Dynamic web cache publishing for IaaS clouds using Shoal

Frank Berghaus, Michael Chester, Ian Gable, Colin Leavett-Brown, Michael Paterson, Robert Prior, Randall Sobie, Ryan Taylor

University of Victoria

https://github.com/hep-gc/shoal

Abstract

It has been demonstrated that HEP workloads can run effectively on remote IaaS cloud resources. Typically each running Virtual Machine (VM) makes use of the CERN VM Filesystem (CVMFS), a caching HTTP file system, to minimize the size of the VM images, and to simplify software installation. Each VM must have access to a nearby HTTP web cache, such as Squid, in order to function efficiently. Traditional grid sites also use CVMFS to provide worker node software, and deploy one or more Squid servers at their site. However, each IaaS cloud site has none of the static infrastructure associated with a typical HEP grid site. Each cloud VM must be configured to use a particular Squid server. We have developed a method and software application called Shoal for publishing Squid caches which are dynamically created on IaaS clouds. The Shoal server provides a simple REST interface which allows clients to determine their closest Squid cache. Squid servers advertise their existence by running Shoal Agent which uses the Advanceded Message Queuing Protocol (AMQP)to publish information about the Squid server to the Shoal server. Having a method for exchanging the information rapidly allows for Squid servers to be instantiated in the cloud in response to load and for clients to quickly learn of their existence. In this work we describe the design of Shoal and evaluate its performance.

Design

Shoal was developed with scalability as the central concern. In order to be viable at the scale of current HEP distributed computing systems, Shoal needs to be able to handle many thousands of worker nodes requesting locations of Squid caches regularly, in addition to receiving continuous updates from many Squid caches. AMQP was selected for the method of communication between Shoal Agent and Shoal Server because of its robustness and the possibility of using message queues to scale system components horizontally. The Shoal Client is designed to be a trivial Python script with no dependencies, to make installation simple.

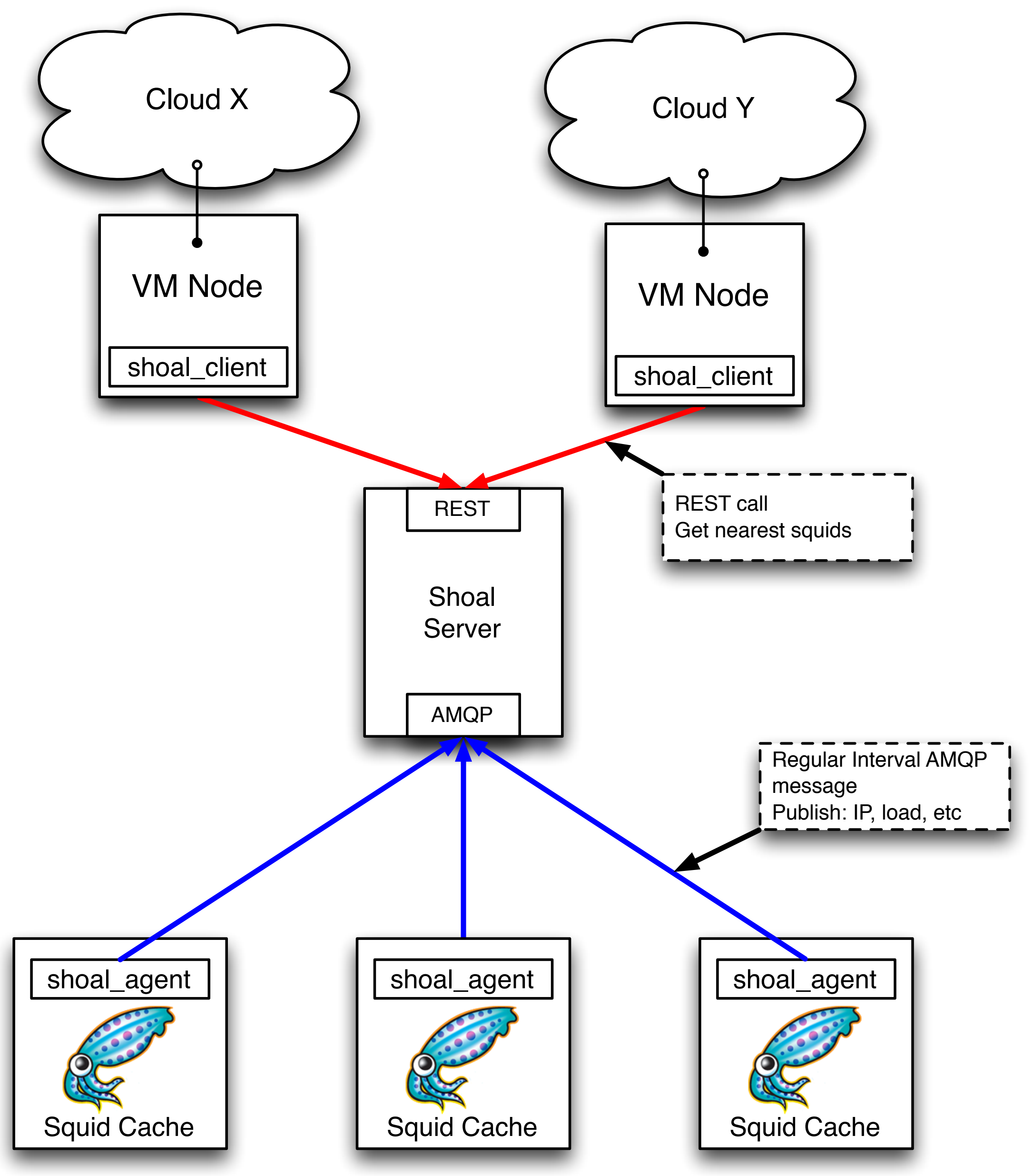

Figure 1. The interaction between Shoal components. Shoal Agents send regular interval AMQP messages to Shoal Server. Shoal Clients residing on IaaS clouds query the REST interface on Shoal Server to retrieve a list of the closest Squid servers.

Shoal Client - is used by worker nodes to query Shoal Server to retrieve a list of geographically nearest Squid servers and configures the CVMFS HTTP proxy server. Shoal Client is designed to be simple (less than 100 lines of Python) with no dependencies beyond the standard operating system default Python install.

Shoal Agent -is 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 wishing to publish its existence runs Shoal Agent on boot. Shoal Agent sends periodic messages heartbeat messages to the Shoal Server (typically every 30 seconds).

Shoal Server - maintains a list of active squid servers in memory and prunes instances from which it has not received heartbeat messages via Shoal Agent. It also provides Shoal Clients a RESTful interface to retrieve a list of the nearest Squid servers. Shoal sever uses the geoip library and the IP address of the incoming http request to the nearest squids in order. A web user interface is provided to easily view Squid servers being tracked.

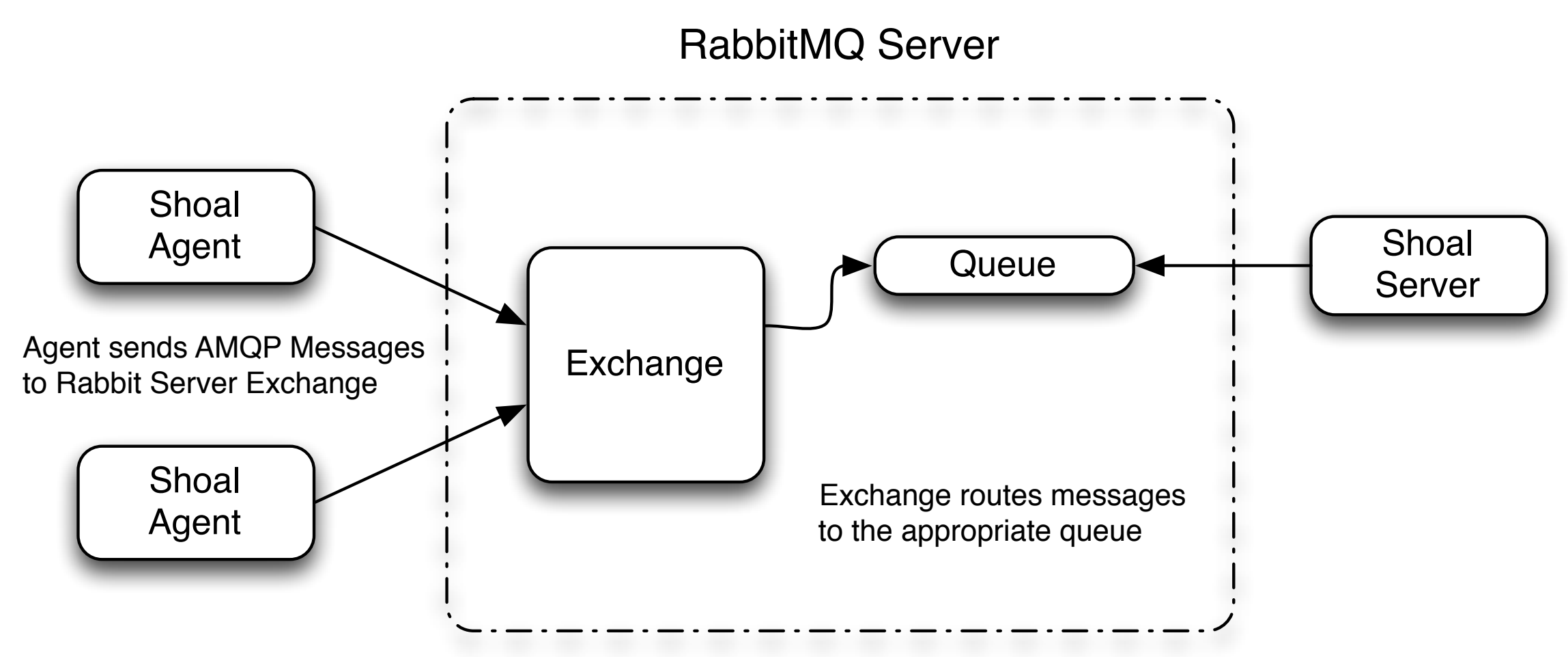
RabbitMQ Server
Shoal Agent
Shoal Agent
Agent sends AMQP Messages to Rabbit Server Exchange
Exchange
Queue
Shoal Server
Exchange routes messages to the appropriate queue

Figure 2. Advanced Message Query Protocol (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.

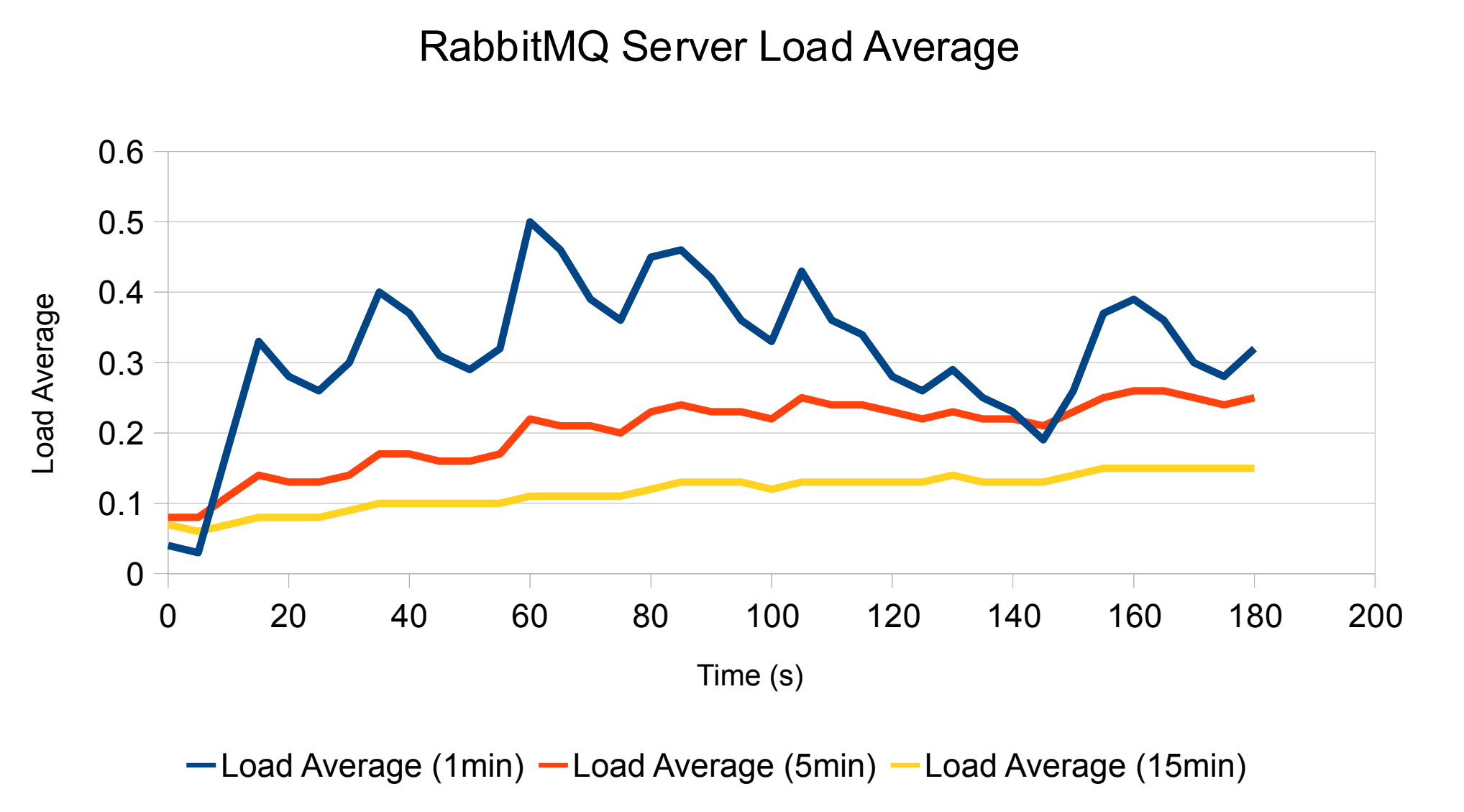

Figure 3. The load average generated on a single core RabbitMQ server machine with 1 GB of memory by 10 000 Shoal Agent AMQP messages per minute. This would be the equivalent load of 5000 squid caches publishing their existence every 30 s.

Results

In order to ensure that Shoal will be able to function at the scale required, a number of benchmark tests were performed. The Shoal Server was set up on a 16 core x86_64 machine with 64 GB of RAM. A RabbitMQ AMQP server was established on a machine with 1 core and 1 GB of RAM. A client machine with the Apache benchmarking tool 'ab' was set up to access the Shoal Server with a configurable number of parallel accesses, thus simulating the load from many worker nodes. Figures 4 and 5 describe the responsiveness of the Shoal REST interface under load using the Apache web server with the mod_wsgi Python module. We see that a single Shoal Server which is tracking 800 squid servers is able to respond to 1000 requests per second. This is equivalent to handling over 1 million worker nodes requesting their nearest Squid Proxy every 30 minutes, demonstrating that Shoal has sufficient scalability for massive deployments.

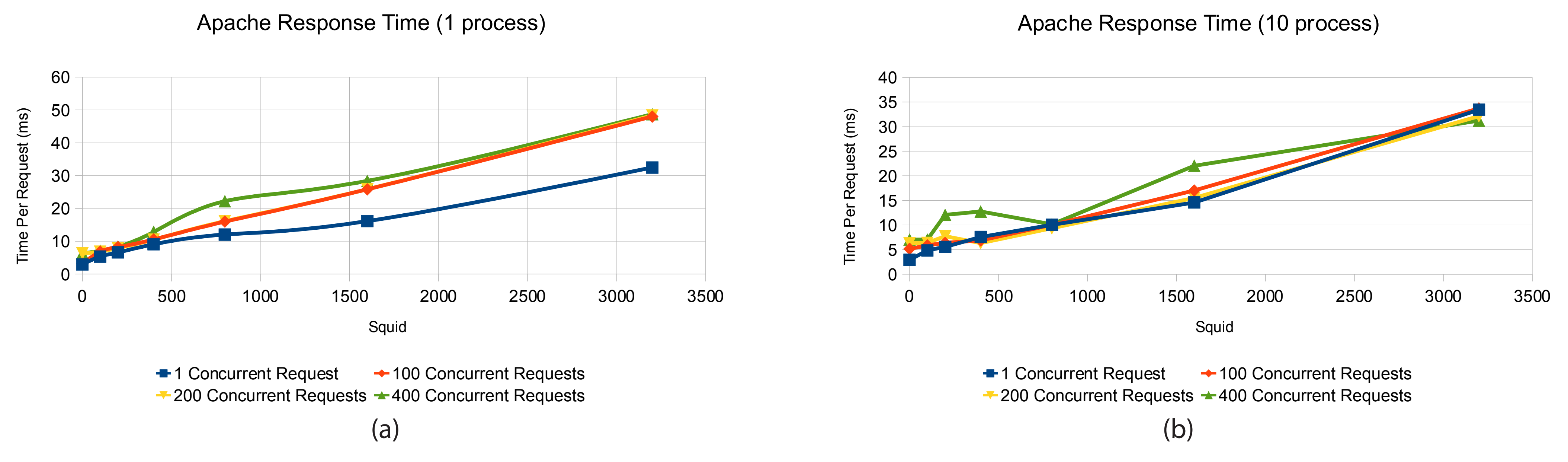

Figure 4. (a) The response time of the Shoal Server shown as a function of the number of Squid caches which it has knowledge. We see that the response time increases roughly linearly with the number of Squid caches with an approximate slope of of 10 ms per 1000 Squid caches. (b) Shows that as expected the response time is not significantly changed by increasing the number of web server processes as expected (mod_wsgi).

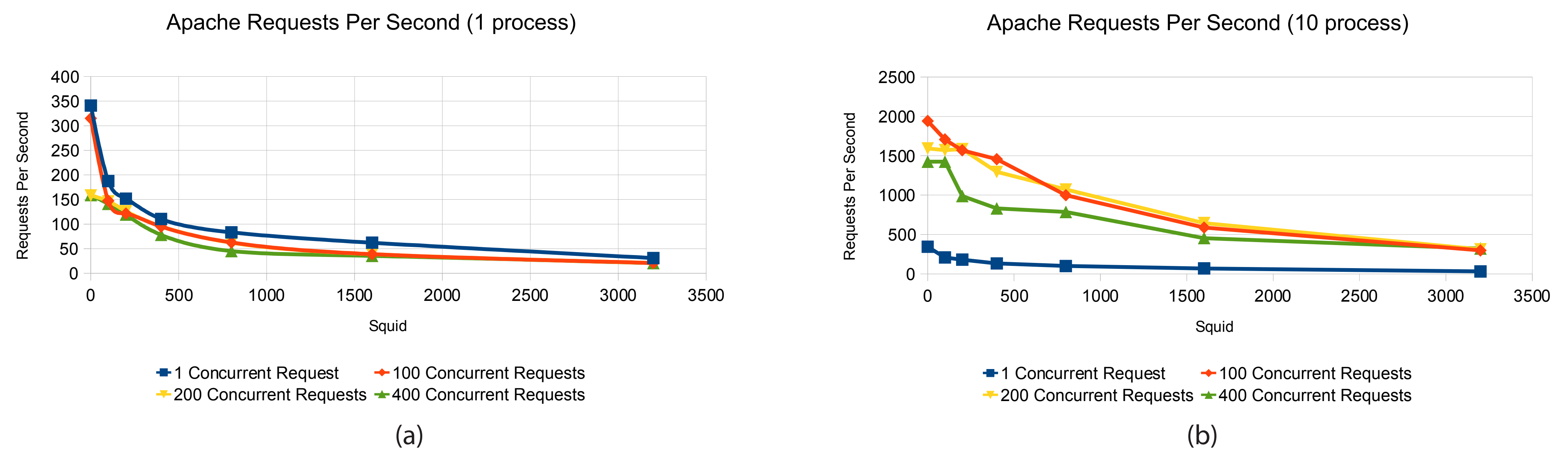

Figure 5. (a) The number of request served per second served by the Shoal Server as a function of the number of Squid Caches which it has knowledge. (b) Shows that the total throughput of requests can be increased by increasing the number of web server processes.

University of Victoria

INSTITUTE OF PARTICLE PHYSICS