

# Improving the performance of static and dynamic requests at a busy web server\*

Bianca Schroeder, bianca@cs.cmu.edu  
Carnegie Mellon University

## 1 Introduction

Running a high-volume web site is a challenging task. Web traffic is bursty with peak request rates rising orders of magnitude above average rates and exhibiting phenomena like flash crowds and hot spots. Yet web users are very demanding: they expect Web sites to be quickly accessible from around the world 24 hours a day, 7 days a week. Even a short period of slowdown or service interruption can have severe effects: it not only sends customers to the "just a click-away" competitor; it also compromises the users' perception of the web site in general, as well as the corporate image as a whole.

The broad goal of my thesis work is to improve the user-perceived performance of both static and dynamic requests at busy web sites. By static requests we mean requests of the form "GET me a file". By dynamic requests we mean those which require the web server to create the requested information on the fly, e.g. by running a script or accessing a database. The approach taken in this thesis does not require buying more hardware or any other costly system upgrades. The main idea is to schedule the existing resources better among requests so as to improve overall performance.

Section 2 of this work briefly summarizes the part of my thesis work dealing with static requests. We first describe the work that presents and experimentally validates the main idea of improving web performance by intelligent scheduling of resources. We then extend this idea to improving the performance of web sites during transient periods of overload. Section 3 details my work on QoS support for dynamic requests. Throughout this work I concentrate on transactional web requests, i.e. dynamic requests that involve accesses to a database backend.

## 2 Scheduling static requests

### 2.1 Size-based Scheduling to Improve Web Performance

This work is based on the observation that current Web servers simply time-share among all incoming requests, although scheduling theory suggests that scheduling jobs in

---

\* Keywords: WWW performance, scheduling, static web requests, database driven web workloads.

the order of Shortest-Remaining-Processing-Time (SRPT) is optimal. The main reason why SRPT is not used in practice is a fear of starving big jobs.

We implement a web server that gives preference to short requests in an SRPT-like fashion. The implementation is at the kernel level and involves controlling the order in which socket buffers are drained into the network. Experiments are executed both in a LAN and a WAN environment. We use the Linux operating system and the Apache and Flash web servers.

Our results indicate that SRPT-based scheduling of connections yields significant reductions server delays. These result in a substantial reduction in mean response time, mean slowdown, and variance in response time for both the LAN and WAN environments. Significantly, the large jobs are only negligibly penalized or not at all penalized as a result of SRPT-based scheduling. We give intuition and theoretical justification for this result based on the statistical properties of web traffic.

A short version of this work has been published in the 7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2001)[1]. The full paper has appeared in ACM Transactions on Computer Systems in May 2003[2].

### 2.2 Web servers under overload

Due to the burstiness of web traffic and the frequent occurrence of hot spots in the web every popular web server experiences transient periods of overload. An overloaded web server typically displays signs of its affliction within a few seconds. Work enters the web server at a greater rate than the web server can complete it, causing the number of connections at the server to build up. This implies large delays for clients accessing the server and might finally lead to clients being rejected by the server.

The contribution of this part of my thesis is two-fold. First I provide a systematic performance study of exactly what happens when a web server is run under transient overload. I consider the effect of many parameters on the overload performance of a web server, including WAN conditions like network losses and delays, and user behavior. Second, this work implements and evaluates a particular kernel-level solution for improving the performance of web servers un-

der overload. The solution is based on SRPT connection scheduling. We show that SRPT-based scheduling improves overload performance across a variety of client and server-oriented metrics.

This work has won the student paper award at the 18th International Teletraffic Congress (ITC 2003)[5].

### 3 Scheduling transactional web requests

A common scenario that I concentrate on in my work on transactional web requests is that of an online retail web site where a database backend is the bottleneck in serving dynamic web requests. Within this scenario the goal is to provide different classes of service. These classes can for example be used by the e-commerce retailer to differentiate between important customers (e.g. big spenders) and less important customers (small spenders), reducing response times for big spenders. The retailer may also want to aim for different response time targets for different transaction types, based on recent studies showing that the patience level of customers varies depending on the transaction type.

While there exists tools for prioritizing transactions in commercial database systems, e.g. the Query Patroller for IBM DB2, we found that those don't always work satisfactorily for the kind of systems we are interested in. Moreover, they usually only offer simple prioritization, rather than more complex QoS goals like response time targets.

In my work I investigate two different approaches to the problem of providing different classes of service for database requests: integrating priority mechanisms into the database server (Section 3.1); providing QoS targets in form of an external front-end scheduler (Section 3.2).

#### 3.1 Implementing priorities inside databases

In order to successfully integrate support for priorities into a database server I first perform a detailed bottleneck analysis of resource usage, based on two different transactional workloads (TPC-C and TPC-W) and commercial and non-commercial DBMS (IBM DB2, PostgreSQL, Shore) under a wide range of configurations. I then implement and evaluate the performance of several preemptive and non-preemptive DBMS prioritization algorithms in PostgreSQL and Shore. The primary contributions of this work include understanding the bottleneck resources in transactional DBMS workloads and a demonstration that prioritization in traditional DBMS can provide 3x improvement for high-priority transactions using simple scheduling policies, without penalty to low-priority transactions.

This work has been accepted for publication at the 20th International Conference on Data Engineering (ICDE 2004)[4]. In some of our very recent work [3] we develop a new, effective preemptive priority algorithm based on our findings in [4] and new statistical analysis.

#### 3.2 QoS for databases

While there has been a considerable amount of prior work on integrating support for priorities into the database server, there is hardly any work on providing QoS for databases in the form of an external scheduler front-end. The reasons are two-fold. First, scheduling inside the back-end server is often considered more effective, since it gives the scheduler direct control over all the requests during their entire lifetime. Second there is a big fear that external scheduling will reduce the throughput of the system and/or increase mean response times, since it requires holding transactions back outside the server, which can potentially cause underutilization of resources.

I show that external scheduling of transactional web requests can be effective without the above drawbacks usually associated with external scheduling. My work provides an implementation of a front end scheduler providing multi-class QoS support for three different goals: (i) Guaranteeing mean response time targets for different classes of transactions; (ii) Percentile guarantees, where at most  $x\%$  of response times are above  $y$ ; (iii) Lowering overall variability in response times across all classes. The algorithms involved are based on queueing-theoretic arguments. Implementing the QoS mechanisms externally, not involving database internals, provides for a portable and easy to implement solution. The system is both self-configuring to a particular set of QoS goals, and also is self-adapting to changes in the workload, using simple guidelines we derive from queueing theoretic analysis. We demonstrate the effectiveness of our solution in experiments with two different database servers, IBM DB2 and PostgreSQL, and two different workloads, under multiple configurations.

This work is in submission for publication [6] and is also the basis for a patent currently being filed by IBM.

#### References

- [1] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal. Implementation of SRPT scheduling in web servers. In *Job Scheduling Strategies for Parallel Processing, 7th International Workshop, JSSPP*, 2001.
- [2] M. Harchol-Balter, B. Schroeder, M. Agrawal, and N. Bansal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 2003.
- [3] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. The case for preemptive lock scheduling in oltp workloads. in *submission.*, 2004.
- [4] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *20th IEEE Conference on Data Engineering (ICDE'2004)*, 2004.
- [5] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. In *International Teletraffic Congress (ITC-15)*, 2003.
- [6] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. Providing QoS from outside the box. in *submission.*, 2004.