PROCEEDINGS OF SCIENCE

Repoman: A Simple RESTful X.509 Virtual Machine Image Repository

M. Vliet[a], A. Agarwal[a], M. Anderson[a], P. Armstrong[a], A. Charbonneau[b], K. Fransham[a b], I. Gable[a], D. Harris[a], R. Impey[b], C. Leavett-Brown[a], M. Paterson[a], W. Podaima[b], R.J. Sobie[a]

[a] *University of Victoria*
Victoria, Canada V8W 3P6
[b] *National Research Council Canada*
100 Sussex Drive, Ottawa, Canada

With broader use of IaaS science clouds the management of multiple Virtual Machine (VM) images is becoming increasingly daunting for the user. In a typical workflow, users work on a prototype VM, clone it and upload it in preparation for building a virtual cluster of identical instances. We describe and benchmark a novel VM image repository (Repoman), which can be used to clone, update, manage, store and distribute VM images to multiple clouds. Users authenticate using X.509 grid proxy certificates to authenticate against Repoman's simple REST API. The lightweight Repoman CLI client tool has minimal python dependencies and can be installed in seconds using standard Python tools. We show that Repoman removes the burden of image management from users while simplifying the deployment of user specific virtual machines.

The International Symposium on Grids and Clouds and the Open Grid Forum - ISGC2011,
March 25-30, 2011
Academia Sinica, Taipei, Taiwan

PoS(ISGC 2011 & OGF 31)052

1. Introduction

Infrastructure as a Service (IaaS) cloud computing is quickly becoming an effective method of providing substantial computational power to researchers, while removing the need to provide and maintain dedicated execution environments for a particular experiment. The use of IaaS allows for the entire application environment of an experiment to be contained within Virtual Machine (VM) images, which can be easily used to deliver on-demand computational capacity at any number of the IaaS offerings available to the user. Many existing IaaS software packages provide mechanism for managing VM images. However, these tools often depend on a particular IaaS implementation and rely on authentication tied to a particular cloud. In this paper we describe a VM image repository we have developed called Repoman, which is independent of particular IaaS implementations and relies upon X.509 certificate authentication familiar to the grid community.

In previous work we developed a batch processing system using a component called Cloud Scheduler[1]. Cloud Scheduler works by booting user provided VM images, retrieved from an HTTP web server, on IaaS clouds to provide computational resources to users. The creation of the VM images is a largely manual process in which an expert creates a basic image, installs the software and configures the image according the needs of a specific researcher. Once the image has been created, it must be placed with the appropriate permissions on the web server by an administrator. This method of requiring an expert to build the image for the researcher, and have an administrator upload the image for use becomes very impractical when dealing with more then a handful of researchers.

In an ideal system, the creation and management of VM images would be offloaded to the researchers. However the creation of VM images can be a daunting and error prone task for users unfamiliar with virtualization technologies. Requiring a researcher to manage the process of building and configuring a VM image from scratch would impose an undesirable barrier for new users. To make the system more accessible to researchers, there is a need for a system which can provide a set of preconfigured VM images. There is also the need to allow more advanced researchers to interactively customize an available VM image by modifying the installed software or configuration and saving their customizations back to the image repository to be used at a later date. These two requirements led to the development of the Repoman image repository. Certain technical design choices made during the development of Repoman were largely driven by the need to provide easy integration into the existing services used within the grid science community. X509 certificates[2] were chosen for authentication because it is the preferred authentication mechanism used in the grid community and is used to authenticate users with the Nimbus[3] IaaS software. A RESTful[4] interface to the repository was chosen as HTTP is the preferred method of image transfer within our IaaS clouds.

Repoman is implemented as a RESTful web service which is authenticated with X509 certificates. Interaction with Repoman requires that a user account is first created on the Repoman server by an administrator. Once an account has been created, the user can start by either uploading an existing VM image to the Repoman repository, or use a preexisting image within the Repoman repository which has been shared with them. In either case, the user can request an interactive session with the chosen VM image and customize it. The user has the ability to save the changes made to the image back to the repository, similar to the functionality provided by Ama-

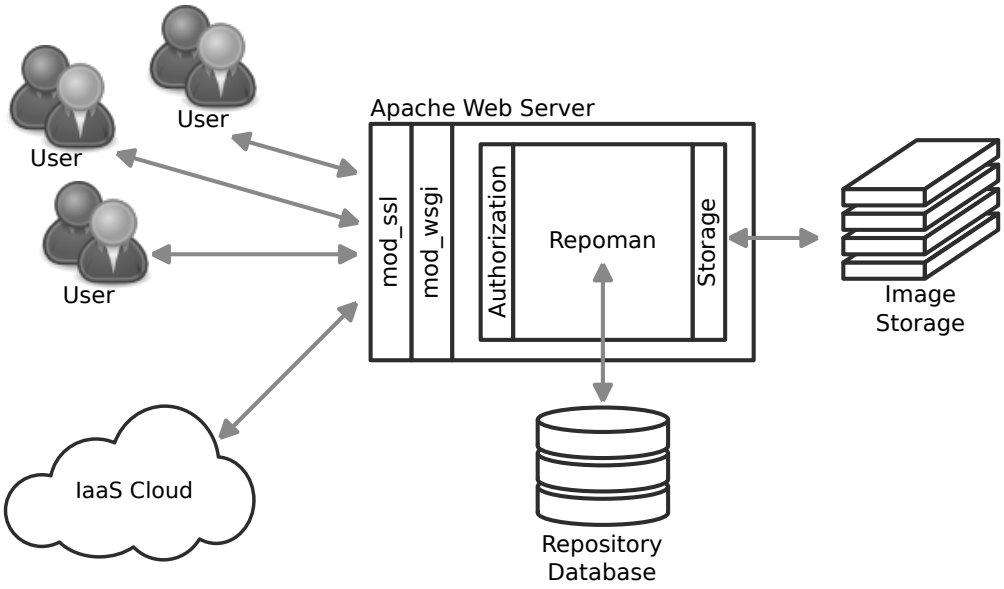

Figure 1: An overview of the Repoman system. A client, a user or IaaS software, talks to the Repoman Repository served by an Apache web server. The client is first authenticated with their X509 certificate by mod_ssl. The request is then passed to mod_wsgi, which in turn passes the request to Repoman. The request is examined to determine if the requesting client is authorized. Repoman interacts with the database or image storage and generates a response for the client.

zons `ec2-bundle-vol`[5] command. VM images stored within the Repoman repository are accessible via HTTPS or HTTP and can be retrieved by tools such as `wget` and `curl`. In addition to storing the VM image itself, the repository is also able to track metadata associated with each image. To facilitate the sharing of VM images, users have the ability to share a read-only copy of their image directly with another user, or to a group of users. A high level overview of the Repoman architecture and how it fits in with existing cloud infrastructure is shown in fig. 1. A detailed explanation of the server and client architectures can be found in section 2 and section 3. In section 4 the performance of Repoman is evaluated to determine possible limiting factors.

2. System Architecture and Implementation

The server component of Repoman is implemented as a RESTful web service written in Python and using the Pylons[6] web framework. The Pylons framework allows for the creation of web applications which are compatible with the WSGI[7] specification and was chosen for its flexibility and large community of developers. Although the Pylons framework includes a built-in web server capable of serving basic applications, it was not capable of retrieving and authenticating X509 client certificates. To run Repoman the Apache HTTP Server[8] was chosen as it provided the mod_ssl[9] extension module which is capable of authenticating X509 certificates retrieved from the client. In addition to mod_ssl, the mod_wsgi[10] extension module was also required to allow for a Pylons application to be run within Apache. In addition to the Pylons framework,

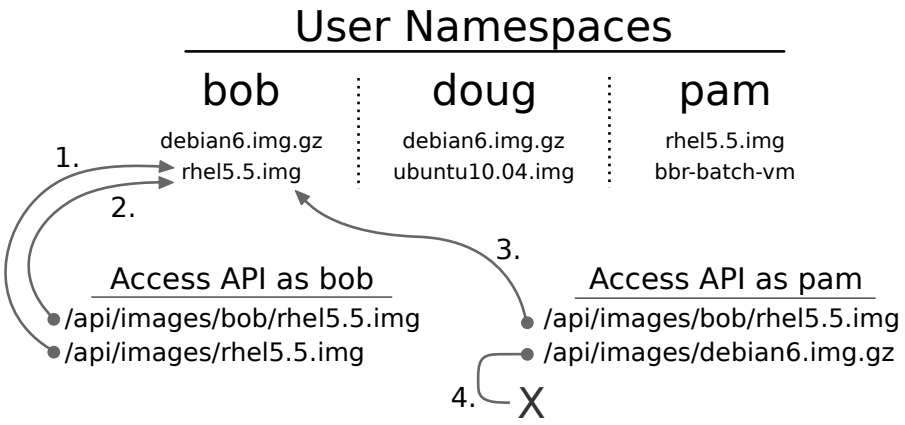

Figure 2: An explaination of namespaces in Repoman. Each of the three users, bob, doug, and pam have two images owned by each of them. 1. User bob accesses /api/images/rhel5.5.img which implicitly maps to his namespace. 2. User bob accesses /api/images/bob/rhel5.5.img which explicitly maps to his namespace. 3. User pam accesses /api/images/bob/rhel5.5.img which explicitly maps to the namespace of user bob. 4. User pam accesses /api/images/debian6.img.gz which does not map to any image in her own namespace, so returns "404 Not Found" error

SQLAlchemy[11] is used to interface with an SQL database to provide persistence for image metadata as well as user and group account information.

The API implemented by Repoman is based on REST design principles. It is implemented as a set of URLs which can be queried using the standard HTTP methods (GET, POST, PUT, and DELETE) to perform various actions within Repoman. There are three main URL stubs from which the majority of URLs are constructed. These stubs correspond to the three main objects that Repoman implements, those being images, groups and users. Each of the base URLs take the form of `/api/{object}`, where object is replaced by one of `users`, `groups` or `images`. In keeping with REST principles, sending a GET request to one of the base URLs will result in a response containing a list of all objects of that type, while sending a POST request will create a new object of the specified type. From the basic URLs, an API has been built allowing Repoman to service a comprehensive range of functions.

2.1 Image Namespaces

In order to allow multiple users to store images with the same name within Repoman an implicit namespace, represented by their username, has been implemented. Within each user namespace all image names must be unique. A simple rule is used to determine in which namespace an object belongs. If no namespace is specified within the URL, the namespace of the current user is implied. Otherwise, the specified namespace will be used to find the image in question. An illustration of this namespace resolution is shown in fig. 2. One of the benefits which results from the use of namespaces to store images, is that images can be accessed via a human readable string rather then a complex image ID used in other systems.

2.2 Image Storage

Uploading image files to a Repoman repository is done by first creating a slot for that image to be stored. This is done by sending a POST request containing image metadata to /api/images which will trigger Repoman to create an entry in the database. Once the slot has been created the image file is uploaded to the repository by performing a PUT operation to the corresponding slot which is a URL of the form `/api/images/raw/{[user/]image-name}`, where `image-name` is the name of the image that was passed as a metadata value during the slot creation. The body of the PUT operation contains the actual image file.

When the upload has completed, the image will be available for download at the same url to which it was uploaded. Behind the scenes, the image file is stored in a directory on the local file system of the server, and an entry in the database image object points to the location on disk. The image storage location does not have to be a local disk however, it can be any mountable storage system.

3. Client Implementation

Interaction with the server can be accomplished by using the REST API directly, or by using a command-line client written for Repoman. The client which accompanies Repoman is a separately installable Python script which works on Linux and Mac OSX platforms. The client tool is currently hosted within the Python Package Index[12], and can be installed with a single command (`pip install repoman-client`). The client provides basic features to the user allowing image creation, deletion, modification and sharing as well as administration tools for user management. In addition to providing a simple interface to the API for the user, the client also features a tool that allows for the snapshotting of a running machine, either virtual or physical, and saving it back to a Repoman server. This is similar to the functionality that Amazon's `ec2-bundle-vol` command provides, yet it provides a simpler interface where much of the complexity is hidden from the user. To the end user, only a single command needs to be issued to create and upload any changes made to an image while running interactively. In addition to providing a simpler interface, the image size restriction of 10GB imposed by the `ec2-bundle-vol` command does not apply to Repoman. The maximum size of the saved image is determined only by the amount of free space available when performing the snapshot.

The snapshot feature acts by first creating a sparse image file of the same size as the current root partition on the running machine. To avoid issues with lack of free space when creating the sparse image file, it is common to specify that it be placed on a sufficiently large secondary disk attached to the running machine. Once the sparse image file is created, it is formatted with a file system in preparation for the snapshot. The new file system, containing the sparse image file, is then mounted locally and rsync is used to create a complete copy of the running root file system, excluding certain directories such as tmp and proc. Any excluded directories are later recreated in the sparse file as empty directories to ensure the image is bootable at a later date. The list of excluded directories and files can be customized by the user to prevent any sensitive information from being stored on the image. Once the rsync is complete, the image file is uploaded to the Repoman repository where it can be later downloaded and booted. This snapshot feature is useful

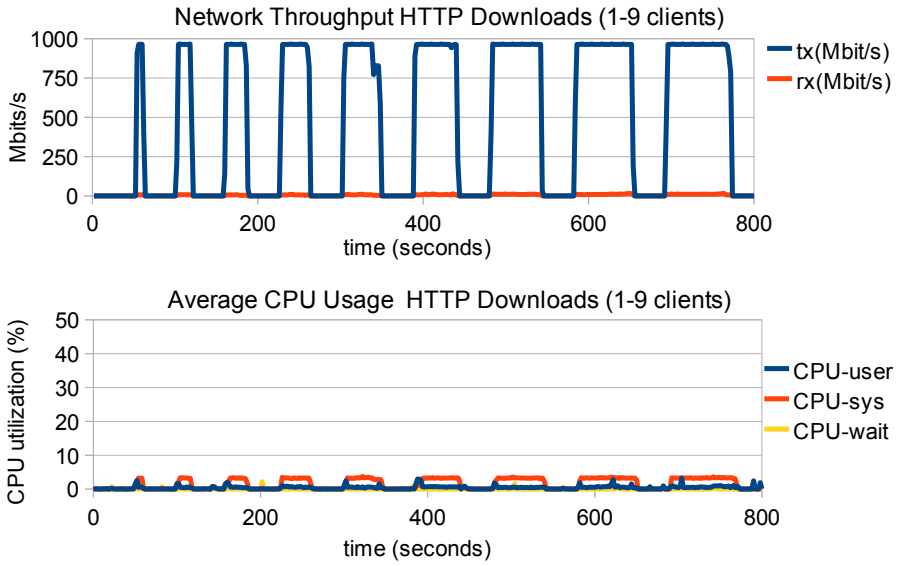

Figure 3: Server statistics while downloading via HTTP.

for prototyping virtual machines in an interactive manner and saving the modified VM image for later use. With this system, the user is in complete control of what directories and files are saved.

4. Performance

In this section we will examine the performance of the server in different test situations in order to determine any performance bottlenecks. The aim of the tests outlined in this section is to demonstrate the ability for Repoman to saturate the available network capacity. Although a full scale benchmark of the server is not provided in this paper, Repoman has successfully been used to simultaneously deploy up to 150, 15GB VM images in a production environment.

For all tests, Repoman was run on a dual CPU Xeon E5520 server, allowing 16 hardware threads on 8 cores, with a RAID 5 storage system and connected to a local switch via a gigbit ethernet link. Each of the clients in the tests were located in the same cluster as the server, and each was connected to the same local switch via a gigabit ethernet link. As a note about the CPU utilization graphs, the values are representative of the average of the sum of the utilization of each of the 16 threads. In the test system , full utilization of a single core is equivalent to 6.25% of the total CPU utilization.

For each of the tests performed, an increasing number of simultaneous clients, ranging from 1 to 9, was used to measure the network throughput and the CPU utilization on the server. Each test started with a single client performing either an upload or download operation. After all clients at a particular stage have completed performing the upload or download, there is a short pause before the number of clients is increased by one and each client runs the previous upload or download operation once more. This continues until the number of simultaneous clients reaches 9, at which point the test has completed.

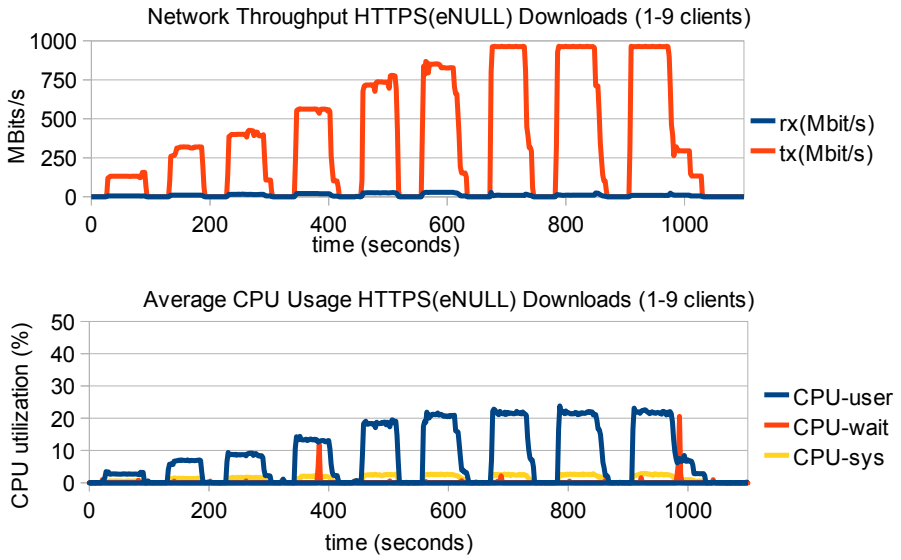

Figure 4: Server statistics while downloading via HTTPS with the eNULL cipher suite.

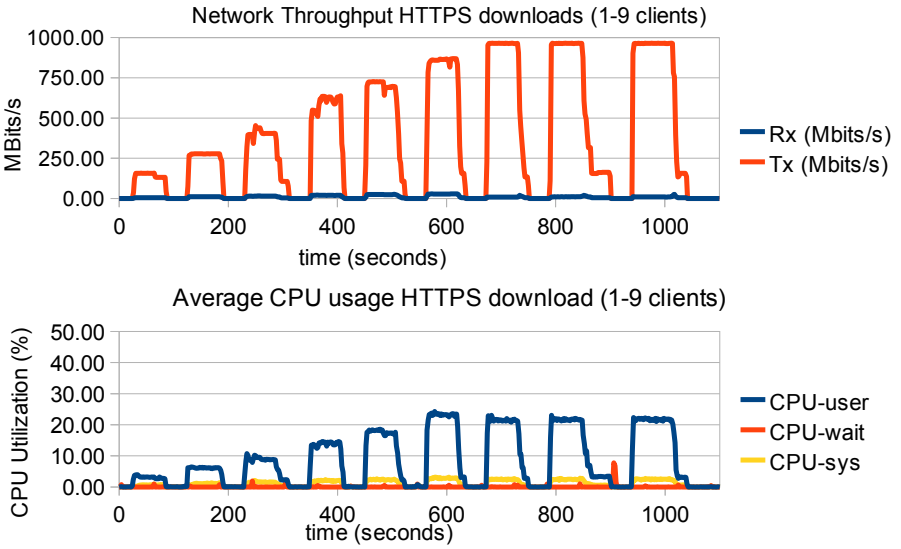

Figure 5: Server statistics while downloading via HTTPS using default SSLv3 cipher suite.

The first set of tests examined the maximum possible download throughput from the server. Each of the clients was configured to use the null device (`/dev/null`) to discard the image file as it received it rather than writing it to disk, eliminating local disk as variable in performance. This test was repeated three times, each time with Repoman serving the images in a different manner. In the first test, each of the clients downloaded images via the HTTP protocol (fig 3). In the second and third tests each client downloaded images via HTTPS, first using the eNULL[13] block cipher (fig 4) and second using SSLv3 default ciphers (fig 5). The image file used in the tests was a 1GB file stored on the local disk of the server.

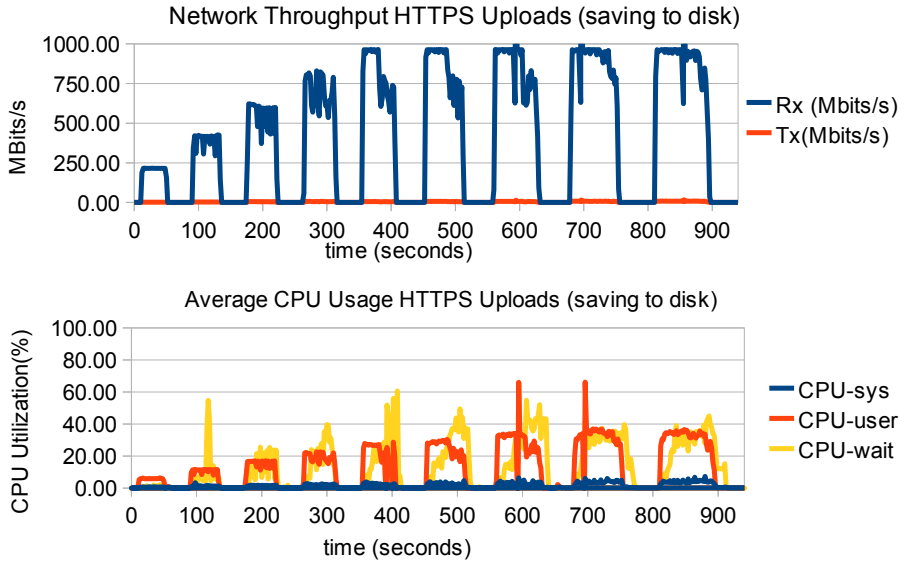

Figure 6: Server statistics while uploading via HTTPS and with the server saving uploads to disk.

As we have shown, it can be seen that the Repoman server is quite capable of saturating a gigabit network with a single HTTP stream to a single HTTP client, while at the same time causing minimal load on the server. As the number of clients increases, the ability to saturate the network remains, and the load on the server remains minimal.

As expected, once the Repoman server started serving images via HTTPS, a substantial performance hit is incurred both in CPU usage and network throughput. The single client throughput drops below 200Mbits/s, and one core becomes fully utilized. From this data it can be seen that the main bottle neck in the single client maximum throughput is found to be the CPU when serving images via HTTPS. As the number of simultaneous clients increases, the CPU usage and network throughput both increase until the network starts to saturate with seven clients, at which point CPU usage starts to remain steady and the network becomes the limiting factor to performance.

In the third download test, the network throughput to a single client again drops slightly when compared to the downloads which used the eNULL cipher. In this case, the higher demand on the CPU to fully encrypt the data stream further limits the network throughput. With a single client, the limiting factor is again CPU. Once the network begins to saturate with seven clients, the main performance bottleneck again shifts from CPU to network throughput.

The second set of tests examined the maximum possible upload performance of the Repoman server. This test was run two times with different server configurations, and followed the same pattern of the first tests in which 1 to 9 simultaneous clients were used. In each test, each client uploaded a 1GB file to the server. The first test (fig 6) was run by having the clients upload images via HTTPS using the default SSLv3 ciphers, with the Repoman server saving the images to disk. In the second test (fig 7), the local disk access was removed as a variable by configuring the server to use the null device to discard the uploaded images rather then saving them to disk.

Uploading images to Repoman must be done over HTTPS for authentication reasons, and exhibits similar performance to downloading via HTTPS. In the first upload test, it can be seen that

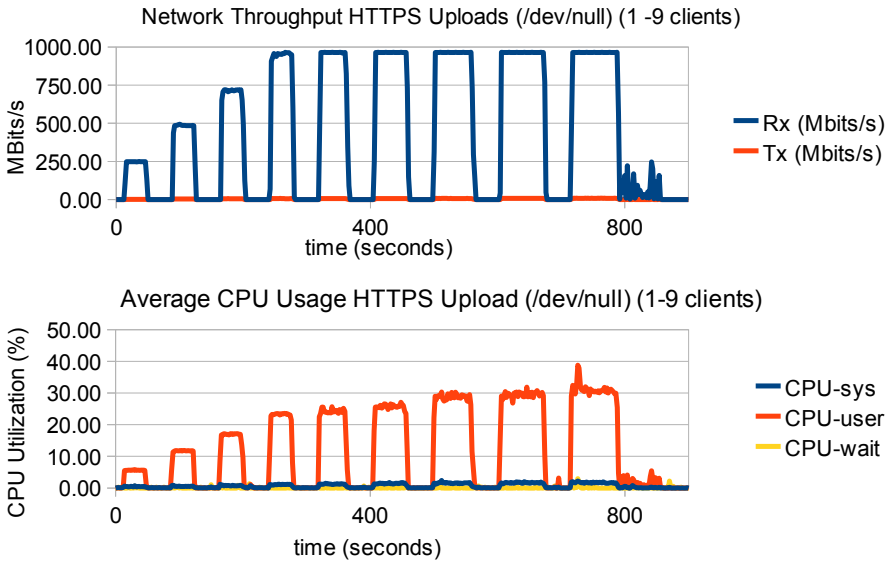

Figure 7: Server statistics while uploading via HTTPS and with the server writing to dev/null.

with a single client uploading one CPU core on the server is completely utilized, making CPU the bottleneck during uploading. As the number of clients increased each additional client fully utilized one core. Saturation of the network occurs as the number of clients reached five at which point network throughput became the factor limiting performance. As the number of clients continued to climb, CPU usage remains consistent and the network remained fully saturated. During this set of uploads, there was also considerable time spent waiting for disk writes on the server.

To see if the disk writes were contributing to low throughput the server was modified to write the upload images to the null device (`/dev/null`) and the set of uploads was run again. As shown in the second test, CPU usage lowered slightly and network throughput slightly increased. In this test the network became fully saturated with only 4 clients uploading images. Although writing to `/dev/null` would never be done in real situations, it illustrates that the upload bottleneck on the server comes from a combination of disk and CPU.

5. Conclusion

Repoman fills the role of a IaaS implementation independent VM image repository suitable for use with Grid X.509 credentials. Repoman has been shown to provide an easy-to-use and secure method of allowing users to upload VM images for use within IaaS clouds without the intervention of an administrator. In addition, Repoman provides tools for users to easily create snapshots of running machines allowing for changes within an image to be saved for later use. This feature allows users to interactively test and debug their code within the same environment which will eventually be used to run batch jobs.

It has been shown that Repoman is capable of saturating an available ethernet connection with a minimal number of clients. This is an important performance characteristic within IaaS clouds as it will allow for more VMs to be transferred and booted within a shorter time span.

The support of CANARIE, the Natural Sciences and Engineering Research Council, and the National Research Council of Canada is acknowledged.

References

[1] P. Armstrong, A. Agarwal, A. Bishop, A. Charbonneau, R. Desmarais, K. Fransham, N. Hill, I. Gable, S. Gaudet, S. Goliath, R. Impey, C. Leavett-Brown, J. Ouellete, M. Paterson, C. Pritchet, D. Penfold-Brown, W. Podaima, D. Schade, R.J. Sobie. Cloud Scheduler: a resource manager for distributed compute clouds. arXiv:1007.0050v1 [cs.DC].

[2] S.Tuecke, V.Welch, D.Engert, L.Pearlman,and M.Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. IETF RFC 3820: http://www.ietf.org/rfc/rfc3820.txt.

[3] K.Keahey, I.Foster, T.Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grid. Sci. Program. Vol. 13 (2005) 265.

[4] Roy Thomas Fielding, Richard N. Taylor. Principled design of the modern Web architecture. TOIT. Volume 2 Issue 2, May 2002

[5] ec2-bundle-vol: A commandline utility provided by Amazon Web Services. http://aws.amazon.com/.

[6] Pylons: A lightweight HTTP web framework for Python. http://pylonshq.com/

[7] Phillip J. Eby. Python Web Server Gateway Interface v1.0.1 http://www.python.org/dev/peps/pep-3333.

[8] Apache HTTP Server: An HTTP web server developed by Apache. http://httpd.apache.org/.

[9] Mod_SSL: A module for the Apache web server used to provide HTTPS. http://www.modssl.org/

[10] Mod_wsgi: A module for the Apache web server which implements a WSGI interface. http://code.google.com/p/modwsgi/.

[11] SQLAlchemy: Flexible SQL Object Relational Mapper for Python. http://www.sqlalchemy.org/.

[12] Python Package Index(PyPi): An online repository of python packages hosted by python.org http://pypi.python.org/pypi

[13] T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2 IETF RFC 5246: http://tools.ietf.org/rfc/rfc5246.txt.