University of Victoria
Faculty of Software Engineering

Dynamic Squid Web Cache Monitoring

Department of Physics
University of Victoria
Victoria, BC

Robert Prior
V00699787
Work Term 4
Software Engineering
rprior@uvic.ca

January 10, 2014

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

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

List of Figures

Dynamic Squid Web Cache Monitoring

Robert Prior rprior@uvic.ca

January 10, 2014

Abstract

Shoal is a service and software product developed by the University of Victoria High Energy Physics group. It provides clients a dynamic list of the nearest Squid caches. As caches go online or offline, Shoal keeps an up to date list. Shoal was taken from a beta state to production. SSL x509 certificates support was added to secure communication between the web caches and the server. The minimum requirements were reduced and RPM were packages provided to facilitate a large user base. Phantom, a service for scaling VMs, was investigated. Puppet, a tool to automatically configure systems, was used to make installation of the Shoal caches automatic. The overhead of SSL was measured and found to be tolerable given the frequency of messages between the caches and the server. Another solution, Airbnb's SmartStack, was also investigated. While being more feature rich than Shoal, SmartStack is also early in development and the extra features add complexity to configuration.

1 Report Specification

1.1 Audience

The primary intended audience of this report are the members of University of Victoria's High Energy Physics group, the future maintainers of Shoal and the Engineering Co-op Office.

1.2 Prerequisites

A basic understanding of Linux operating systems and some knowledge of distributed computing is required.

1.3 Purpose

The purpose of this report is to describe the current state of a software product called Shoal developed by the HEP computing group and the University of Victoria. New features added to the product are described as well as the rational for those additions.

2 Introduction

The Large Hadron Collider (LHC) at CERN in Geneva, Switzerland produces a large amount of data. Particles are accelerated and smashed together roughly 600 million times a second (40 million collisions per second for the ATLAS experiment[2]). This generates around 15 petabytes of data every year[1]. All of this data is pushed out around the world for analysis. This is done over the Worldwide LHC Computing Grid (WLCG)[5]. The WLCG is made up of tiers; each responsible for a portion of the data. Tier 0, the CERN Computer Centre, has all the data passed through it and stores the raw data generated by the LHC. Tier 1 centres are large centres which keep a portion of the raw data, and distribute data to Tier 2. There are ten Tier 1 centres located around the world including one in Vancouver at TRIUMF. Tier 2 are generally specific universities/science facilities conducting analysis.

Some sites within the WLCG process data using Virtual Machines (VMs). These VMs are set images; they have a specific OS and set of software. Any other software needed for processing must be brought in. This is done by the VMs using CERN Virtual Machine File System (CVMFS)[6]. CVMFS is a network file system; data files are accessed as if they were local but are requested from a remote source. CVMFS only supports retrieving data; data cannot be remotely modified. To work efficiently, CVMFS is used with HTTP web caches to store data retrieved. Data passes through the cache and is available there for future use. Next time the data is requested, it is brought in from the cache instead of from a remote location. CVMFS is typically uses Squid, a free (GNU GPL) open source caching proxy[19].

CVMFS uses a static list of Squid caches; if a new Squid cache is made, configuration files need to be changed manually to use it. CVMFS will not be able to find new caches as they are created. CVMFS needs to have a mechanism for dynamically finding caches. As new caches are created (or shutdown), CVMFS should not need to be reconfigured. The University of Victoria's High Energy Physics group have been developing a service to handle dynamic Squid caches called Shoal[7]. Shoal is a useful piece of beta software, though there were a number of outstanding deficiencies that prevented it from being deployed over a large scale in a distributed computing environment. This report will describe the features and development required to bring Shoal to a production deployable product.

The two most critical areas in need of improvement were authentication between the shoal-agent Squid caches and the shoal-server (described in section 3.3 implementation details in A.2) and packaging for production deployment (sections 3.4 and 3.5 more details B.1 and B.2). In order to insure low performance overhead to implementing the proposed security implementation, benchmarks were performed (section 4).

2.1 Shoal Overview

Shoal[1] is written in Python and separated into three branches: shoal-server, shoal-agent and shoal-client. All three have a similar directory layout; the main script is provided along with a configuration file and a configure parser in a sub-module.

shoal-server provides a list of Squid caches to clients who request them. shoal-agents send AMQP messages to post their existence. The server uses the provided information to generate location and load data for the cache. This information is accessible through a human readable website and through a REST interface. Any HTTP Get requests can be sent to the server to receive a JSON populated with data for the nearest variable number of caches. Squids which stop sending information to the server are purged from the list of caches periodically. Typically shoal-server would run on a machine with Apache and RabbitMQ though, they are not required to be together. The server is packaged with the all the web files (HTML/CSS/JS) needed for the web interface as well as an Apache plug-in script.

shoal-agent posts its existence to a shoal-server using RabbitMQ. Shoal-agent should be only used on machines also running Squid. The shoal-agent determines outgoing kilobytes per second and sends that information along with its public and private IP. The shoal-agent is packaged with a daemon script to ensure it runs on start up along with Squid.

shoal-client Reads and parses the Squid data from the server. It then updates the CVMFS configuration file to point to the nearest Squid.

[1] The name of Shoal comes from a group of squids (like how a group of crows is called a murder)

2.1.1 Libraries Used

RabbitMQ is the AMQP broker used. It stores a set of queues storing messages with specific IDs. Shoal-server, on start, creates a queue in RabbitMQ that persists as long as the server remains running. Any AMQP messages received and properly tagged are added to this queue; shoal-agent sends these messages[17].

Apache is a HTTP web server. It supports shoal-server to manage load and distribute cache data to clients. It is also used to distribute shoal software using an RPM repository[9].

pika is a Python interface for AMQP. It simplifies sending AMQP messages by generating and formatting the message header. It is used in both shoal-agent and shoal-server.[8].

netifaces is a Python module that provides a cross platform way to get a machine's public or private IP address. Used by shoal-agent to provide an IP for shoal-server.[10].

pygeoip gets location data from an IP address. Given an IP, it can return data such as city, country, area code, as well as latitude and longitude. This data is used to calculate distance between a given shoal-client and the set of shoal-agents[11].

web.py is a minimalistic web framework. It is used by shoal-server to supply the list of available caches[12].

simplejson is a JSON parsing module that was adopted into the Python standard in version 2.6. In older python versions, this is a separate module that must be included[13].

3 New Features

3.1 Minimizing shoal-client

Low installation requirements was necessary for Shoal. Users who want to get a list of Squids should be able to use any VM image or OS version. Selecting Python version 2.4 as the minimum version allows a very wide range of OSs that can be used (as Python 2.4 is 9 years old[15]). Shoal-client relied on the JSON parser module from standard Python 2.6+. The parser is available as a separate module called simplejson for older Python versions, but this adds a dependency to shoal-client. A custom JSON parser was built to handle the server data. It was not a fully featured JSON parser; arbitrary JSON strings could not be parsed. By adding this restriction, the parser can be reduced and simplified compared to a full JSON implementation. In the case of Shoal, a full JSON parser is not necessary. Adding a new data field requires a change in all three modules. The shoal-agent needs to be modified to gather the new data, shoal-server needs to store the new data and shoal-client needs to use it. Adding a new field to the custom parser is trivial but it is another step. A sample from the parser is given in appendix A.1. Every Squid is expected to be identified by a number and then have a dictionary of properties. These properties are uniform across Squids; every Squid has the same fields. If Python 2.6 becomes the minimum Python version or use of an external module is acceptable, this code should be replaced by simplejson/the standard JSON module.

3.2 Phantom Decision Engine

One of the main purposes of Shoal is to mitigate and speed network transfer. If the caches become overburdened by many clients, performance will degrade. Shoal was built to handle a dynamic number of caches; the server can handle a fluctuation in the number of active shoal-agents. More caches should be started if they become overburdened to distribute the load. Similarly, caches should only be active if they are in use. Phantom solves this problem[20].

Phantom is a service provided by Nimbus[21] that provides a mechanism for automatically scaling a number of active VMs. Nimbus manages an invite only instance that supports a set list of clouds (Amazon EC2 and FutureGrid; a private cloud between a few Universities[22]) and can handle multiple types of clouds at once (Nimbus, OpenStack etc.). It is still in development however, and is impossible for a user to add their own clouds. Until that restriction is lifted, Phantom can not be used in a production server without running a custom instance.

Phantom has a web user interface to use its scaling features. VMs can be booted and scaling can be setup using their built in sensors. A sensor measures performance metrics (memory used, number of processes etc.)

and reports that information back to Phantom. Phantom uses that data to determine if it should scale the number of active VMs.

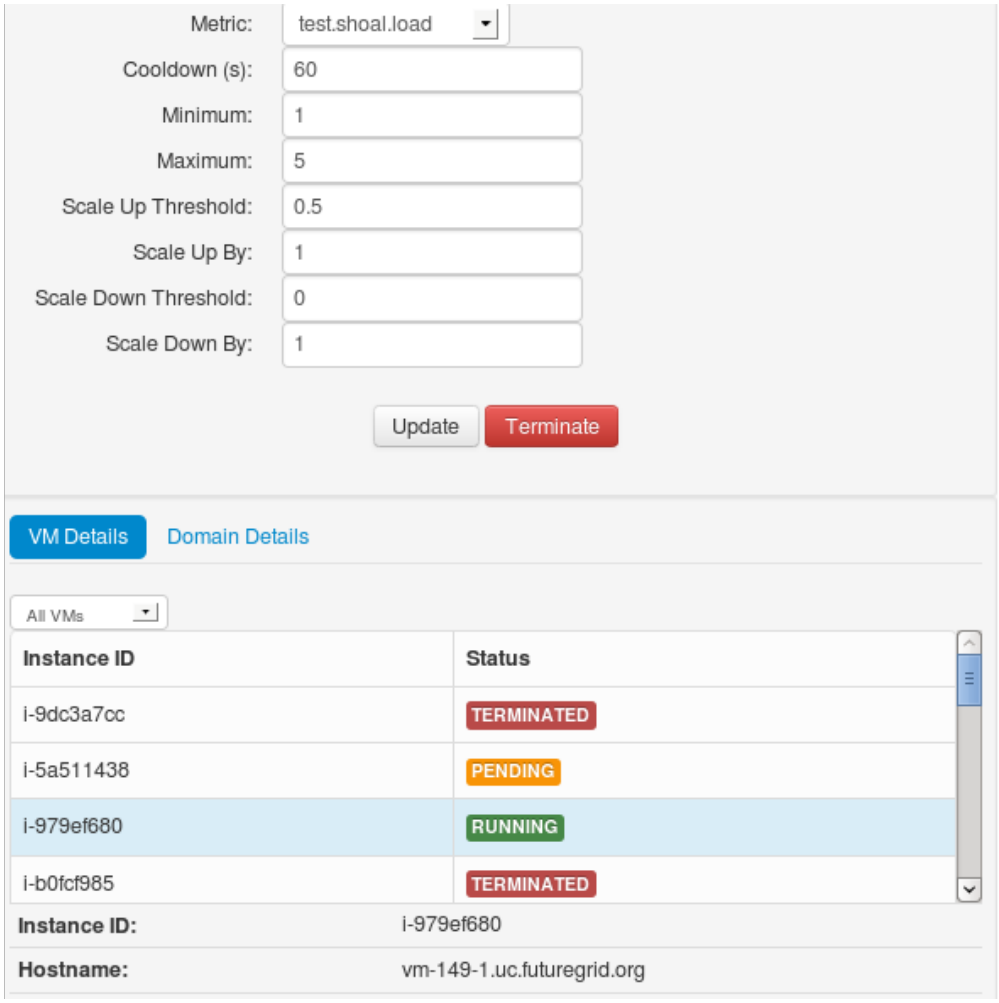

Figure 1: Phantom Web Interface in Use

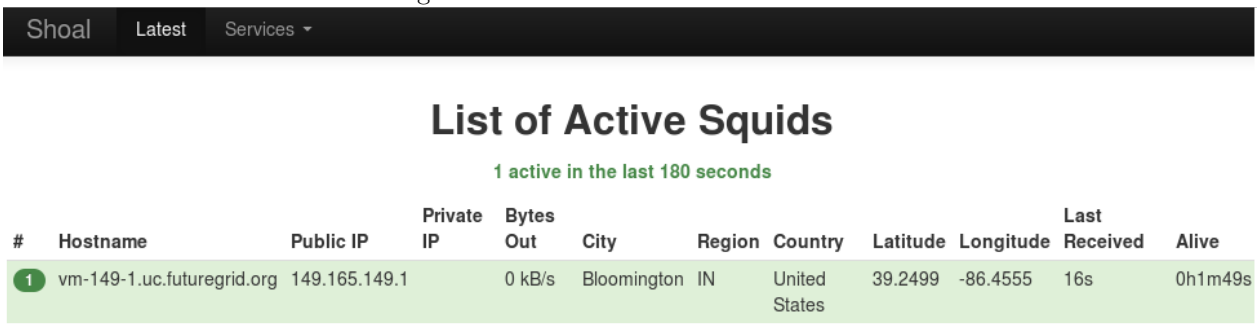

Figure 2: Shoal Web Interface

Figure 1 shows a running example of the Phantom web interface and with the corresponding shoal server list of active shoal agents in Figure 2. The figure shows the list of VMs, including active, recently terminated and those pending launch. A custom sensor was run that periodically reports (once every 60 seconds) out going kilobytes to Phantom. When this value passes the scale up threshold, the number of VMs will increase by the scale up by value. One potential problem with the web interface method is that each instance is running its own sensor. If one VM has a high load and another zero load, Phantom rapidly fluctuates the number of VMs. A VM pending launch (from the high load sensor value) could be immediately terminated

after booting due to a zero load sensor. Phantom allows use of custom decision engines, which are scripts to handle scaling manually. Phantom will not scale on its own but rely on commands from the decision engine. Phantom, in that case, just provides an API for launching and terminating VMs on multiple clouds.

A decision engine script needs to request an API token from Phantom using user credentials. The API is RESTful interface (Representational State Transfer); all of Phantom's resources can be retrieved or manipulated using HTTP GET and POST messages. To add or modify a domain (set of managed VMs), a HTTP POST is sent to the Phantom API. Phantom expects a JSON with data such as the number of VMs to run as well as the VM type and location to run them. A decision engine was made to run on shoal-server that scales by the accumulation of load on all shoal-agents. Scaling was more stable with the accumulation compared to having a sensor on each shoal-agent; the number of VMs did not fluctuate as there was a single measurement used. The decision engine is not included in the distribution of Shoal as shoal-agents will be running in clouds not supported by Phantom. These tests show that Phantom can be used to dynamically scale VMs for Shoal. Unfortunately, Phantom can not be used by Shoal in production currently, as arbitrary clouds are not supported.

3.3 x509 Certificates

The shoal-server needs to be able to verify the identity of any agents advertising to it. Verifying shoal-agents prevents malicious agents from being returned from the shoal-server. Software distributed with CVMFS is cryptographically signed; data is verified for correctness by clients. This prevents code injection vulnerabilities, though clients would not be given useful shoal-agents. To solve this problem, x509 certificates were used.

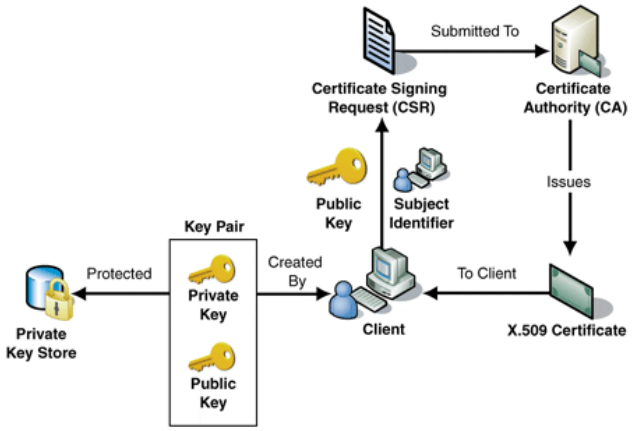

Figure 3: Basic Overview of Certificate Authority Infrastructure[3]

Certificate Authorities (CAs) work through a network of trust. The server has a set of CAs that it trusts and will only accept connections from agents who have a certificate issued by one of those CAs. In the case of Shoal, the server has its own private CA; potential shoal-agent administrators must be sent signed certificates from Shoal administrators. The server accepts a very limited set of shoal-agents. This verification works both ways as well. The server also has a certificate which the shoal-agent can use to verify the identity of the server. OpenSSL was used to generate the CA and certificates used in Shoal. Both RabbitMQ and Pika can have SSL functionality enabled and the steps to do so are shown in appendix A.2.

3.4 Distribution

To deploy Shoal in production, it needs to have few setup/configuration steps. System administrators will not want to do an install from source. Three methods of distribution were considered: turning all needed Python scripts into an executable, a conary RPM repository and a yum RPM repository.

Python has a set of tools for freezing a Python script into an executable. cx_Freeze[4] was used to do the freezing. It parses all imports and turns the script into an executable with a myriad of shared object files. This would be distributed as a compressed archive and has a couple benefits:

1. requires almost no prerequisites (just a method to obtain and extract the archive). The Python install used in development becomes a part of the executable allowing development to be done using newer features of Python without the end user installing any extra dependencies.

2. no install required for basic use. Upon extraction, Shoal can be immediately run (though it requires a basic install script to add daemon support).

The drawbacks of this method are quite large however. The biggest flaw is that adding security patches is nearly impossible; if in the future a security flaw is found in a library used in Shoal, a patch would need to be applied to a custom binary executable. The patch would need to be applied in source, refrozen, then redistributed. RPMs support use of patch fixes. Distributing an executable also is closed source and the number of shared object files is quite large. For these reason freezing was not used and instead RPM repositories were investigated.

One major benefit of RPM packages is module and, language packages are separated from Shoal's packages. Packages can be marked with dependencies which, the user's package manager will automatically resolve. For example shoal-agent requires Python 2.6. A package manager recognizes this and, if the dependency is not met, will install Python 2.6 along with shoal-agent. Unfortunately some of the Python modules used by Shoal do not have packages provided by their developers. While all the Python modules used have a script for generating RPMs (a requirement for a module being in PyPI), it adds extra overhead for maintenance of a Shoal repository. Conary was the first RPM investigated.

Conary is an exotic package manager used by default in CERNVM. It is more feature rich than other package managers and has features found in source control management (SCM) systems. Rollbacks and patches work as deltas do in SCM; when a new version is released only the portion that is modified needs to be downloaded. Conary's development is troubled. The company that made Conary, rPath, was purchased last year and very little of the documentation has been migrated from rPath's domain[24]. Conary is not well supported and setting up a repository for it is complicated.

Yum is the standard package manager in a few Linux flavours (RHEL/Fedora/Centos/Scientific Linux). Yum has a command line tool that handles generation of the repository. Using this tool creates the repository in a single command. Yum also allows for incorporation into Apache web server which, allows for multiple, concurrent, remote connections. Due to this simplicity, Yum was used to distribute Shoal. Shoal has two branches in its repository, production and development. The latest development releases are available to test, along with stable packages.

3.5 Packaging Shoal

After a distribution technology, methods for packaging were investigated. Building Python RPMs is complicated. The standard packaging module distutils is not feature rich and groups within the Python community have created a number of libraries; the largest of these, setuptools, is a *de facto* standard. Many Python packages are distributed in the Python Package Index (PyPI) [16] which, uses setuptools. The Python Packaging Authority also recommends setuptools[25]. For Shoal, distutils is sufficient but setuptools is also supported as setuptools is a super set of distutils. Both libraries use a setup Python script that contains packaging information such as name of package, maintainer, license as well as any configuration/data files needed. The script can be passed various commands to install the Python module or to package it in various forms (RPM, Windows executable, etc.). Using this script is sufficient for generating packages with dependency lists. It is not possible to use shell commands with this method however. Shoal-agent is expected to be run as a service. Setuptools can not generate a RPM which automatically installs shoal-agent as a service. For more detailed step on how to use setuptools see appendix B.

3.6 Puppet Setup for Shoal Agent

Puppet is an automation tool for configuring machines[23]. It runs using ruby with a list of resources and dependencies called manifests; it assures that files are present, any needed packages are installed and services are running. It relies on the system's package manager to get any needed software (see previous sections for packaging Shoal). Puppet allows for automatic installation of both Squid software and shoal-agent.

4 Overhead of SSL

Adding SSL communication reduces response time of Shoal; when an agent attempts to connect to RabbitMQ, the server must verify the agent's certificate. The time delay was measured as the round trip time from a shoal-agent to shoal-server (shoal-server sends an acknowledgment back when it receives a message). To simplify benchmarking of the relative performance of SSL and non-SSL connection methods, blocking AMQP connections were used. This is a non-optimal configuration which limits the value of benchmark to being comparative between the two methods. The shoal-agent was running on FutureGrid at the University of Indiana while the server ran at the University of Victoria. First one message was sent per connection established.

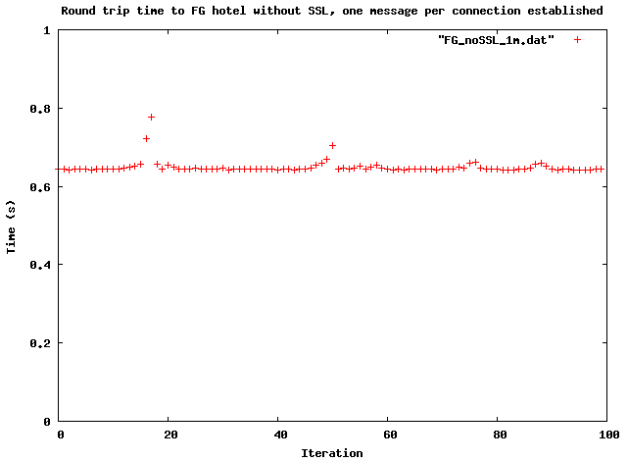

Figure 4: Sending 1 message without SSL

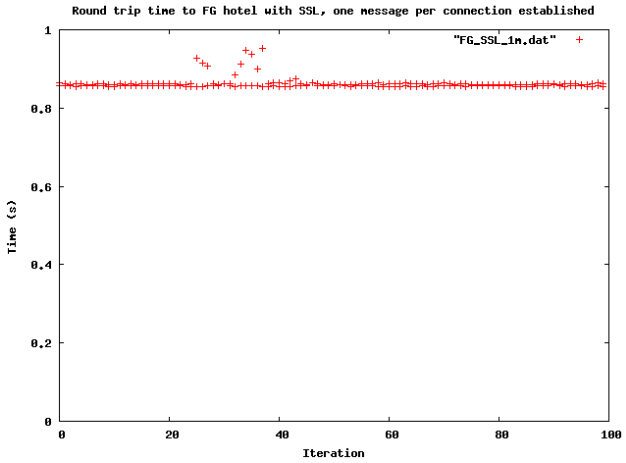

Figure 5: Sending 1 message using SSL

The average round trip time without SSL (Figure 4) was roughly 640ms while using SSL took around 860ms (Figure 5). Using SSL adds a significant delay. To see if this overhead is only incurred once per connection, 100 messages were sent before closing the connection. Pika uses acknowledgments to guarantee message arrival; when the receiver gets a message, it sends a response back to the sender. In this test one message was sent, then the sender waited for the acknowledgment before sending the next message. This was repeated for 100 messages. If SSL overhead is incurred for every message, the relative difference in timing would be the same as one message (using SSL would be around 30% slower).

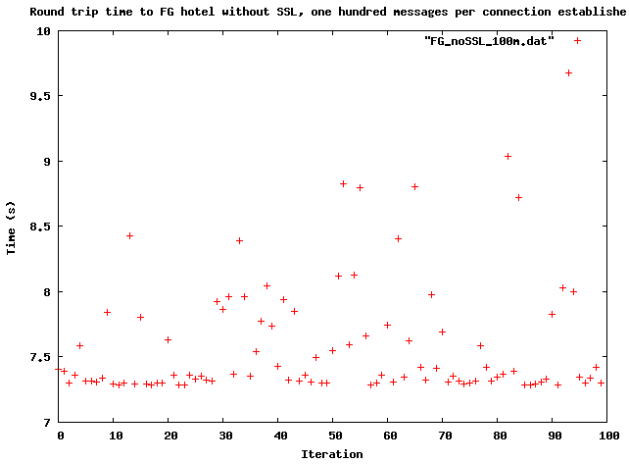

Figure 6: Sending 100 messages without SSL

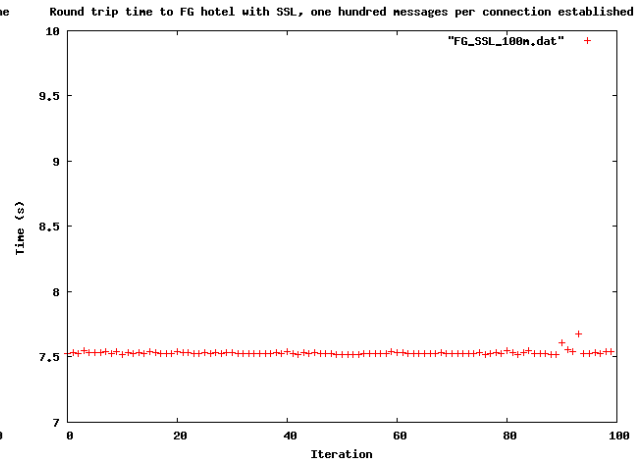

Figure 7: Sending 100 messages using SSL

Here the difference is much smaller; without SSL took around 7.4 seconds (Figure 6) and with SSL taking 7.5 seconds (Figure 7) to send 100 messages sequentially. As the difference is roughly the same between 1

and 100 messages sent, the overhead is only incurred once. For Shoal, it is not practical to keep a connection between shoal-server and shoal-agent open that long. Currently, Shoal is configured to only require a message once per 90 seconds. As the agents send messages infrequently, the added time to process SSL certificates may not matter. Future testing will need to be done to determine the proper balance of server load, and update frequency. If shoal-agents are required to send data to the server more frequently, the server load increases. SSL adds to the load as well so using SSL may necessitate fewer updates.

5 Alternative Solution: Airbnb SmartStack

Airbnb is a company that provides a service to allow homeowners to rent out their homes/apartments nightly. Travelers can find and book a stay at these locations. Airbnb has created an open source tool called SmartStack which, routes requests from clients to available back-ends[26]. Currently, the GitHub project has had three contributors. It provides a similar service to Shoal; in the case of Shoal, back-ends would be shoal-agents. SmartStack is written in ruby and can handle multiple services. Shoal provides just Squid web caches, with SmartStack, multiple service could be provided to clients. Like Shoal, SmartStack is broken into three main components: nerve, Apache Zookeeper, and synapse. These have a rough correlation with Shoal.

Nerve is similar to shoal-agent as it advertises its existence to zookeeper. A shoal-agent's health (whether it is online and functioning correctly) must be done via a heartbeat. If shoal-server stops receiving messages from a given agent, the agent is presumed to have gone offline. Nerve can provide a more robust status check using HTTP. Anything message can be sent to zookeeper. Nerve has to have a list of all zookeeper nodes' IPs.

Zookeeper keeps a list of active machines running nerve and when a client connects, it provides the location of a suitable nerve like shoal-server. Zookeeper runs as a cluster of many distributed nodes on as many machines as necessary. This allows for SmartStack to load balance requests as well as services. If a particular Zookeeper node is overburdened with synapse requests, one can be rerouted to another node. Each instance of synapse/nerve needs a list of every Zookeeper node and need to be reconfigured/restarted on addition of new server. According to Airbnb's site, SmartStack can not properly handle the failure of a single Zookeeper node. If one node fails, the entire SmartStack service could go down. RabbitMQ can also run as a cluster though Shoal has not been configured to make use of this feature.

Synapse requests an available nerve from the server similar to shoal-client. This works by running a proxy locally on synapse. A local port is routed to a port on a nerve node. CVMFS would route to this local proxy and then be routed again to the Squid cache on nerve. This local proxy has to restart when a new Squid cache is added; nodes on the known lists can go on and offline but if a completely new node is added, the local proxies across all instances of synapse get restarted.

SmartStack could accomplish the same thing as Shoal. It provides a more general purpose service at the cost of being harder to setup and being less deterministic. Shoal-client modifies the CVMFS configuration file at set intervals. When run, it modifies the file once. In this way clients can configure how often and when they want to update the cache in use. Synapse will update as soon there is a more suitable node (based on load). This load balancing is difficult to configure or disable and does not account for proximity. Also, adding SSL support is not possible within SmartStack. A wrapper like stunnel could be used in conjunction with SmartStack but that adds another proxy. If a group of clients needs multiple services SmartStack would be much better suited than Shoal. If only squid proxies are required, the extra complexity of SmartStack does not seem worth the drawbacks.

6 Conclusion

There is a need for managing Squids caches and Shoal is ready for production testing. It is easy to install/distribute, verification using x.509 certificates works and simulation tests indicate it can handle relatively large workloads[14]. Once some production data is gathered, Shoal can be optimized to handle heavier loads and distributed further.

A Code Samples and Example Configuration Files

A.1 Custom JSON parser

```
def parseServerData(jsonStr):
    # Nested properties need to be handled separately
    dataTypes = ["load",        "distance", "squid_port", "last_active", "created", \
                 "external_ip", "hostname", "public_ip",  "private_ip"]

    for dataType in dataTypes:
        p = re.compile("\""+dataType+"\": ([^,]+)[,|}]")
        dataList = p.findall(jsonStr)
        for i, val in enumerate(dataList):
            outDict[unicode(str(i))][unicode(dataType)] = convertServerData(val)
```

Figure 8: Portion of the JSON parser in shoal-client

The regex string `([^,]+)[,|}]` stores any non comma characters (the "(" brackets indicate to save that data) until a comma or a brace is found. This data is returned as a list by the Python regex module findall function. The values need to be converted into their proper type from a sting (float, int, etc.). The enumerate method returns the index and value of each element in this list. The convertServerData method will turn strings into any arithmetic types (float, int, etc.) Unicode casts are necessary to provide exact parity with the standard JSON module. The custom parser and standard parser return the exact same output for Shoal. Any future modifications to the custom parser should use the standard parser to verify correctness. Adding a non-nested field (any field that is not a list or a dictionary) requires modifying the list of data types to include it.

A.2 SSL configuration

```
[
  {rabbit, [
     {tcp_listenere, []},
     {ssl_listeners, [5671]},
     {ssl_options, [
                           {cacertfile,"/path/to/ca/cacert.pem"},
                           {certfile,"/path/to/server/cert.pem"},
                           {keyfile,"/path/to/server/key.pem"},
                           {verify,verify_peer},
                           {fail_if_no_peer_cert,true}]}
   ]}
].
```

Figure 9: RabbitMQ SSL configuration file

This configuration file in figure 9 (which, by default, should be located at /etc/rabbitmq/rabbitmq.config) tells RabbitMQ to listen on port 5761 for SSL connections. It verifies that any connections are signed by the "cacertfile" and provides it's own certificate using "certfile" with "keyfile". It verifies every connection and will drop any connections if the certificate provided was not signed or if no certificate was provided. By default, RabbitMQ opens an unsecured TCP listener on port 5672; this port must be explicitly closed by providing an empty list of TCP listeners.

```
connection = pika.BlockingConnection(pika.ConnectionParameters
                                        (
                                         host = config.amqp_server_url,
                                         port = config.amqp_port,
                                         ssl  = config.use_ssl,
                                         ssl_options = sslOptions
                                        ),
                                       )
```

Figure 10: Pika modifications to enable SSL

Figure 10 shows to changes that need to be made to shoal-agent and shoal-server to enable secure communication. To establish a connection without SSL, Pika just needs an IP address and a port to use. Enabling SSL requires a few more parameters. The SSL parameter is a Boolean enabling SSL and ssl_option is a dictionary containing the same CA certificate as the server and its own certificate/key file. After these changes, a handshake will happen first where they each verify the other's provided certificate. Upon success, the connection will be established and messages can be sent.

A.3 Yum Repository

```
[shoal]
name=Shoal Repository
baseurl=http://shoal.heprc.uvic.ca/repo/prod/
enabled=1
gpgcheck=0

[shoal-dev]
name=Shoal Repository
baseurl=http://shoal.heprc.uvic.ca/repo/dev/
enabled=0
gpgcheck=0
```

Figure 11: Yum Repository Configuration

Yum has createrepo, a tool for configuring a repository in one command. Running createrepo on a specified directory creates metadata for any RPM packages under it. The metadata includes file/version listings as well as an Sqlite database. The createrepo command must be run every time a RPM is added or modified (createrepo provides an –update command line option to hasten updates). To use this repository, users must put a .repo Yum configuration file in their /etc/yum.repos.d/ directory. The configuration file for shoal is give in Figure 11. Each repository does not need require more than a location and name. The gpgcheck option specifies that theses files are not signed. It is also possible to configure the repository to make use of an Apache server. Assuming default Apache install location, putting the repository under /var/www/html/ will make the repository accessible remotely without any extra configuration.

A.4 Puppet Manifest for Shoal-agent

```
# Setup shoal agent
# install the shoal repo
file {'shoal.repo':
  path    => '/etc/yum.repos.d/shoal.repo',
  ensure  => present,
  mode    => 0666,
  content =>
'[shoal-dev]
name=Shoal Repository
baseurl=http://shoal.heprc.uvic.ca/repo/dev/
enabled=1
gpgcheck=0',
}
# install the shoal agent package
package { 'shoal-agent':
  ensure => latest,
}
# make sure shoal agent is running and starts on boot
service { 'shoal_agent':
  ensure     => running,
  enable     => true,
  hasstatus  => false,
  hasrestart => true,
}
```

Figure 12: Shoal-agent Puppet Manifest

Running Puppet apply on this manifest will install shoal-agent (not included in this example is Squid configuration as that requires a large configuration file). The manifest is relatively simple, it ensures the Shoal repository is setup by creating the yum configuration, installing shoal-agent and then ensuring it is running. The service hasstatus flag specifies that shoal-agent does not return numerical status flags. It is still possible for Puppet to check that shoal-agent is running by checking its PID file. The rest of the manifest is relatively self explanatory.

B Building RPM Packages for Shoal

This appendix gives detailed information on how to package Shoal.

B.1 Using the Setup Script

Packaging Shoal is done using standard Python setuptools/distutils. If no modifications are needed to the spec file this can be done using sudo Python setup.py bdist_rpm. This command should only be executed if Shoal is not installed. As setup.py is used both for install and RPM building it has a few quirks. It checks if configuration or service scripts are already created before installing or building. For install, this means it will not overwrite a configuration file if one is already present on the system. For building, this means the configuration file will not be included in the RPM, which, is undesirable as a configuration file is necessary to run Shoal. Using setup.py also will check the privileges of the user running the script. If they are root, the configuration file will be installed to /etc/shoal/shoal_BRANCH.config otherwise it will be installed in the home directory of the user who installs the RPM. When setup.py is run, it will create a folder called dist which contains: a gzipped source file, a source RPM, and a binary "noarch" RPM (the netifaces library used by Shoal is the exception to this and will be made as the architecture of the host machine). Setup.py should be passed a dependency list via the command line option –requires. When the user installs the RPM using yum, if they are missing any dependencies they will be prompted to install them.

B.2 Custom spec file

A spec file is basically a configuration file for RPMs it includes the list of files to use, how to install/build the program and any scripts that should be before or after install. Using a spec file is necessary if a more complicated install process is required. A tool like rpmbuild is needed to package source into an RPM. The setup.py script is still quite useful for making an RPM manually. Running sudo Python setup.py bdistm_rpm and then sudo Python setup.py bdistm_rpm –spec-only, will create a useable gzipped source file as well as a base spec file. The generated source file needs to be moved to /rpmbuild/SOURCES (if running as sudo replace with /root). Then after modifying the spec file, run rpmbuild -ba spec.file. This will use the provided source and spec file to generate a source and binary RPM. These can be found under /rpmbuild/SRPMS and /rpmbuild/RPMS respectively.

The following is an example of shoal-agent's spec file.

```
%define name shoal-agent
%define version 0.7.1
%define unmangled_version 0.7.1
%define release 1

Summary: A squid cache publishing and advertising tool designed to work in fast changing environments
Name: %{name}
Version: %{version}
Release: %{release}
Source0: %{name}-%{unmangled_version}.tar.gz
License: 'GPL3' or 'Apache 2'
Group: Development/Libraries
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-buildroot
Prefix: %{_prefix}
BuildArch: noarch
Vendor: Mike Chester <mchester@uvic.ca>
Requires: netifaces >= 0.8 pika >= 0.9.13
Url: http://github.com/hep-gc/shoal

%description
UNKNOWN

%prep
%setup -n %{name}-%{unmangled_version} -n %{name}-%{unmangled_version}
```

```
%build
Python setup.py build

%install
Python setup.py install --single-version-externally-managed -O1 --root=$RPM_BUILD_ROOT --record=INSTALLE

%post
touch /var/log/shoal_agent.log
chown nobody:nobody /var/log/shoal_agent.log
chmod 0644 /etc/shoal/shoal_agent.conf
chkconfig --add shoal-agent


%clean
rm -rf $RPM_BUILD_ROOT

%files -f INSTALLED_FILES
%defattr(-,root,root)
```

Setup.py generated a large majority of this file; the requires string was put in manually as was the post section. Setuptools is used to build and install the package as seen in the build and install sections. In the post section, shoal-agent will automatically create an empty log file, change the owner to nobody (the user who runs agent as a service), set permissions and install itself as a service when the RPM is installed.

To verify the RPM package built correctly, use the commands rpm -qpl and rpm -qpR . These commands list the files and requirements of the package respectively. When run on a shoal-agent package built for Python 2.6, the following output is produced:

```
$ rpm -qpR shoal-agent-0.7.1-1.noarch.rpm
/bin/bash
/bin/sh
/usr/bin/Python
netifaces >= 0.8
pika >= 0.9.13
Python(abi) = 2.6
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
$ rpm -qpl shoal-agent-0.7.1-1.noarch.rpm
/etc/init.d/shoal-agent
/etc/shoal/shoal_agent.conf
/usr/bin/shoal-agent
/usr/lib/Python2.6/site-packages/shoal_agent-0.7.1-py2.6.egg-info
```

The full list of files is omitted here as shoal-agent has many files for installing itself as a Python module. The files more likely to be changed are the configure file and daemon script. These files must not be present on system packaging Shoal and the rpmbuild command must be run as root for the configuration file to be placed in /etc/shoal (instead of /.shoal).

7 Glossary

AMQP Advanced Message Queuing Protocol

API Application Program Interface. Interface to access a library or service's features.

CERN Conseil Europen pour la Recherche Nuclaire (European Council for Nuclear Research).

HTTP Hypertext Transfer Protocol. Basic protocol for transferring data on the internet.

JSON JavaScript Object Notation. A file format consisting of name/value pairs.

LHC Large Hadron Collider.

OS Operating System.

Phantom is an auto-scaling tool made in conjunction with the Nimbus project

regex Regular Expression. A pattern matching language.

RPM Resource package manager. An easy way to install software on UNIX systems.

SSL Secure Socket Layer. Internet protocol layer that adds encryption.

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

References

[1] Computing, [Online]
Available: http://home.web.cern.ch/about/computing [December 12, 2013]

[2] ATLAS Fact Sheet
Available: http://www.atlas.ch/pdf/atlas_factsheet_4.pdf [January 9, 2013]

[3] X.509 Technical Supplement, [Online]
Available: http://msdn.microsoft.com/en-us/library/ff647097.aspx [December 12, 2013]

[4] cx_Freeze, [Online]
Available: http://cx-freeze.sourceforge.net/ [December 12, 2013]

[5] The Tier Sites, [Online]
Available: http://lcg-archive.web.cern.ch/lcg-archive/public/tiers.htm [December 12, 2013]

[6] CernVM File System, [Online]
Available: http://cernvm.cern.ch/portal/filesystem [December 12, 2013]

[7] A squid cache publishing and advertising tool designed to work in fast changing environments, [Online]
Available: https://github.com/hep-gc/shoal [December 12 ,2013]

[8] Introduction to Pika, [Online]
Available: http://pika.readthedocs.org/en/0.9.13/ [December 12, 2013]

[9] The Apache Software Foundation, [Online]
Available: http://www.apache.org/ [December 12, 2013]

[10] Netifaces – Portable Access to Network Interfaces from Python, [Online]
Available: http://alastairs-place.net/projects/netifaces/ [December 12, 2013]

[11] pygeoip, [Online]
Available: https://pypi.Python.org/pypi/pygeoip/ [December 12, 2013]

[12] Welcome to web.py, [Online]
Available: http://webpy.org/ [December 12, 2013]

[13] simplejson - JSON Encoder and Decoder, [Online]
Available: http://simplejson.readthedocs.org/en/latest/ [December 12, 2013]

[14] Shoal: IaaS Cloud Cache Publisher
Mike Chester, April 24, 2013

[15] Python 2.4, [Online]
Available: http://www.python.org/download/releases/2.4/ [December 12, 2013]

[16] PyPI - the Python Package Index, [Online].
Available: https://pypi.python.org [December 12, 2013]

[17] RabbitMQ - Messaging that just works, [Online].
Available: http://www.rabbitmq.com [December 12, 2013]

[18] ATLAS experiment at the CERN Laboratory in Geneva.
Available: http://www.cern.ch/ [December 12, 2013]

[19] Squid: Optimizing Web Delivery, [Online].
Available: http://www.squid-cache.org [December 12, 2013]

[20] Nimbus Phantom, [Online]
Available: http://www.nimbusproject.org/doc/phantom/latest/ [December 12, 2013]

[21] Nimbus, [Online]
Available: http://www.nimbusproject.org/ [January 8, 2014]

[22] FutureGrid an XSEDE resource provider, [Online]
Available: https://portal.futuregrid.org/ [December 12, 2013]

[23] Puppet Open Source, [Online]
Available: http://puppetlabs.com/puppet/puppet-open-source [December 12 ,2013]

[24] SAS Acquires Key rPath Assets for Broader Deployment of SAS Solutions, [Online]
Available: http://www.sas.com/news/preleases/rpath-asset-acquisition.html [December 12, 2013]

[25] The Future of Python Packaging, [Online]
Available: https://python-packaging-user-guide.readthedocs.org/en/latest/future.html [December 12, 2013]

[26] SmartStack: Service Discovery in the Cloud, [Online]
Available: http://nerds.airbnb.com/smartstack-service-discovery-cloud/ [December 12, 2013]