Towards Predictable Overload Control of Large Real-Time Server Systems

Thiemo Voigt

September 28, 2000

1 Introduction

Real-time systems such as Web servers are the heart of the Internet. Due to the enormous growth of the Internet, server systems have to cope with a huge number of users and requests. To satisfy that demand, server systems are organized as large symmetric multiprocessor systems (SMPs) or as a cluster of servers.

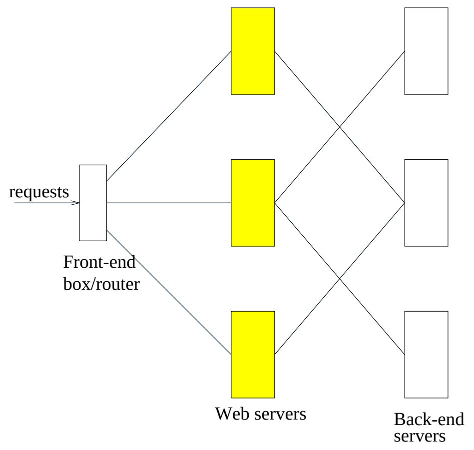

Figure 1: Typical web site configuration

As shown in Figure 1, a cluster consists of a front-end (often called dispatcher) that forwards requests to the Web servers. The back-end servers are used for e.g. database queries. With the emerge of new innovative and more demanding services such as e-commerce, Web severs not only have to provide fast response times, but also *predictable* service. In particular, under overload it is of major importance that at least preferred customers are provided with continuous service and a certain level of Quality of Service (QoS). It is also desirable that customers can conclude transactions leading to an electronic purchase.

2 Project Plan

The current Internet architecture is moving towards application service providers and other Web hosting services that co-host multiple customer sites on the same server cluster or large SMP. In such environments a flash crowd can overload multiple sites simultaneously and affect the performance of a large number of users. An overloaded server may reach a thrashing state where little or no useful work is done. This results in the clients observing higher response times, connection timeouts, failures and an eventual loss of service.

In the first phase of our project, we extend previous work [3] [4], that has focused on kernel mechanisms that protect web servers against overload. These mechanisms can be applied to both traditional servers, large SMPs or on the servers in a server cluster. The experiments we conducted in the course of this research have revealed a need for recovery mechanisms that allow a server to recover fast from overload, once it has already set in. Fast recovery is necessary even with overload prevention, in case the onset of overload could not be predicted with sufficient accuracy. Overload prevention attempts to provide *uninterrupted* service and performance to clients, overload recovery tries to *restore* server performance as soon as possible after overload.

There are several actions that can be taken when overload is detected:

1. Discard all new requests.
2. Reduce the rate of new requests.
3. Discard requests in the system, i.e. flush the listen queue.

In order to provide service differentiation one can drop low priority requests while high priority requests are rate controlled to provide service continuity during recovery. The first two actions lead to quite high recovery times. The third mechanism decreases recovery time but at the cost of discarding already accepted work. Furthermore, customers will be annoyed when their connections are reset by the server.

To allow fast overload recovery without discarding accepted connections we propose a new mechanism that under overload blocks server processes when they call "select" or "accept" to get the next piece of work after having serviced a request. By decreasing the number of active processes this mechanism will lower contention for memory and allow for fast overload recovery.

Our short-term goal is to implement these mechanisms and perform experiments to validate their efficiency.

Note that the blocking mechanism will only be efficient for HTTP 1.0. For HTTP 1.1 which uses persistent connections, we want to base the decision on connection reset on the requested object, i.e. the requested URL. For performance reasons this should be done in the kernel of the operating system. To enable control of persistent connections is important since resetting the "wrong" connection might for example result in the abortion of a session that leads to an electronic purchase. On the other hand it is impossible to keep all persistent connections alive under overload and recover fast.

Currently, no operating system provides kernel-based URL parsing for all requests on persistent connections. Support for this is announced for Linux 2.5.

Using this support, we will extend the work above to incorporate persistent connections. It is an open research question if this mechanism gives substantial benefit, i.e. it is not obvious if the server system can afford parsing requests on persistent connections under overload.

In the next phase of the project we will collaborate with Lars Albertsson using the tool he is developing in his PAMP project [1] to explore the behavior of a server system under overload in more detail. We hope to gain valuable insights.

We have also some ideas on the last phase of the project. One option is to model the behavior of a large server system under overload. Another option is to actually build an adaptive system that uses the mechanisms developed in earlier phases of the project and during Thiemo's internship at IBM to dynamically prevent overload on large server systems. The system will also use the developed mechanisms to recover from overload, in case overload actually sets in.

2.1 Expected Results

The contributions of the first phase of our work are the implementation and experimental evaluation of mechanisms that support fast recovery from overload. Our main innovation is a mechanism that enables fast recovery without discarding work that has already been accepted by the system.

We aim at publishing our results in one workshop or conference paper for every phase of the project. Our short term goal is a first paper on fast overload recovery at the end of the year and a second paper on handling persistent connections under overload at the beginning of next year. In the course of the project we plan to publish at least one more paper. Together with Thiemo's previous publications and using the material from Thiemo's licentiate thesis [2] we expect that Thiemo will be able to present his doctoral dissertation in spring 2002.

2.2 Industrial Relevance and Collaborations

For this kind of research there is definitely an industrial interest since the mechanisms can be deployed in any Web server system. Also, any other type of server system that needs mechanisms to prevent and handle overload will benefit from our research. The possible systems include among others look-up services and billing information.

Since some of the input for this project stems from Thiemo's visit at IBM TJ Watson Research Center, we will further collaborate with researchers at IBM during this project. We have also an industrial contact person at Ericsson SARC, Dr. Lars Björnfot.

3 Relation to other PAMP Projects

We plan to collaborate with Lars Albertsson at SICS who uses complete system simulation to analyze high performance systems with quality of service requirements. While providing him with an interesting application with real-time requirements, we hope to get a more thorough understanding on how web server systems behave under overload. We hope that this insight will help us to identify bottlenecks and lead to the development of even more efficient mechanisms to avoid server overload and speed up recovery from overload. Furthermore, we hope to gain some understanding on how to use the developed mechanisms in a highly efficient way.

References

[1] Lars Albertsson and Peter Magnusson. Using complete system simulation for temporal debugging of general purpose operating systems and workloads. In *Proceedings of MASCOTS 2000*, August 2000.

[2] Thiemo Voigt. Providing quality of service guarantees to networked applications using the nemesis operating system. Licentiate Thesis, Uppsala University, September 1999.

[3] Thiemo Voigt, Renu Tewari, and Ashish Mehra. From servers to survivors: kernel mechanisms for handling web overload. submitted to Infocom 2001.

[4] Thiemo Voigt, Renu Tewari, and Ashish Mehra. In-kernel mechanisms for adaptive control of overloaded web servers. accepted for Eunice, Open European Summer school, September 2000.