MASTH Proxy: An Extensible Platform for Web Overload Control

Vipul Mathur* vipul@cse.iitb.ac.in
Sanket Dhopeshwarkar sanket@cse.iitb.ac.in
Varsha Apte† varsha@cse.iitb.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology – Bombay
Powai, Mumbai, Maharashtra 400076, India

ABSTRACT

Many overload control mechanisms for Web based applications aim to prevent overload by setting limits on factors such as admitted load, number of server threads, buffer size. For this they need online measurements of metrics such as response time, throughput, and resource utilization. This requires instrumentation of the server by modifying server code, which may not be feasible or desirable. An alternate approach is to use a *proxy* between the clients and servers.

We have developed a proxy-based overload control platform called *MASTH Proxy*–Multi-class Admission-controlled Self-Tuning HTTP Proxy. It records detailed measurements, supports multiple request classes, manages queues of HTTP requests, provides tunable parameters and enables easy implementation of dynamic overload control. This gives designers of overload control schemes a platform where they can concentrate on developing the core control logic, without the need to modify upstream server code.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Reusable Libraries*

General Terms

Experimentation, Measurement, Performance

Keywords

Admission control, overload, Web server, proxy, multi-class

1. INTRODUCTION

When the load offered to a system exceeds its maximum processing capacity, the system is said to be overloaded. Typical symptoms of an uncontrolled overload situation are increased delays, thrashing, saturated resources and backlogged queues. There have been many proposed approaches to overload control of Web applications [1–4]. Most mechanisms use a combination of online measurements of performance metrics, overload detection mechanisms, and tuning of controllable parameters to protect the system from adverse effects of overload. Some solutions additionally call for architectural changes, such as adding 'work queues', to the Web server.

This gives rise to the question of where such overload control mechanisms are implemented. Modifying the Web server code requires having access to the code, and the resulting implementation is specific to the particular Web server. An alternative approach is to use a *proxy server* introduced in to the request flow path between clients and servers [4]. The proxy can act as an *admission controller*, preventing the upstream servers from going into an overloaded state by managing the load sent upstream. A proxy based approach is easily portable to any Web servers.

Existing Free and Open Source Software (FOSS) HTTP proxy projects like Muffin (`http://muffin.doit.org/`) and Squid (`http://squid-cache.org/`) have shortcomings such as lack of multi-class support or work queue architecture. Also, most are focused on acting as gateways for traffic outbound from a private network while our usage is akin to the *reverse proxy* scenario for inbound traffic. Given our need for measurement infrastructure and architectural elements discussed below, we chose to develop an extensible proxy-based platform, for performance management of Web applications, from the ground up.

2. FEATURES AND ARCHITECTURE

The main design goal of our proxy-based solution (called *MASTH*–Multi-class Admission-controlled Self-Tuning HTTP–Proxy) presented here, is to provide an easily extensible platform for development and deployment of Web application performance management mechanisms. Using this platform, one should be able to quickly prototype any control method by concentrating on the core logic, rather than on implementing measurement instrumentation or tunable parameters.

The proxy provides the following features:

1. *Request Classification and Work Queues:* The proxy incorporates classification of incoming requests into a configurable number of classes according to any HTTP header field. A separate queue of received HTTP requests is maintained for each class.

2. *Measurement Framework*: The proxy implements a Round Robin Database (RRD4J 2.0.5, `http://rrd4j.dev.java.net/`) that maintains histories of measurements at various granularities in an efficient manner. We can query the database for any measured metric in any specified measure-

*This research was supported by the Philips Fellowship.
†This research was supported by the 2007 IBM Faculty Award.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

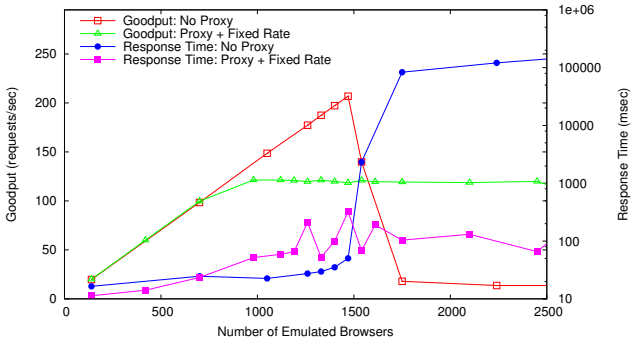

Figure 1: Results from TPC-W Benchmark

ment interval at various resolutions. e.g. 5 second averages for past 10 minutes, 2 minute averages for past hour, etc. Measurements include arrival rate, waiting time, response time, goodput (usable throughput), client timeouts, queue length, and resource utilizations. The set of measurements is stored on a per-class as well as aggregate basis.

3. *Online-tunable control parameters:* Controls such as rate of admitted requests, buffer size, rate of upstream request dispatch, maximum number of pending upstream requests, queue priorities, queuing disciplines (FIFO, LIFO), and multi-queue scheduling policies (round robin, priority) are provided for use by overload control mechanisms. Admission control in the proxy is implemented by placing token-bucket regulators at the ingress and egress points of each queue. This allows flexible traffic-shaping of the incoming load on a per-class basis.

4. *Extendability:* Internally, the proxy is designed with a modular multi-stage event-driven architecture with separate specialized threads/ thread-pools for request parsing, classification, queue management, upstream dispatch, recording and querying of measurements, parameter tuning etc. This makes the design flexible, and it is quite straight forward to extend the functionality of the proxy on two fronts. First to implement new features, such as adding an aggregate rate control over and above the per-class token-bucket regulators, adding new queue scheduling policies, etc. without doing major code restructuring. Secondly, implementing new control loops/ feedback methods becomes straight forward. This can be done in two ways. First is to extend the `TuningThread` base class and implement the core logic of the method being developed. Interfaces to measurements, tunable parameters are directly available. The other way is to use the proxy's Web API to remotely control the tunable parameters from an external implementation of the control method. In this case the proxy acts as a measurement and management subsystem.

3. EXAMPLE USE OF MASTH PROXY

To test the efficacy of our proxy, we carry out load-testing experiments on a test-bed of one server (Dual Intel Xeon, 8GB), and three clients (Intel P4, 1GB). We use a Java+Tomcat+MySQL implementation of the TPC-W benchmark (`http://www.ece.wisc.edu/~pharm/tpcw.shtml`. Note that we have modified the TPC-W load generator to add request timeouts and think time between successive retries of a blocked request. We varied the load from 140-2500 *Emulated Browsers (EB)*.

WebAPI call: `HEAD http://proxy:port/API/ratelimit/35/1`

(a) Fixed rate control: Class1 set to 35 req/s

```
class ThrottleHomeOnOverload extends TuningThread {
void doTuning() {
  if (Measurement.isOverloaded()) {
   AppQueues.setTokenBucketRate(TPCW_Home, 0.6 *
     Measurement.getGoodputMax(OneDay,TPCW_Home));
    AppQueues.enableIngressThrottling();
  } else
    AppQueues.disableIngressThrottling();
}}
```

(b) Limit to 60% of peak goodput only if overloaded

Figure 2: Rate control of `TPCW_Home` (Class1)

In Fig. 1, the graph of aggregate goodput shows that the peak without proxy is at 200 req/s. However, as the number of EBs increases, the goodput drops to less than 20 req/s.

We can use MASTH proxy with a request admission rate control mechanism to manage such an overload. For instance, setting a fixed rate limit of 35 req/s on the `TPCW_Home` class using the WebAPI call in Fig. 2(a) prevents the degradation of goodput after overload—the goodput settles at 115 req/s even as load increases to twice the capacity. The response time charts in Fig. 1 show that the fixed rate limit successfully keeps response times low.

Fig. 2(b) shows how this static scheme may be extended, with a few lines of code, to apply a rate limit only if an overload is detected, and to set it based on past measurements.

4. SUMMARY

MASTH Proxy is an easily extensible platform for performance management of Web applications with features such as a detailed performance monitoring, multi-class support, work queues and pre-built control points for rate control and online tuning of system parameters. The proxy has been coded in Java 6 for portability, and supports HTTP 1.1 with persistent connections. The code has been released as a FOSS project (at `http://sourceforge.net/projects/masthproxy`), and can be used by anybody to test their overload control mechanisms.

5. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, January 2002.

[2] H. Chen and P. Mohapatra. Overload control in QoS-aware Web servers. *Computer Networks*, 42(1):119–133, 2003.

[3] L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51(6):669–685, June 2002.

[4] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in E-commerce Web sites. In *Proceedings of the 13th international conference on World Wide Web*, pages 276–286. ACM Press, 2004.