421-430.
- [17] http://moo.nac.uci.edu/~hjm/HOWTO_move_data.html
- [18] H. Subramoni, et al., "High Performance Data Transfer in Grid Environment Using GridFTP over InfiniBand," Proc. of 10th IEEE/ACM Intl. Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 557-564.
- [19] GlobalOnline, <https://www.globusononline.org/>
- [20] Amazon Inc, "Amazon S3 Pricing," <http://aws.amazon.com/s3/pricing> [Accessed: 10 January, 2012]
- [21] Amazon Inc, "Amazon EC2 Pricing," <http://aws.amazon.com/ec2/pricing> [Accessed: 10 January, 2012]