Study of Reactive Solutions for Web Application Performance Enhancement during Overload

1Charulata S. Bonde and 2Prof. A.A. Sawant

1M.Tech (Computer Engineering), 2Professor Emeritus, Department of Computer Engineering and IT, College of Engineering, Pune, Maharashtra, India
E-mail: bondcs09.comp@coep.ac.in, charu.bonde@gmail.com, aas.comp@coep.ac.in

Abstract
In today’s world, almost all of the E-commerce websites or applications are available over the Internet. Such websites are often faced with incoming load of requests that exceeds their capacity, i.e. they are subjected to overload. Most existing web applications show severe drop in throughput and thus degraded performance at high overload.

There are multiple reactive solutions being suggested for such overload situations to enhance the performance of the web servers viz. Enhancing the caching ability at the web server, applying compressed caching for the underlying OS, applying optimum timeout for the TCP connections, effective overload handling etc.

The approach taken for this project is to enhance the web server performance by effective overload handling (overload detection, control etc)

This paper attempts to make a survey of the available suggestions / solutions about web server performance enhancement. It also comments on the advantages and limitations of these suggestions/solutions.

Index Terms: Web application; performance enhancement; reactive solutions;

Introduction

Overload
Cause for performance degradation Online services of today, such as banking, shopping, stock market trading are supported by Web-based multi-tiered server systems. Such services are exposed to variable load. The possible reasons for overload

Peak hour usage phenomenon
Events viz. sales, holiday shopping, or headline events. Peak load during such events can sometimes be orders of magnitude higher than average load, and can exceed the capacity of the system. When the incoming request rate on a system exceeds its processing capacity, it is said to be overloaded.

Overload consequences
Most server systems display unstable behavior when overloaded. Although ideally a system should operate at its maximum capacity when overloaded, many systems experience a drop in throughput (successful request completion rate), which is often drastic. The consequences of overload

• Increase in the response time
• Timing out of requests
• Abandoning the server system after being serviced for some amount of time.
• Request abandonments (either manual or protocol-triggered) lead to retries which further elevate the effective load on the servers.
• The overloaded server ends up being busy serving a large number of requests which timeout, resulting in a severe drop in throughput. This "feedback phenomenon" further deteriorates the performance of the website.

Approaches for overload handling
The problem of overload could be partially eliminated by proper server capacity planning through server duplication, data redundancy and request redirection; it cannot be fully eliminated. Unexpected peak hour usage of a Website can always happen; e.g. due to a major breaking news event for a popular news Web-site, or server-side failures that reduce total capacity. Since it is not prudent to size server systems for such occasional overload situations, a mechanism is required which specifically aims to keep the server system operating in a stable manner, even in the presence of overload. Overload can be controlled using two broad approaches; Pro-active and Reactive.

1. Pro-active approach: In this approach, the control mechanism prevents the system from getting overloaded by exercising admission control for the incoming requests. A fair amount of knowledge of the system’s capacity, a request’s resource needs and monitoring of system resources is required to be able to make an accurate decision about request admission. Such complex mechanisms are best employed when user QoS requirements (viz. max response delay, max tolerable precision loss etc.) are exclusively expressed, and when the server system is required to be a QoS-aware system, that provides specific and differentiated performance guarantees.

Many of the existing approaches depend on the knowledge of the utilization of a resource or estimation of the resource demands of the resource which has been identified as the bottleneck. This seems reasonable if in a system the bottleneck remains unchanged with the varying workloads.
However, in a complex multi-tiered server system, determining which resource is the bottleneck resource can be very difficult.

Furthermore, the bottleneck resource itself may vary with changing workloads or software and hardware configurations.

2. **Reactive approach:** Most Web-sites aim for a simple "best-effort" service, where the users do not express any explicit QoS targets - thus the system goals are those of ensuring stability on overload, maintaining the throughput near capacity, and response times that do not result in a large number of request abandonments. Such systems can activate an appropriate overload control mechanism only upon overload detection - a reactive overload control. For a reactive approach, two components are required: an overload detection mechanism and an overload control mechanism.

A number of existing approaches [2, 3, 4] use overload detection mechanisms based on resource utilization. These mechanisms assume that the potential bottleneck resource is known and can be monitored i.e. high utilization of this resource can indicate an overload. However, system bottlenecks may not be known a-priori; they may vary based on

- **Type of workload**
  - CPU intensive
  - Network I/O intensive etc.

- **Machine hardware configuration**
  - CPU speed
  - Network bandwidth
  - System cache and memory sizes etc.

- **Software configuration**
  - Thread pool size
  - Buffer size
  - Object pool size etc.

Thus, determining the bottleneck resource is nearly impossible in the case of multi-tiered heterogeneous systems which support varying workload mixes. This motivates the need for an overload detection mechanism that does not require the knowledge of the bottleneck resource, and therefore does not need to monitor it.

**Proactive Solutions**

There are many solutions to enhance the performance of the web application in overloaded situations viz.

- Increasing the processing power i.e. CPU capacity or adding multiple servers
- Load balancing of the servers
- Increasing memory size
- Increasing storage capacity
- Optimizing database in terms of data organization, number of connections, query optimization etc.
- Code optimization i.e. utilizing the resources properly, removing memory leakages etc.

All the above mentioned solutions deal with performance enhancement by hardware up gradation or software optimization. These solutions have some practical difficulties viz. Hardware upgrade increases the cost of implementation and software optimization can be time consuming or may not be possible in certain situations e.g. application is already implemented, running in production environment.

**Reactive Solutions**

To overcome these difficulties, numbers of solutions are being proposed for enhancing the web application performance in overloaded situations. These are the reactive solutions that could be applied on top of the existing application without disturbing it. Few significant solutions are mentioned below.

**Proxy based Self-tuned Overload Control Mechanism**

The paper [1] claims that an absolute indicator of a system in overload is when its throughput (rate of successful completion of requests) is lower than its request arrival rate. As long as requests arrive at a rate that the system can process them, the completion rate has to be close to the arrival rate. If the completion rate (smoothed and averaged, to ignore transient effects) drops below arrival rate, it is a clear indicator, that the server cannot process the requests at the rate they are arriving, and is hence, overloaded.

The paper [1], proposes a proxy-based, reactive overload control mechanism which uses the ratio of the throughput to the arrival rate as an indicator of overload. Overload is flagged by the proxy when this ratio is lower than 1 by a given amount (determined by a threshold value). On overload detection, the proxy uses a self-clocked admission control on incoming requests that are queued at the proxy. The request at the head of the queue is admitted into the server system, only when a request is seen successfully exiting from the server, indicating that there is room for a new request. The mechanism is similar in concept to window-based flow control mechanisms used in networking.

Thus, the mechanism is self-tuned, and requires no knowledge of the system hardware. It enables the server to operate at its maximum capacity while keeping response times within acceptable bounds even at very high overload.

**Advantages:**

- In overloaded situation, serves more number of requests with reduced response time than web server without proxy.
- Avoids web server crashing due to overload.
- Can be used for already deployed applications without changing a single line of code in the application.
- Web proxy is the only additional component needed.

Many web proxies are available as FOSS.

**Limitations**

- Does not guarantee that in overloaded situation, all the requests coming to the web server will be served with reduced response time.
Delayed caching at Web Cache Server

Mechanism
Web caching is an economic and efficient solution for the problem of degraded web server performance at the time of heavy load. Many of the existing solutions focus on increasing the caching capacity of the cache server instead of really enhancing the cache server throughput with available capacity. The paper [5] proposes delayed caching as a solution to enhance web server performance.

In case of overload, the caching proxies do not cache the data and just focus on serving the request. This means for every new request, response is provided but the caching of the response is avoided to save time and to serve more requests in available time. Because of this, when the server moves to normal condition after overload, the cache does not have latest served information and loading the cache with latest information when the request arrives; takes time. The delayed caching solution involves recording the meta-information of the requests at the time of overload and loading the cache with this information when the server moves back to normal condition.

Delayed caching can be seen as the method to improve the performance of the server in order to improve system reliability and provide a quick service to users’ service requests. The module for delayed caching can be implemented inside the caching web proxies viz. SQUID.

Advantages
- Caching reduces the time to fetch data from the repository.
- Delayed caching loads the cache after server exits from overloaded condition to normal condition and hence can serve new requests from cache itself.
- Web proxy is the only additional component needed.
- Many web proxies are available as FOSS.
- Can be used for already deployed applications without changing a single line of code in the application.

Limitations
- Maintaining the cache could be expensive especially in distributed scenario where web server(s) and web proxy are deployed on separate hosts.

Site based caching at the web cache server

Mechanism
The existing designs and solutions for web caching systems commonly make caching decisions based on document or uniform resource locator (URL) information. This site-based approach makes caching decisions based on the website that an object belongs to, rather than the object itself. The paper [6] shows that this new approach can benefit different scopes of cache design, ranging from internal operation of a single proxy (host level), mapping of proxy array in a local area network (LAN level), to load reduction in the global cache hierarchy (wireless area network (WAN) level). Since disks are usually the performance bottleneck in a proxy, to overcome this, site-based cache architecture is proposed that tries to store web objects belonging to the same site in nearby disk blocks. This new architecture reduces disk access time as compared to the conventional URL-based cache architecture. Besides, in the LAN-level design, a site-based mapping scheme can be used to map all requests targeting the same website to the same proxy, resulting in reduction in the total transmission control protocol connection overhead.

Advantages
- Caching reduces the time to fetch data from the repository.
- Reduces disk access time. Web proxy is the only additional component needed.
- Many web proxies are available as FOSS.
- Can be used for already deployed applications without changing a single line of code in the application.

Limitations
- Maintaining the cache could be expensive especially in distributed scenario where web server(s) and web proxy are deployed on separate hosts.

Main memory compression

Mechanism
Current web servers are highly multithreaded applications whose scalability benefits from the current multi-core/multiprocessor trend. However, some workloads cannot capitalize on this because their performance is limited by the available memory and/or the disk bandwidth, which prevents the server from taking advantage of the computing resources provided by the system. To solve this problem, paper [7] proposes the use of main memory compression techniques to increment the available memory and mitigate the disk bandwidth problem, allowing the web server to improve its use of CPU system resources.

This solution is proposed for Linux Operating system where the memory management subsystem is changed to use the compressed page cache (CPC). Compressed memory systems are based on the reservation of some physical memory to store compressed data, virtually increasing the amount of memory available to the applications. This extra memory reduces the number of accesses to the disk and allows the execution of applications with larger working sets without trashing.

Advantages
- Maximum data can be cached with the available physical memory.
- Faster as reduces disk accesses.
- Can be used for already deployed applications without changing a single line of code in the application.

Limitations:
- Deep knowledge about the operating system i.e. Linux is needed.

Active TCP Connection management and delay prediction

Mechanism
The paper [8] proposes a forward neural network model to predict the optimum TCP active connection timeout. This model and the calculations can be used for deciding the optimum session timeouts so that a web server can serve maximum requests in given time.
Active TCP connection deals with setting connection maintenance time when a TCP connection is established or a request arrives. During this period of time, the connection is always effective, and can be used to serve the HTTP requests that follow. Whenever a new request arrives, the server resets this connection maintenance time, once the connection maintenance time expires, the server closes the connection.

**Advantages**
- The unused connection can be closed depending on the timeout period suggested / predicted by the model and new connections can be made to serve newly arriving requests.
- Can be used for already deployed applications without changing a single line of code in the application.

**Limitations**
- The connection timeout period suggested / predicted by the model does not guarantee correct value in all the situations.

**Conclusion**
This paper surveys various suggested solutions for web server performance enhancement during overload. Implementing any of these mechanisms can improve the performance of the web application without changing the code of the already implemented web application. Multiple mechanisms can also be implemented in parallel to give significant performance enhancement.

The following table summarizes the features of the above mentioned approaches:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Proxy based self-tuned overload control</th>
<th>Delayed caching at web cache server</th>
<th>Sit-based caching at web cache server</th>
<th>Main memory compression</th>
<th>Active TCP connection management and delay prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique used for performance enhancement</td>
<td>Request admission control</td>
<td>Caching</td>
<td>Caching</td>
<td>Modification of memory page structure</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>Component available as PDSS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Additional components needed</td>
<td>Web proxy</td>
<td>Web cache server</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Software modules implemented or embedded</td>
<td>Load detection, overload control modules in web proxy</td>
<td>Load detection, delayed caching module in web cache server</td>
<td>Site-based caching module in web cache server</td>
<td>Compressed page cache module in web cache server</td>
<td>Active TCP connection management module in web server</td>
</tr>
<tr>
<td>Is OS specific</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**References**