University of Victoria
Faculty of Engineering
Winter 2013 Work Term Report

Shoal: IaaS Cloud Cache Publisher

Department of Physics
University of Victoria
Victoria, BC

Mike Chester
V00711672
Work Term 3
Computer Engineering
mchester@uvic.ca

April 24, 2013

In partial fulfillment of the requirements of the
Bachelor of Engineering Degree

Supervisor's Approval: To be completed by Co-op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (**select one**): ☐ NOT CONFIDENTIAL ☐ CONFIDENTIAL

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____ Fax #: _____

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.

Contents

1 Report Specifications **4**
- 1.1 Audience . . . 4
- 1.2 Prerequisites . . . 4
- 1.3 Purpose . . . 4

2 Introduction **5**

3 Shoal **5**
- 3.1 AMQP and RabbitMQ . . . 8
- 3.2 Scalability . . . 8
- 3.3 Compatibility with Current Systems . . . 8

4 Scalability Tests **9**
- 4.1 RabbitMQ . . . 9
- 4.2 Shoal Server . . . 10

5 Discussion **13**
- 5.1 Improvements . . . 13
- 5.2 Future Work . . . 14

6 Conclusion **14**

7 Acknowledgements **14**

8 Glossary **15**

List of Figures

1 Interaction between Shoal modules, Shoal Agents will send regular interval AMQP messages to Shoal Server. Shoal Clients residing on IaaS clouds will query the REST interface on Shoal Server to retrieve a list of geographically closest Squid servers. 7
2 RabbitMQ message routing between Shoal Agents and Shoal Server. All messages sent will be routed through the Exchange, and put into a queue. Shoal Server will then consume messages off this queue. 8
3 RabbitMQ memory usage 9
4 RabbitMQ server load average at 10000 messages/minute 10
5 Apache average request response time for varying Squids, 1 *mod_wsgi* process 11
6 Apache average request response time for varying Squids, 10 *mod_wsgi* process 11
7 Apache average request per second for varying Squids, 1 *mod_wsgi* process 12
8 Apache average request response time for varying Squids, 10 *mod_wsgi* processes 12
9 Apache Requests Per Second as a Function of Concurrent Requests 13

Shoal: IaaS Cloud Cache Publisher

Mike Chester
mchester@uvic.ca

April 24, 2013

Abstract

The department of Physics at the University of Victoria is involved with running a number of High Energy Physics (HEP) applications. Some of these applications are dependant on specific software available through the CERN Virtual Machine File System (CVMFS). It has been shown possible to run these HEP workloads on remote Infrastructure as a Service (IaaS) cloud resources. Typically each Virtual Machine (VM) makes use of CVMFS, and a caching HTTP file system to minimize the size of VM images and simplify software installation. Each VM must be configured with an HTTP web cache, usually a Squid Cache, in close proximity in order to function efficiently. Regular grid computing sites which use CVMFS to host worker node software use one or more Squid servers for their site. However, each IaaS cloud site has none of the static infrastructure associated with a typical HEP site. Each cloud VM must be configured to use a particular Squid server. Shoal was developed to provide a way to dynamically track Squid servers hosted in an IaaS cloud.

1 Report Specifications

1.1 Audience

The intended audience of this report are members of the High Energy Physics group at the University of Victoria, future co-op students that may work on this or similar projects and the University of Victoria Engineering Co-op Office.

1.2 Prerequisites

A basic understanding of virtualization and distributed computing. General knowledge of HTTP, AMQP and RESTful interfaces may be useful to the reader.

1.3 Purpose

This report is intended to explain how Shoal can be used to compliment current Squid tracking services and analyze Shoal performance in a production environment.

2 Introduction

The University of Victoria High Energy Physics Group (HEP) is involved in a number of particle physics experiments around the world. One such experiment is the ATLAS detector at the European Organization of Nuclear Research (CERN) in Geneva, Switzerland[7]. The ATLAS detector was recently used in the successful search for the Higgs Boson[2]. The ATLAS detector generates petabytes of data that must be analyzed. In recent years the group has begun running the applications inside Virtual Machines (VMs) on Infrastructure as a Service (IaaS) clouds. VMs allow the encapsulation of a set of software and an Operating System inside a single file which can be executed by a virtual machine hypervisor such as Xen, KVM, or VM Ware. IaaS clouds offer users the ability to boot VMs on remote computing resources using an Application Programming Interface (API). This allows users to be able to create VMs which are perfectly suited to the the needs of the application which they wish to execute. The HEP applications this report focuses on are termed embarrassingly parallel, meaning the workload runs in a single process without any interprocess communication. A single job can take upwards of 12 hours to complete. VMs are allocated to run these workloads (a VM running a workload is referred to as a worker node) on IaaS clouds.

Typically, the HEP workloads are run on grid computing sites, these sites contain many static infrastructures to facilitate worker nodes in order to efficiently utilize all available resources. IaaS cloud sites do not contain any of the associated static software and servers required to run HEP workloads, but it has been shown possible to run these HEP workloads on IaaS clouds. CERN has developed an HTTP read-only file system called the CERN Virtual Machine File System (CVMFS) which hosts the software needed to run HEP workloads. CVMFS presents a remote directory as a local file system, in which the client has read access to all available files. On a clients first access request, a file is downloaded and cached locally, and all subsequent requests for that file will use the cached copy. In the case where multiple worker nodes are trying to access the same file an HTTP web proxy can be used to cache the contents, so all subsequent requests for that file will be delivered from the HTTP proxy server. A commonly used HTTP web cache is called a Squid server[8]. Using a Squid server reduces traffic, as the first request is the only request that will hit the CVMFS server, and all future requests for that file will instead hit the Squid server.

Current methods used to accommodate IaaS cloud HEP workloads is the use of a manually updated static list of Squids. Each worker node running a HEP workload and CVMFS will use this list to obtain the available Squid servers it can use, typically these Squid servers are located within a different cloud and thousands of miles away. In order to run as efficiently as possible, a Squid server should be located within the same cloud as the worker nodes running CVMFS. Due to the dynamic nature of IaaS clouds a static list of Squid servers is not sufficient to fully utilize IaaS cloud resources and track dynamically booted Squid servers. A way to dynamically publish and advertise IaaS cloud Squid servers is needed.

The report outlines the solution for Squid tracking, called Shoal[1]. This report also contains benchmark tests of Shoal running in a simulated production environment.

3 Shoal

In order to address the need for a method of using dynamically created Squids a new software application called Shoal was created. Shoal is being developed with the goal to utilize parts of existing Squid tracking systems by providing a completely dynamic aspect. Shoal is broken into three logical modules, a server, agent, and client. Each package will be uploaded to the Python Package Index (PyPI) which is the standard method of distributing new components in the Python language[1]. Each component is designed to provide the functionality of different parts of the system as follows:

Shoal Client - Worker nodes use Shoal Client to query the RESTful[2] interface, provided by Shoal Server, to retrieve a list of geographically closest Squid servers and configure the CVMFS HTTP proxy server.

[1] A shoal is a group of squids, much like a murder is a group of crows.

[2] Representational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web.

This will be the most installed package as each worker using Shoal will contain an installation of Shoal Client.

Shoal Agent - A Daemon process run on Squid servers to send an Advanced Message Query Protocol (AMQP) message to Shoal Server on a set interval. Every Squid server using Shoal will run Shoal Agent. Shoal Agent will run on boot of a new Squid server and advertise its existence within a few milliseconds of gaining network connectivity to Shoal Server. Having Shoal Agent run allows Shoal Server to maintain a list of Squids that is always up-to-date. When the Squid server goes offline, the heartbeat messages will stop being sent, and Shoal Server will cleanse the Squid server out of its tracked Squid list. There may be a slight delay between the time a Squid goes offline and the Squid server is removed from Shoal Servers list, but CVMFS can contain a list of Squid servers to use in its configuration so it will continue working and use the next available Squid.

Shoal Server - Shoal Server is where the majority of the logic resides. Only one Shoal Server is needed for our purposes and should have a globally available IP address. The server is responsible for the following tasks:

1. Maintaining a list of active Squid servers in volatile memory and handling AMQP messages sent from active Squid servers.
2. Providing a RESTful interface for Shoal Clients to retrieve a list of geographically closest Squid servers.
3. Providing basic support for Web Proxy Auto-Discovery Protocol (WPAD).
4. Providing a web user interface to easily view Squid servers being tracked.

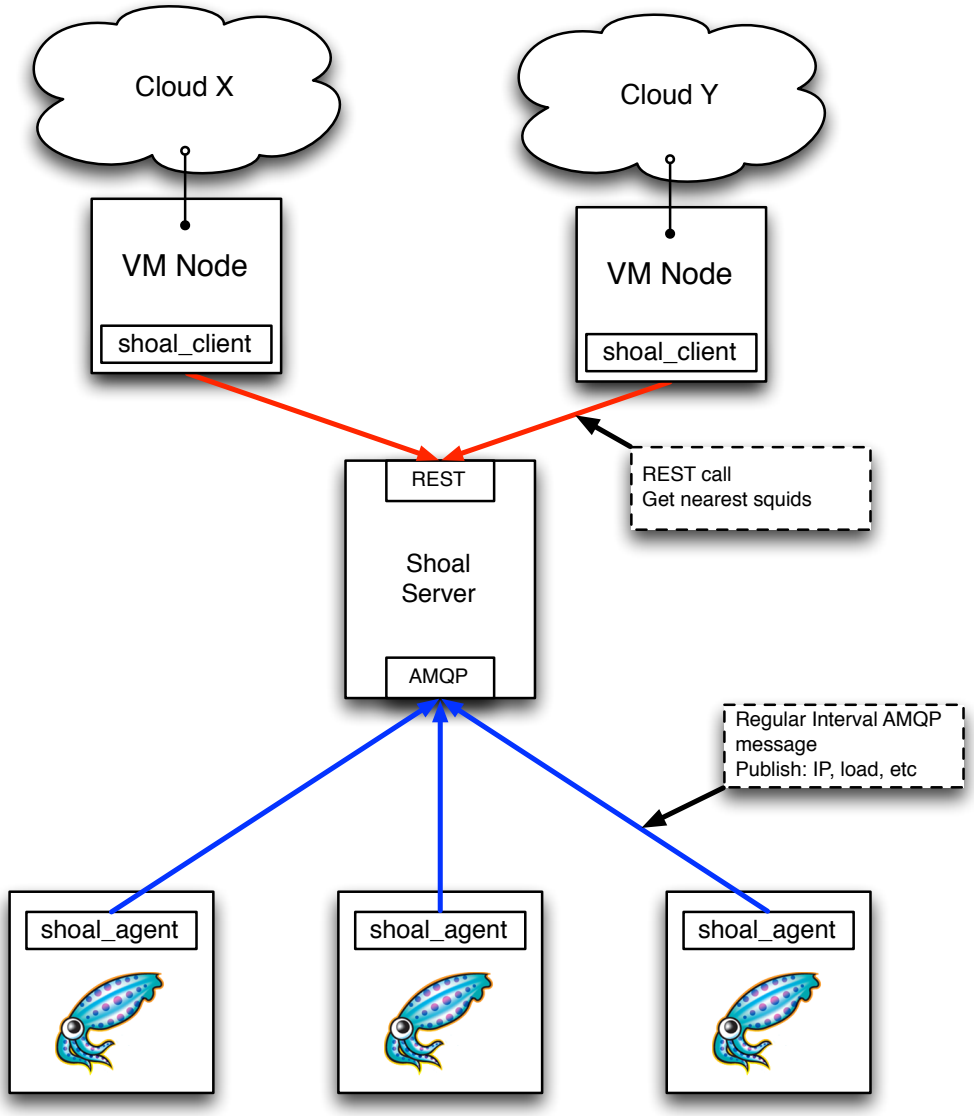

Figure 1: Interaction between Shoal modules, Shoal Agents will send regular interval AMQP messages to Shoal Server. Shoal Clients residing on IaaS clouds will query the REST interface on Shoal Server to retrieve a list of geographically closest Squid servers.

Figure 1 shows an overview of Shoal and the interaction between each module. Shoal Server runs at a centralized location with a public address. For agents (or Squid servers), Shoal Server will consume the heartbeat messages sent and maintain an up-to-date list of active Squids. For clients, Shoal Server will return a list of Squids organized by geographical distance, then load. For regular users of Shoal Server, a web server is provided. The web server generates dynamic web pages that display an overview of Shoal. All tracked Squid servers are displayed and updated periodically on Shoal Servers web user interface, and all client requests are available in the access logs.

3.1 AMQP and RabbitMQ

AMQP is the backbone of Shoal Server. All information exchanges between Shoal Agent (Squid Servers) and Shoal Server are done using this protocol, and all messages are routed through a RabbitMQ Server. Figure 2 gives a basic high level overview of message routing between Shoal Agents and Shoal Server.

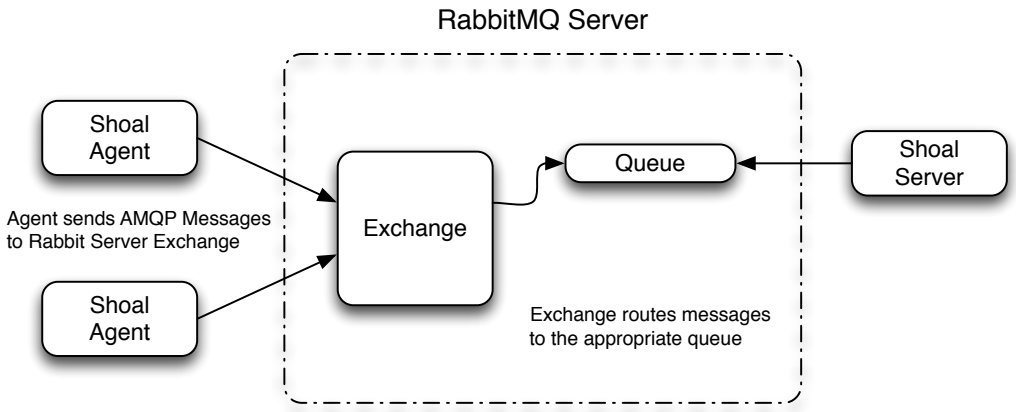

Figure 2: RabbitMQ message routing between Shoal Agents and Shoal Server. All messages sent will be routed through the Exchange, and put into a queue. Shoal Server will then consume messages off this queue.

The exchange will route the message according to a routing key that is set on the agent, and can be dynamic. The message will enter through the exchange and will then be delivered to a specific queue. Shoal Server creates this queue, and will receive all messages sent to the specific exchange. If Shoal Server is not running, all messages sent to the exchange will be discarded as there will be no queue for them to be routed to. More complicated routing can be done with the exchange-queue system, but currently it is a simple message hand-off from exchange to queue.

3.2 Scalability

Shoal is being developed with scalability as the central concern. In order to be a viable replacement to current systems, Shoal needs to be scalable and able to handle thousands of web requests and information exchanges between Squid servers and clients. Shoal utilizes AMQP as its backbone for communication, using a RabbitMQ server. AMQP and RabbitMQ Server is known to be highly scalable and robust[3], making them the ideal candidate for information exchanges. Shoal was also created with the idea of being simple but efficient. Shoal employs concepts such as asynchronous message consumption and threading, these allow Shoal to use little to no system resources under minimal load, and ramp up as load increases. Analysis of Shoal's scalability is discussed in section 4 of this report.

3.3 Compatibility with Current Systems

Ideally the best solution is one that entirely removes the majority of the user interaction in the tracking system. However, Shoal was designed to fall back to a static list in the event of a failure in the dynamic system. The CVMFS software can be configured to use many Squid caches by providing a prioritized list of Squid server URLs. Using a prioritized list of Squids servers available to CVMFS allows Shoal to be integrated with current systems by providing a dynamic Squid IP address at a higher priority than the static list of Squids. Since Shoal can function with or without the current static list, current systems will continue to function as normal even if Shoal is unable to provide a dynamic Squid server URL.

4 Scalability Tests

This section focuses entirely on Shoal Server and its underlying messaging infrastructure as Shoal Agent and Shoal Client are simple scripts and do not generate any noticeable load on systems. This section was broken into two parts, one covering the scalability of Shoals underlying message system, RabbitMQ, and one covering Shoal Servers ability to respond to client requests while maintaining a list of active Squid servers.

4.1 RabbitMQ

For the following tests a RabbitMQ server was configure on a Virtual Machine running Scientific Linux 6.3 with 1GB of RAM and a single processing core. With no incoming or outgoing messages, RabbitMQ idles around 31MB of memory usage. Each test spans approximately 3 minutes. Figure 3 shows the memory usage of the RabbitMQ server as it routes 100 and 10000 messages per minute.

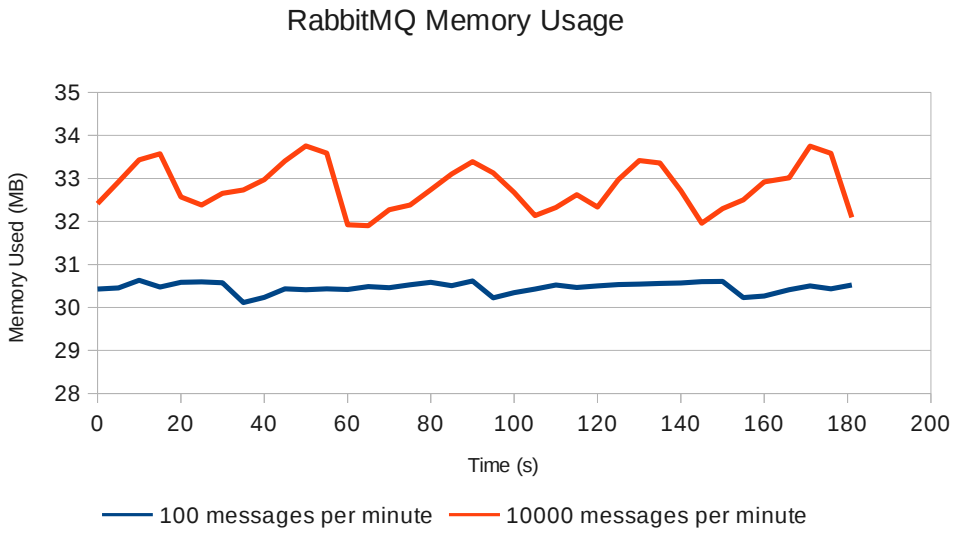

Figure 3: RabbitMQ memory usage

As shown in Figure 3, the RabbitMQ server has no noticeable increase in memory usage at a mere 100 messages per minute and idles around 31MB. Figure 3 also shows the RabbitMQ server barely increases its memory footprint when routing 10000 messages per minute. Figure 4 shows the load average on the RabbitMQ Server during the test.

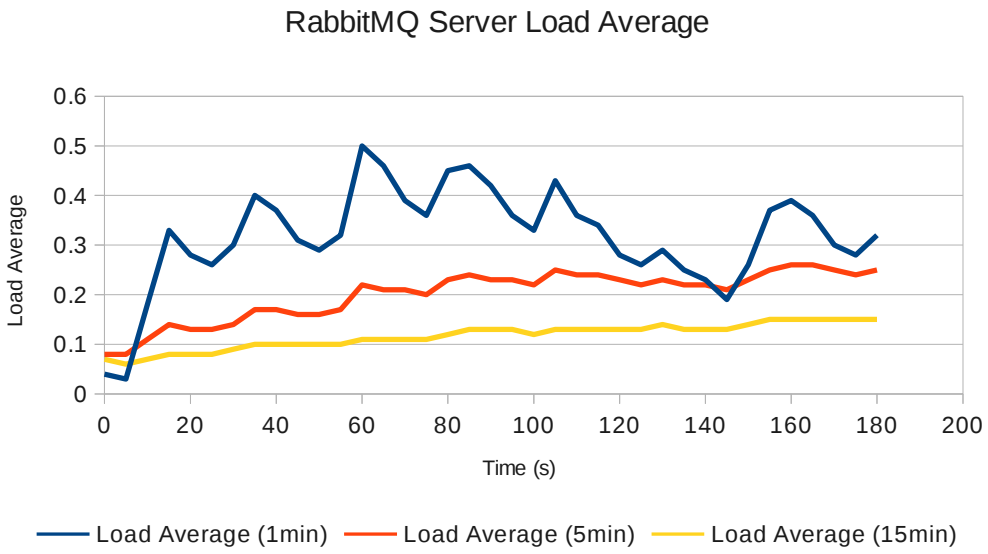

Figure 4: RabbitMQ server load average at 10000 messages/minute

As shown in figure 4, the load generated by routing 10000 messages per minute is consistently below 0.5 for the duration of the test.

By choosing AMQP as the backbone of Shoal it is able to scale greatly as shown by the previous tests. By increasing the number of messages being routed by a factor of 1000, the RabbitMQ server barely increased its memory consumption. The one minute load average on the VM was also shown to be consistently below 0.5 for the duration of the tests. RabbitMQ has successfully shown it will be able to handle the load that will be generated by tracking dynamic Squids on IaaS clouds.

4.2 Shoal Server

As Shoal Server is where the majority of the workload happens, it must be capable of handling thousands of requests to be considered a viable option for Squid tracking. Most of the load is generate via the RESTful API calls to Shoal Server. It has been shown the load generated from tracking Squid servers and consuming messages from the RabbitMQ server is minimal in comparison to HTTP requests. If Shoal were to replace current systems of tracking Squids, it would need to be able to handle thousands of worker nodes calling it periodically. Since each RESTful API call to Shoal Server is unique the distance to the closest Squid must be calculated for each call, as the number of tracked Squids increases, so will the time to complete each web request. Apache uses a plug-in called *mod_wsgi*[4] that can run Python based web servers. Using *mod_wsgi*, Apache can run multiple *mod_wsgi* processes to handle more requests simultaneously.

In order to ensure that shoal will be able to function at the scale required a number of benchmark test were performed. An IBM 16 core x86_64 machine with 64 GB of ram was configured with the Shoal Server. A client machine with the Apache benchmarking tool ab[5] was set up to access the shoal server with a configurable number of parallel accesses, thus simulating the load from many worker nodes.

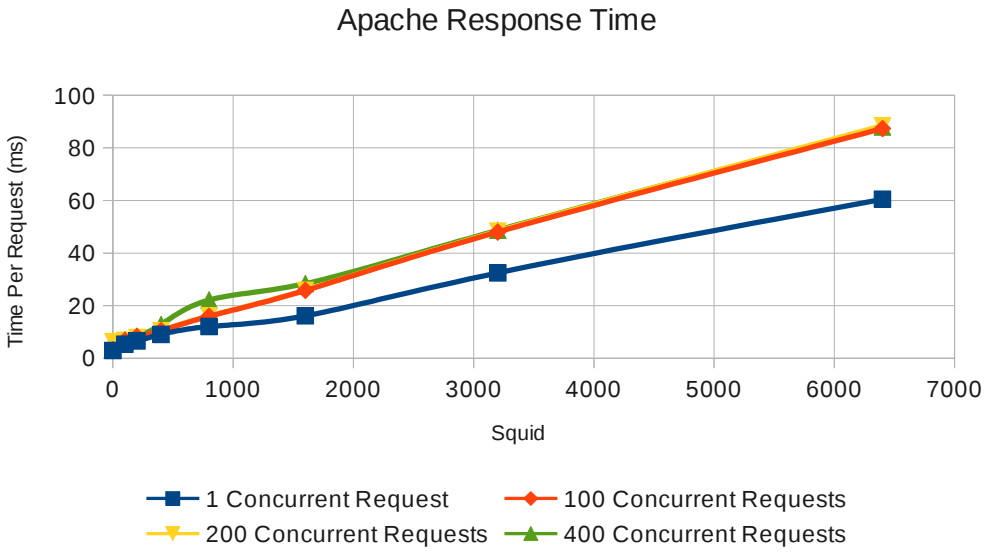

Figure 5: Apache average request response time for varying Squids, 1 *mod_wsgi* process

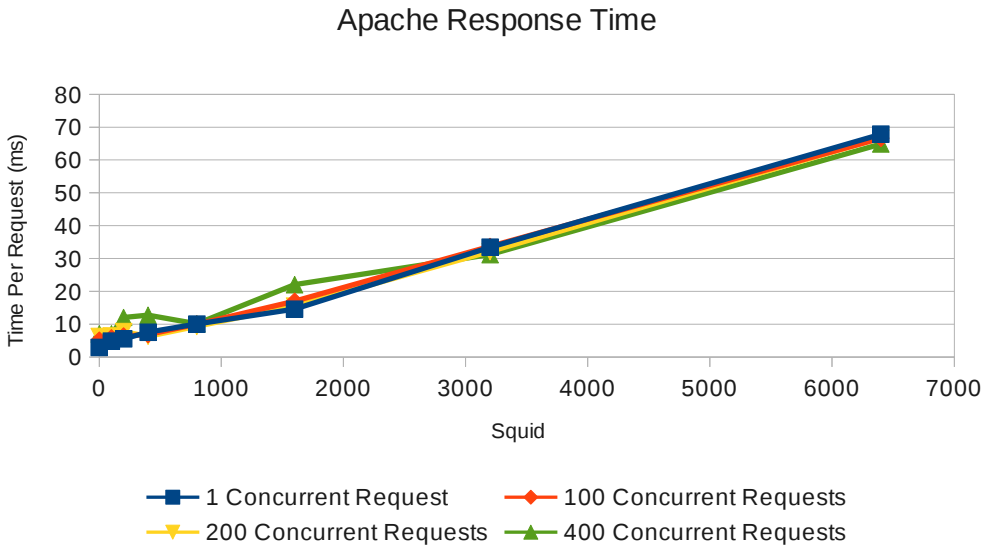

Figure 6: Apache average request response time for varying Squids, 10 *mod_wsgi* process

Figure 5 shows the Shoal server response time as a function of the number of Squid servers which are advertising to the Shoal Server. The test was conducted using 1, 100, 200, and 400 concurrent connections to the server. As was expected, the response times increased with increasing tracked Squid servers. The time per individual request is not decreased by increasing the number of process (as show in Figure 6)

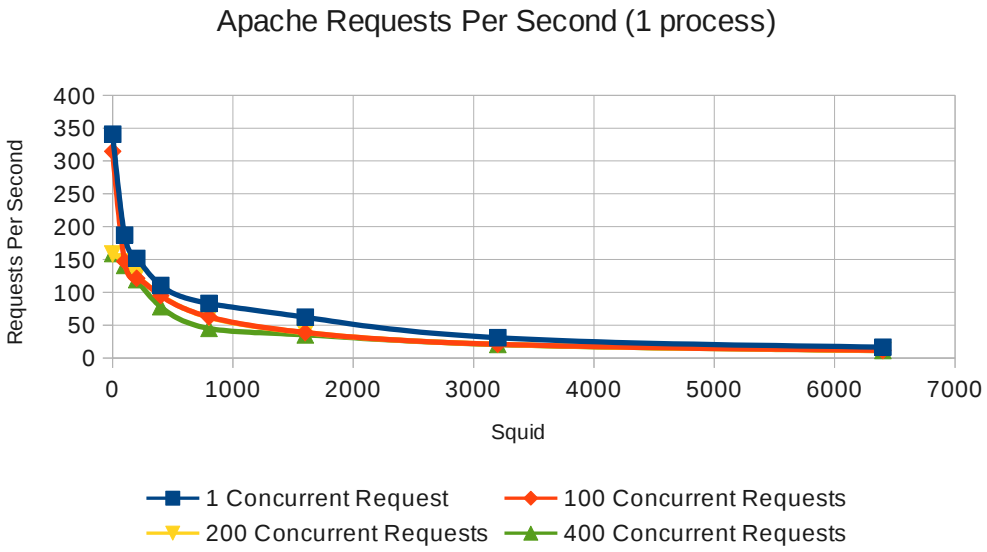

Figure 7: Apache average request per second for varying Squids, 1 *mod_wsgi* process

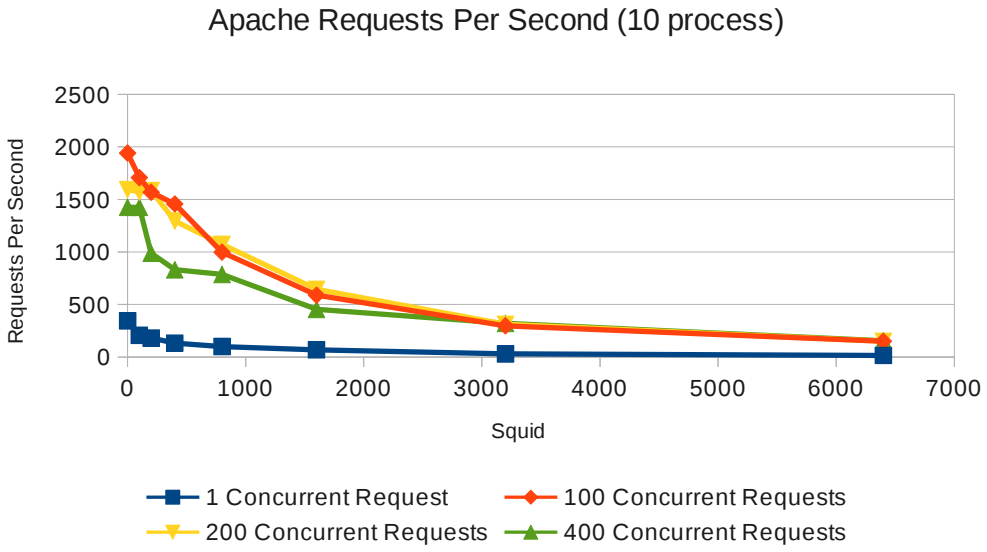

Figure 8: Apache average request response time for varying Squids, 10 *mod_wsgi* processes

Figure 7 and Figure 8 shows the same benchmarking with the data plotted as total number of request which can be satisfied per second. In Figure 8 we see that the number of requests per second can be increased by increasing the number of *mod_wsgi* processes running on the server.

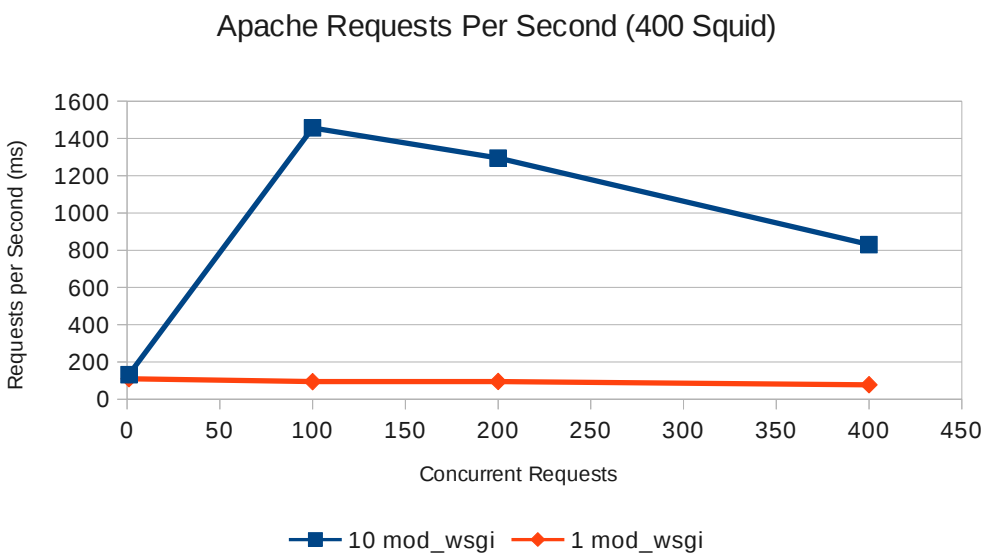

Figure 9: Apache Requests Per Second as a Function of Concurrent Requests

Figure 9 shows Apache is able to serve on the order of 10 times more requests per second with 10 *mod_wsgi* processes when the number of concurrent requests is large (greater than 100). With a small number of concurrent requests, some Apache *mod_wsgi* processes may remain idle, resulting in lower request per second metrics as seen in Figure 9.

As expected Shoal Server performs within an acceptable range to be integrated into current systems. When tracking under 1000 Squid servers Shoal can manage to handle thousands of requests per minute with only one *mod_wsgi* process. The load generated on the Apache server was proportional to the number of Apache *mod_wsgi* processes.

The algorithm for finding the nearest Squid server could be substantially improved if a temporary cache was used for different IP subnets, this would allow Shoal Server to use a hash table to store nearest Squid servers reducing the majority of RESTful API calls to complete in O(1) time, whereas in its current implementation each web request requires O($n$log$n$) time, where n is the number of tracked Squids. Due to time constraints Shoal was only thoroughly tested on Apache, but in theory could run on other web servers which support Python.

5 Discussion

5.1 Improvements

Shoal Server is developed with the idea that all data and information be stored in volatile memory. This unfortunately makes Apache optimization rather difficult as multiple Apache processes do not share the same memory, thus having multiple *mod_wsgi* processes means each process would contain its own list of tracked Squid servers which could lead to consistency issues. A potential work around for this could be the use of a time-to-live attribute on RabbitMQ queues. Currently Shoal Server will establish a queue with the RabbitMQ server, and the exchange will route messages to the unique queue, Shoal Server will then acknowledge and remove these messages from the queue. Instead a polling approach could be taken with one generic queue that all instances of Shoal Server will consume. Messages will still be routed to this generic queue, but instead of Shoal Server acknowledging and removing each message, the queue itself will remove messages after a set time. This new approach simulates the idea of a static database, but does not introduce

any new technologies. Although some caveats that may need to be accounted for is the possibility of a race condition for when a new Apache *mod_wsgi* process starts and a client tries to retrieve a list of nearest Squid.

There are some additional improvements in algorithm design and enhancements in the web user interface that could improve performance. The majority of the load generated from Shoal is the distance calculations performed for each RESTful API call. This could be alleviated by caching previous distance calculations and mapping IP subnet masks to a specific Squid server/cluster. This would reduce computations done on Shoal Server ten-fold as this information could be stored in a hash table, which are known as a high performance input/output data structure. Other enhancements that could be developed involve message routing with RabbitMQ Server. Shoal Server could subscribe to multiple queues and filter messages based on clouds, error logs, etc. This filtered information could be displayed on Shoal Servers web user interface and allow users to quickly see all Squids running on a specific cloud, or show Squid servers that have generate error logs.

5.2 Future Work

Future work for Shoal should consist of testing and optimization of Apache deployment. Time should be spent investigating the best approach to synchronize data between *mod_wsgi* processes or investigating ways to optimize a single *mod_wsgi* process to handle more requests. Shoal will need to be optimized with Apache to meet future requirements on the system. Some other areas of work include writing of unit tests to ensure future updates and improvements added to Shoal do not break other areas of the system. Time could also be spent improving the nearest Squid algorithm and caching results into a temporary hash table that updates periodically to keep the cached data fresh.

6 Conclusion

Being able to fully utilize and efficiently run HEP workloads within IaaS clouds is extremely important. Having a way to track dynamic Squids and advertise their existence was needed. Early testing has shown Shoal is able to scale sufficiently to accommodate early adopters and able to perform well under some more extreme loads. First impressions indicate Shoal will be able to perform well in a production environment and run beside current Squid tracking methods without any noticeable performance hits. Shoal's next stage should be small scale deployment on production systems while development should continue to work towards higher optimization on the Apache web server and unit testing.

7 Acknowledgements

I would like to thank Dr. Randall Sobie for this work term opportunity and the Natural Sciences and Engineering Research Council of Canada for partially funding my work. Additional thanks to Ian Gable for guidance on this work term, and all members of the HEP-RC group.

8 Glossary

Cache A collection of data duplicating original values elsewhere.

Cloud The use of computing resources that are delivered as a service over a network.

Daemon A computer program that runs as a background process.

DHCP Dynamic Host Configuration Protocol, a network protocol used to configure devices so they can communicate using the Internet Protocol.

DNS Domain Name System, a hierarchical distributed naming system for computers, services, or any resource connected to the Internet or a private network.

Grid Computing A federation of computer resources from multiple locations to reach a common goal.

IaaS Infrastructure as a Service, basic cloud-service model. Typically IaaS providers offer physical or (more often) VMs to users. IaaS providers may also offer VM image libraries, storage, IP addresses and software bundles.

Load Average The average processing load on the system over a period of time. It is conventionally displayed as three numbers, a 1-minute, 5-minute, and 15-minute average.

mod_wsgi Apache HTTP Server module that provides a WSGI compliant interface for hosting Python 2.3+ based web applications under Apache[4].

PyPI Python Package Index is a public repository of software for the Python programming language.

Shoal Custom open source software developed to provide a solution for tracking dynamic Squid cache servers. Also commonly used to describe a group of Cephalopoda of the order Teuthida.

Squid A proxy server and web cache service.

VM Virtual Machine, an instance of a machine(computer) running in software.

WPAD Web Proxy Auto-Discovery Protocol. A method used by client to locate the URL of a configuration file using DCHP and/or DNS discovery methods.

WSGI The Web Server Gateway Interface defines a simple and universal interface between web servers and web applications or frameworks for the Python programming language[6].

References

[1] PyPI - the Python Package Index, [Online].
Available: https://pypi.python.org [April 16, 2013]

[2] ATLAS Colaboration, Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC, arXiv:1207.7214v2 (Phys.Lett. B716 (2012) 1-29)

[3] RabbitMQ - Messaging that just works, [Online].
Available: http://www.rabbitmq.com [April 04, 2013]

[4] modwsgi - Python WSGI adapter module for Apache, [Online].
Available: http://code.google.com/p/modwsgi/ [April 18, 2013]

[5] ab - Apache HTTP server benchmarking tool - Apache HTTP Server, [Online].
Available: http://httpd.apache.org/docs/2.2/programs/ab.html [April 23, 2013]

[6] WSGI - WSGI.org, [Online].
Available: http://wsgi.readthedocs.org/en/latest/ [April 18, 2013]

[7] ATLAS experiment at the CERN Laboratory in Geneva. http://www.cern.ch/

[8] Squid: Optimising Web Delivery, [Online].
Available: http://www.squid-cache.org [April 24, 2013]