

# Testing and Improving Web Application Performance

Thomas Zwanzinger  
email: e0125069@student.tuwien.ac.at  
July 29, 2004

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Performance Testing Objectives</b>	<b>3</b>
<b>3</b>	<b>Creating workloads</b>	<b>4</b>
3.1	Workload Characterization History . . . . .	4
3.2	Workloads in a client/server environment . . . . .	4
3.3	The Self-Similar character of web traffic . . . . .	5
3.4	Using web usage patterns . . . . .	6
<b>4</b>	<b>Issues on running a performance test</b>	<b>7</b>
<b>5</b>	<b>Server Performance Issues</b>	<b>9</b>
<b>6</b>	<b>Multiple Web Server and Load Balancing</b>	<b>10</b>
6.1	Using Multiple Web servers . . . . .	10
6.2	Load Balancing through Round-Robin-DNS . . . . .	10
6.3	Load Balancing through Connection Routers . . . . .	11
6.4	Load Balancing through Application Routing . . . . .	12
<b>7</b>	<b>Caching Efficiently</b>	<b>12</b>
7.1	Caching Overview . . . . .	12
7.2	Cache Consistency . . . . .	13
7.3	Invalidates and Updates . . . . .	14
7.4	Cachability Myths . . . . .	15
7.5	Caching through http header information . . . . .	16
<b>8</b>	<b>Performance Comparison of Middleware architectures for generating dynamic web content</b>	<b>16</b>
8.1	Language Overview . . . . .	17
8.2	Performance Comparison . . . . .	18

<b>9</b>	<b>Miscellaneous Studies and Techniques about Enhancing Performance</b>	<b>18</b>
9.1	Performance enhancements through Invariants in Web Server Workloads . . . . .	19
9.2	Performance Issues of a Web database . . . . .	20
9.2.1	Additional Metrics . . . . .	20
9.2.2	Relations between different Web database parameters . . . . .	21
9.3	Overuse of HTTP Services . . . . .	21
<b>10</b>	<b>Conclusion</b>	<b>22</b>
<b>11</b>	<b>Bibliography</b>	<b>23</b>

## Abstract

Performance Testing is a crucial part of a Quality Testing Process in a web application system. Some Issues of testing web application performance are discussed. The author characterizes workloads of websites and limits to Client side testing. Having identified the bottlenecks of a given web system, various ways of solving the performance problems are given. Techniques like caching, replication with load balancing and web accelerators are introduced. Finally three common web middleware systems are compared and performance issues of a web database system are discussed.

## 1 Introduction

In most Software Engineering Process Models, software testing has an important role in guaranteeing the quality of a software product. In most software projects, the greatest part of the testing effort goes to functional testing, and some software testers may tend to ignore performance testing of their software product altogether.

For websites and Web applications, especially in an e-commerce situation, performance testing is crucial. Even a completely bug-free web application is doomed to fail if it can only serve an average amount of traffic, but is not able to handle the significant peak loads in a real life situation. To ensure that a web application satisfies certain performance criteria like data throughput or response time, exhaustive testing in an environment as close to the future working environment as possible is necessary.

This paper describes issues that have to be considered while performing load and performance tests on web application systems. After that, it will describe various performance bottlenecks and how to get rid of them. It compares some common middleware architectures used in web application systems and tries to characterizes a real life workload of web servers.

## 2 Performance Testing Objectives

In this paper we focus on performance evaluation through the eyes of the user, so data throughput and response times need to be analyzed. We try to describe all activities involved in evaluating the actual performance of a web application system in the field.

Considering such a framework, Vokolos et. al. [34] suggests a number of different goals considering performance testing, including following:

- Test cases need to be designed towards performance criteria rather than functional correctness criteria.
- Metrics need to be defined to get quantitative results out of performance tests.
- The comparison of different hardware platforms or architectures for a given application

Several things can be measured when evaluating a web applications performance: resource usage, throughput, response time and even queue lengths, describing the average or maximum

number of tasks waiting to be served. Some resources that ought to be considered include network bandwidth requirements, CPU cycles, disk access operations and memory usage. Database access rates could be an important criteria, too.

Results of performance testing can have serious implications on the design of a web application. Therefore the testing process itself and the resulting actions may be expensive and time-consuming. Nevertheless, on the long time run performance testing can be considered cheap, thinking of a slow and unstable web application that makes customers look for a different seller or website.

### 3 Creating workloads

A traditional way to do performance testing is to create a *benchmark*. A *benchmark* is considered a representative workload of the system in its production environment. Of course, the idea of a representative workload is problematic. Where does the data come from? Can we measure how representative a workload really is? A second issue that must be considered is, whether the workload describes an average load or a stress load. Given that the necessary statistical material from previous systems is at hand, what will be the timeframe of these numbers? The peak load of a 24-hours day will be different than the load maximum a whole month. This section describes some ways the characterize workloads in different situations.

#### 3.1 Workload Characterization History

Calzarossa et. al. [7] describes workload characterization in various systems at length. The author concludes that the performance of any given system cannot be determined without characterizing the workload, meaning the requests the system is going to process.

“Workload characterization consists of a description of the workload by means of quantitative parameters and functions. The objective is to derive a model able to show, capture, and reproduce the behavior of the workload and its most important features.” *Calzarossa et. al. [7]*

Workload characterization started in the early 70’s. When computers meant speaking of mainframes, workloads were generally batch jobs and transactions. Increasing processor power and graphical interfaces with the design of Personal Computers and creation of local networks has opened the system to new processing requirements. Distributed systems and client/server applications provided new services as well.

The rapid growth of the world wide web leads to the idea of multimedia workloads, consisting of different types of application (ftp, audio & video streams, ...). These applications have in common that they have different needs on resources of the server and clients as well as on the network.

#### 3.2 Workloads in a client/server environment

A client/server environment consists of numerous clients connected to a number of servers through a network, generally a LAN or the WWW. Beside web application systems, distributed file systems, distributed databases and distributed multimedia systems are examples of such an environment.

In these environments, servers wait for requests initiated by different clients. These requests are transported through the connecting network. The server processes these requests and returns the result. Some web applications may be even more hierarchically structured, called n-tiered system, where servers may act as clients as well. Workload characterization normally builds on the data collected at each layer, typically in the means of server logs or software monitors like the well known *tcpdump* which captures all the packets send on TCP protocol level.

Arlitt [4] describes log files common to NCSA httpds v. 1.4 [15] which manages an *access log*, an *agent log*, an *error log* and a *referer log*. The access log captures information about the requests and responses processed by the server. The agent log includes information about the browser types used by the client. The error log collects events that may need an administrators help. The referer log contains information on which web pages are linked to documents on the Web server.

The Access log consists of a number of line following this standard:

```
hostname - - [dd/mm/yyyy:hh:mm:ss tz] request status bytes
```

Due to this, it is possible to determine the name of the client, the current time and the requested document. Response information and the number of bytes transfered is also documented.

In most cases access logs are used to create a proper workload model, but generally logs do not contain all the information that is of interest. Log file entries only describe the actual amount of bytes transferred, but not the transferred documents size. No information of the available documents of the server is given and the elapsed time during the document transfer is not reported. Of course, such logs cannot differ between human initiated requests through a browser or software initiated requests (e.g., by a Web Crawler).

### 3.3 The Self-Similar character of web traffic

Traffic in local or wide area networks was typically modeled using different distribution models like the Poisson distribution. These distributions have the property that over an increasing time scale the characteristic burst lengths, which are moments of high traffic, would be smoothed.

Some recent paper suggest that WWW traffic may be self-similar. The term “self-similarity” was originally associated with fractals. Fractals look the same in whichever scale they are viewed. In the case of stochastic objects like timeseries, self-similarity is used in the distributional sense: when viewed at varying scales, the object’s distribution is unchanged.

In figure 1 we see 4 different timescales taken from the same source that all show a bursty plot. Another characteristic is its long-range dependence, meaning the observed values for traffic volume (bytes per one second interval) at any time are correlated with the values of all future instances.

Crovella et. al. [13] try to explain the reasons for these unexpected properties of Internet traffic: A self-similar process may be constructed by superimposing many simple renewal reward processes, in which the rewards are restricted to the values 0 and 1, and in which the inter-renewal times are heavy-tailed. For Internet traffic, “1” represents a file transfer, while “0” acts for a time where no file transfer occurs.

The distribution of transmission times are heavy tailed due to the heavy tailed distribution of web files. The time where no traffic accrues has also a heavy tailed distribution. As Transmission times are much heavier tailed then No-Traffic times, this is likely the primary determiner of

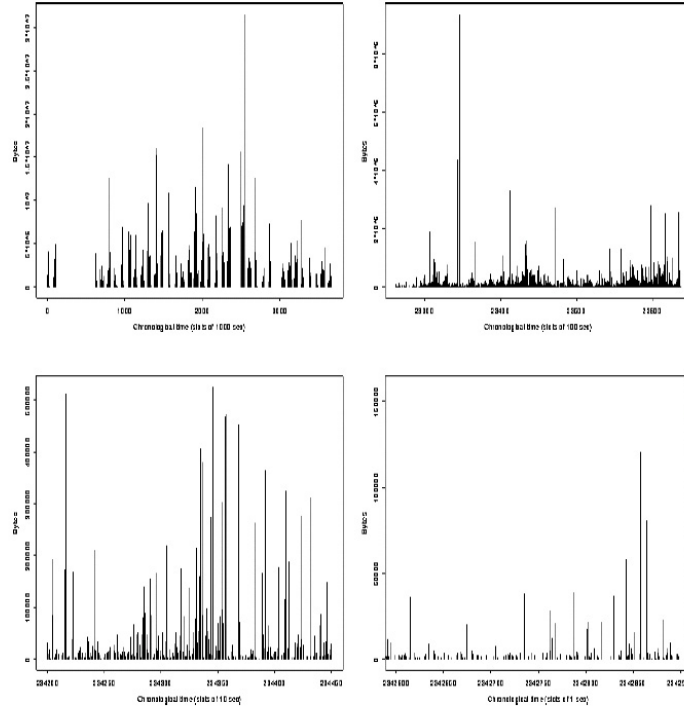


Figure 1: Traffic Bursts over Four Orders of Magnitude; Upper Left: 1000, Upper Right: 100, Lower Left: 10, and Lower Right: 1 second intervals [13]

Web traffic self-similarity. Additionally, Crovella et. al. [13] suggests that changes in protocol processing or document display are not likely to remove the self-similarity of Web traffic, only reduce it.

### 3.4 Using web usage patterns

As shown above, most workload characterization is done through analyzing logs from various servers. Kobsal et. al. [22] describe an approach using both web server logs and web usage patterns. The study focus on user modeling systems and refers to Rozanski et. al. [31] The authors summarized client side logs to generate a classification of web users:

**Quickie sessions (8%):** These are short (one minute) visits to one or two familiar sites, to extract specific bits of information (e.g., stock quotes, sports results). Users visit 2.2 pages per site on average, and spend about 15 seconds on a page.

**Just the Facts sessions (15%):** Here users seek and evaluate specific pieces of information at related sites (e.g., compare product offers). Sessions last 9 minutes on average. Users visit 10.5 sites and 1.7 pages per site, with about 30 sec. per page.

**Single Mission sessions (7%):** Users focus on gathering specific information or completing concrete tasks (e.g., finding the website of a scientific conference and registering for it). They visit two websites on average, which belong to the same category (e.g., search

engines or portals). Users quite carefully read the content of (frequently unfamiliar) web pages in approximately 90 seconds. The average session length is 10 minutes, and 3.3 pages per site are being visited.

**Do It Again sessions (14%):** These are focused on sites with which the user is familiar (e.g., online banks, chat rooms). Users spend about two minutes for each page. The average session lasts 14 minutes, with 2.1 sites and 3.3 pages per site being visited.

**Loitering sessions (16%):** Users visit familiar "sticky" sites, such as news, gaming, telecommunications/ISP, and entertainment. Sessions last 33 minutes, with 8.5 sites and 1.9 pages per site being visited (two minutes per page on average).

**Information Please sessions (17%):** Users gather broad information from a range of often unfamiliar websites from several categories (e.g., they collect facts about a specific car model, find a dealer, negotiate a trade-in, and arrange a loan). Users visit 19.7 websites and 1.9 pages per site. The average session length is 37 minutes, and pages are viewed for one minute on average.

**Surfing sessions (23%):** They appear random, with users visiting nearly 45 sites in 70 minutes on average (about one minute per page and 1.6 pages per site).

Kobsa et al. [22] used this information, especially request interval and relative type frequency to create a test plan for a medium and a great sized user modeling web application system. They claim that this testing approach makes performance testing under various workloads more realistic.

## 4 Issues on running a performance test

Several tools are available to simulate a great number of clients on a dedicated client platform. While presenting the *httperf* tool, Mosberger et al. [27] describe general issues while using a tool that simulates a potentially infinite user base.

While using a tool that simulates client users to test performance on a server system seems to be a straight forward approach, early tools only simulated a fixed sized number of clients. These tools include early versions of SPECweb or WebStone. For more realistic measurement it is important to simulate a variable sized user base, describing the bursty nature of web traffic.

While performing performance tests, it is important to keep in mind the limits of load a client can sustain. Besides the obvious CPU and Memory performance limits, there is a surprising variety of possible client side bottlenecks that may falsify the result. Two important resource limits are described:

**Size of TCP port space:** Due to the technical specification of the TCP protocol [19], a performance tool may use 64512 port numbers at most. Since a given port number cannot be reused until the TCP TIME-WAIT state expires, this may limit the client sustainable offered rate. Thinking of a BSD-derived OS with a 1 minute timeout, the maximum sustainable rate per client is 1075 request per second. Using the RFC-793 [14], the recommend timeout grows to 4 minutes leading to a maximum rate of 268 requests per second.

The RFC-1122 [5] suggests a way to handle problems concerning the TCP TIME-WAIT timeout: A connection may be reopened directly with a SYN<sup>1</sup> from the TCP TIME-WAIT state if its initial sequence number for the new connection is larger than the largest sequence number it used on the previous connection incarnation and returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

The RFC-1644 [6] describes the T/TCP Protocol that implements this behaviour. However, an entry in a public newsgroup [38] suggests that today's implementations of TCP in Operating Systems make use of this possibility as well. This assumption is further hardened by the fact, that IP-options for the Linux Kernel 2.4 [2] contain the parameter `tcp_max_tw_buckets` :

“The `tcp_max_tw_buckets` variable tells the system the maximum number of sockets in TIME-WAIT to be held simultaneously. If this number is exceeded, the exceeding sockets are destroyed and a warning message is printed to you. The reason for this limit to exist is to get rid of really simple DoS attacks.”  
*Oskar Andreasson [2]*

**Number of open file descriptors:** Most OS limit the number of possible file descriptors. Figures range from 256 to 2048. although file descriptors may be reused, the timeout value that was configured by the user of the testing tool limits the number of requests per second. Assuming a timeout of 5 seconds and 2000 possible open file descriptors, a rate of 400 requests per second is possible. Increasing the possible number of open file descriptors and decreasing the configured timeout value may increase this rate.

Limits to open file descriptors on Linux Systems have gone through a change during the development of Linux Kernels. Generally there are 3 different limits on the maximum number of file descriptors:

**Soft-Limit:** This limit for processes can be changed by the user itself. Often a command like “`ulimit`” can be used to change this limit.

**Hard-Limit:** The Hard-Limit is a system-wide boundary limiting all other Soft-Limits set by users. Only the root user can modify this limit.

**Kernel-Limit:** The Kernel-Limit has gone through some changes during the development of different Linux Kernels. In Linux Kernel 2.2., this limit can only be changed by recompiling the whole Kernel, a laborious task. Newer Linux Kernels may set this limit by changing entries in system-files, e.g. for Solaris 2.4+ in `/etc/system` [36], or some commands have to be executed during startup, e.g. Linux Kernel 2.4. requires following commands at startup:

```
“echo 16384 > /proc/sys/fs/file-max” and  
“ulimit -n <limit>” as root user [35], [3].
```

However, old binaries or some system calls may not be able to handle high limits for open file descriptors, e.g.: the `stdio`-library or the `select()` system call (in Solaris Operation Systems) [36].

---

<sup>1</sup>A TCP Message containing a set SYN-flag, used for initializing a TCP connection through the 3-Way-Handshake [19]

It is often difficult to predict when the client platform becomes the performance bottleneck of a performance test, it is important to verify that the measured performance is indeed due to server bottlenecks. This may be done by varying the number of client machines participating in the test.

Measuring the throughput of a server system seems easy: after sending a number of request, count the number of replies of the server and divide it by the time it took the test to complete. This approach faces two problems: first, to compute a number describing the robustness of a given server system, it is important to run the test several times and this takes a lot of time. Second, computing only one throughput estimate over the whole test period hides variations on a time frame shorter than the performed test. A testing tool should therefore compute several estimates during test execution and offer this information besides the overall test result. Statistical analysis may show that the seen confidence intervals can be computed without having to make assumptions on the distribution of the samples.

The following section describes ways to heighten web application performance. As there are many ways to do this, only some techniques are discussed here. Choosing the right technique has to be done by analyzing the generated test results and identifying the given bottlenecks of the web application.

## 5 Server Performance Issues

Clearly it is very interesting to know, how quickly a given Web server can respond to requests, or how well it scales with load. But what happens if overload occurs? Asking these questions, it is important to understand how Web servers actually work.

Web servers are applications that have to communicate with possibly thousands of client computers. This leads to a great amount of concurrency. Using non-pre-emptive First-Come First-Serve (FCFS) tactics have to fail due to the sheer number of requests and connection times may last minutes. Another reason is that clients frequently request files not stored in memory and while wait for the IO operation to complete, other requests could be served. A interesting factor while talking about Web server is the architectural model they are based on. A. Wolman [21] describes 4 different models:

**Process-Based Servers** Processes are the most common form of handling concurrency on a single system. The original NCSA server [15] or the widely known Apache server [29] use processes to handle individual requests. The advantages of this model are the easy implementation and the isolation and protection between different clients. The main disadvantage is the relatively heavy weightiness of processes in most Operating Systems.

**Thread Based Servers** Threads work like processes but are considered light-weighted as several threads run in the same process and share the same address space but provide a separate stack. Switching between different threads is cheaper than switching between processes. Servers that use threads include JAWS [20] and Sun's Java Web Server [26]. Threads have disadvantages as well, as threads are not protected from other threads as different processes are. Thus, program failures in a thread may crash the whole server.

**Event-Driven Servers** An event driven architecture uses a single process with a non-blocking I/O and asynchronous reads and writes on a socket. When a read or write

operation is not possible, the system call returns immediately and the server may process other request. The OS later notifies the server through a *Notification mechanism*. Event-driven Web servers like Flash [28] or Zeus [33] are generally very fast. Of course, like in a thread based Web server failures may easily lead to the crash of the whole server. OS limits like open file-descriptors become an issue as well. Implementing such Web servers is difficult because of concurrency challenges.

**In-Kernel Servers** Servers like AFPD [30] and Tux [18] pack situate their code in kernel-space. This is an extremely fast way but leads to great risks, as a failure may crash the whole OS! Implementation is very difficult and dynamic content a great challenge. Due to this, in-kernel Web server often redirect dynamic websites to other Web servers like an Apache server.

There is clearly no best model for designing a Web server, but different approaches may be better for different situations. Dedicated server appliances may lead to an in-kernel approach for its performance. Dynamic content server will rely on a more general Process-based approach like the Apache Server. However, it is important to keep in mind these differences while deciding on a Web server.

## 6 Multiple Web Server and Load Balancing

### 6.1 Using Multiple Web servers

Only small sized websites can rely on only one web server to serve all request from clients. Generally, a number of Web servers together try to handle the incoming request. There are several possibilities to distribute incoming requests. One way is to use additional web servers as mirrors, often distributed geographically. Generally users have to select a proper mirror based on the country they are living in.

This approach has some problems. First, users become responsible for selecting a mirror, so the website loses control which web server will be used to query the request. This may lead to an uneven distribution of load on the different mirrors. This way, users have to react to web server failures themselves as well. Additionally, there may be administrative problems keeping these individual computers updated.

Due to these problems, it is easier to have several servers close together, using distributed file systems, shared databases or replication through independent file systems. To distribute incoming request equally, load balancing through a Load Balancer as shown in Figure 1 is necessary. Following techniques are described in Jim Challenger et. al. [9] and Arun Iyengar et. al. [21].

### 6.2 Load Balancing through Round-Robin-DNS

One way to perform distribution is through a Round-Robin Domain Name Server (RR-DNS). This server allows a single domain name to be associated with multiple ip-addresses. As Clients use DNS to resolve their domain name, clients will be connected to different server machines using the Round-Robin Algorithm.

Round-Robin Algorithm works as following:

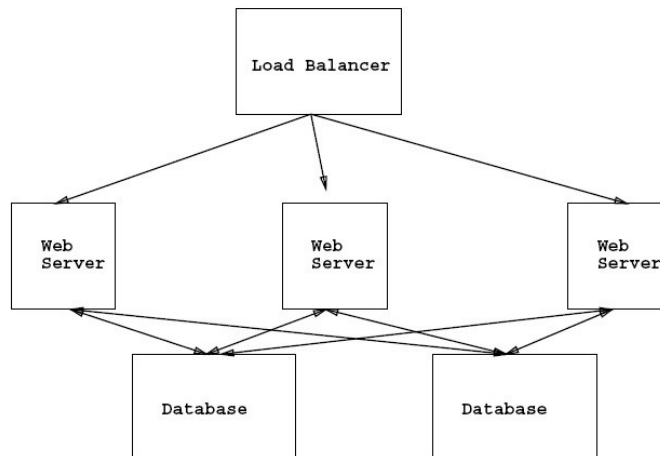


Figure 2: Architecture of a scalable Web site. Requests are directed from the load balancer to one of several Web servers. The Web server may access one or more databases for creating content [21]

“Round-Robin: A scheduling algorithm in which processes are activated in a fixed cyclic order. Those which cannot process because they are waiting for some event [...] simply return control to the scheduler.” *William Stallings [32]*

Round Robin scheduling has some problems. First, caching of name-to-IP addresses at name servers may cause load imbalance. Before the DNS request of the client leads to the RR-DNS server, several other DNS servers are contacted and these servers may have cache that stores the IP-addresses of the Web servers. The only way to enforce forwarding these requests to the RR-DNS server is setting the time-to-live (TTL) value to a low value. Nevertheless, many DNS server ignore too low TTL values and store IP-Address information nonetheless. Additionally, clients normally cache these IP-addresses too. If server nodes fail, this caching makes high availability of web services difficult. Finally, Round-Robin Algorithm is often a too simplistic method for realizing good load balancing.

### 6.3 Load Balancing through Connection Routers

Another approach to load balancing is the use of a connection router. These routers hide the IP-Address of the underlying web servers and forwards connections to the Web servers. So IP-Address caching does not limit load balancing in any way and Web servers may be connected and disconnected any time. Good Connection Routers only handle request from the client but let the response performed by the web server itself, reducing the overhead of connection forwarding. A problem arises while using SSL. SSL adds security the http protocol and uses session keys that have a TTL of 100 seconds because generating session keys is considered expensive. Routing SSL session to different web servers leads to generation of multiple session keys in the 100 second timeframe. Some Connection Routers therefore make SSL connections “sticky“ so that such requests are forwarded to the same web server.

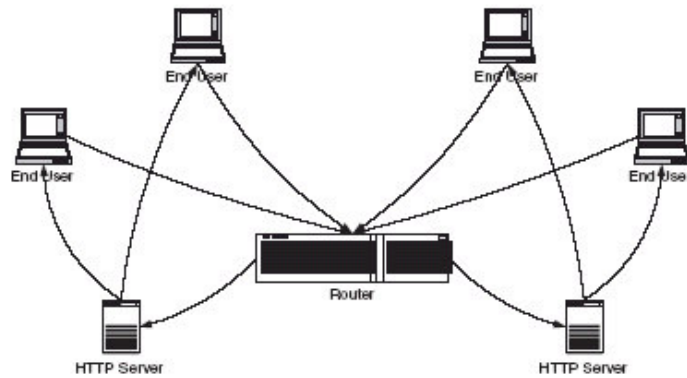


Figure 3: Architecture of a scalable Web site. Requests are directed from the load balancer to one of several Web servers. The Web server may access one or more databases for creating content [9]

If a single Connection Router has insufficient capacity to route requests to a site without becoming a bottleneck, a hybrid approach may be attempted by using both RR-DNS routing in combination with Connection Routers, using router nodes as hooks for the RR-DNS server.

## 6.4 Load Balancing through Application Routing

Instead of routing on the Level 4 of the Open System Interconnection Reference Model (OSI model), routing could also be performed on the Level 7, also called content driven routing. Using such a router, more complicated routing strategies can be applied like routing static content to a different set of Web servers as dynamic content.

The key problem in using such an approach is the high overhead of application level routing. All data may have to be transmitted through the router in a simple implementation. Better performance is achieved by using the TCP handoff protocol in which the client connection itself is forwarded to the Web server.

## 7 Caching Efficiently

### 7.1 Caching Overview

Web caching is a possibility to significantly reduces latency on a Web server. Server-side Caches hold frequently requested files in memory and serve these files quickly to the requester rather than reading them from a (possibly remote) hard disk.

Objects stored in web caches normally have lifetimes. Has the lifetime of an object expired or the object was invalidated in any way, the object cannot be served directly from the cache. The cache-hit situation can be classified into 2 categories:

**Hit-Fresh** The cached object is still within its lifetime and therefore is treated as the remote object on the server. The cache can serve the data directly.

**Hit-Stale** The cached object was invalidated or it has expired. Now the cache has to retrieve the requested document from the origin server and forwarding it to the client.

Of course, only *Hit-Fresh* can reduce network latency, because the cache can serve the data directly. Even if the remote server signals the cache that the requested content has not changed, the communication overhead is rather high compared to a Hit-Fresh.

Generally Web servers provide two different types of content: static and dynamic content. Static content are websites or files that can be served directly. Only data transfer occurs on server side. Dynamic pages are completely or partly generated by the web server or any background server (e.g. application server). While it is possible for a Web server to serve a great number of static content per second, dynamic content suffers from a high processing overhead. Very often dynamic content is the main bottleneck of a web application system. While caching static content is easy, caching dynamic data is a challenge as consistency management is difficult or even impossible, thinking on personalized homepages like the website of Amazon.com [17].

## 7.2 Cache Consistency

Wherever a caching technique is applied, the technical focus is on providing the necessary degree of consistency. As caching can be seen as a way of replicating data on a number of machines, the replicated data needs to be updated frequently. How often depends on the type of data stored in the cache. This degree of consistency is one of the main issue of designing and implementing concrete cache mechanisms for a web application.

Arun Iyengar et. al. [21] describe following degrees of consistency:

**strong consistency:** If a cache always returns a file as it is stored in the server, the cache is considered strongly consistent. This level of consistency can hardly be arranged for practical reasons, as the involved overhead for ensuring this consistency is very high. If it is necessary, typically a two-phase message exchange is used while handling unbound message delays by using timeouts.

**delta consistency:** A cache is considered  $\delta$ -consistent if  $\delta$  time-units may pass after the latest write-commit was done at the server until it is also updated in the cache. Necessarily,  $\delta$  ought to be greater than the network delay.

**weak consistency:** As the name implies, this “weak“ consistency is already given if the data in the cache represents some correct previous data from the server.

**mutual consistency:** In this case it is guaranteed that the data is mutually consistent in respect to a defined group of objects. This means that no-one in the group has a more current set of data than any other member of the group. This consistency level may be achieved while following one of the types described above.

Strong consistency is often enforced for mirror sites that need to reflect the current state of the base server. Financial transaction often require strong consistency, too. If outdated data can be tolerated for a given time limit, delta-consistency is sufficient, if the possible network delay is

somehow bounded to an certain value. Cooperating systems that need to have the same data base may require a mutual consistency level to ensure all the group members have either the newest or a agreed older version of a data set.

On the web, weak consistency guarantees are given most of the time. This is tolerable for systems that serve data that is static and updated only frequently, as users often tolerate receiving stale data. On the other hand, Web applications often handle requests that require to write data as frequently as they require to read data. This type of Web applications will grow, and as servers tend to handle the consistency problem in marking their served data as uncachable, effectiveness of proxy caching is significantly reduced.

Mechanisms used by cache proxies to provide a given level of consistency are following:

**server-driven mechanisms:** This technique, also referred as *server-based invalidation*, can be used to provide strong or delta consistency. The server is responsible to inform the proxies when a write occurs and their data is not valid any more. For the server this means managing the state of remote proxies and possibly delaying writes to data if a proxy is not reachable due to network failures.

**client driven mechanisms:** the approach, also known as *client-polling*, requires for strong consistency to contact the server on every read action. However, the server does not need to handle any information of the proxies. Weak consistency is easily enforced by this scheme if the server explicitly specifies a time-to-live value (TTL) for an object. Proxies may also use *periodic polling* of the server to verify the data from time to time. TTL values are generally set as a part of the http response in an EXPIRES tag or using the CACHE-CONTROL header. Such an *a-priori* knowledge of when the object will be changed is rather difficult.

**leases:** This technique may guarantee strong consistency and provides a trade-off between keeping state information of proxies on the server and reducing the messaging overhead for keeping data consistent. Proxies may request a *lease* for a set of data. Leases are valid for a given period of time and guarantee the proxy that it will be informed if the data changes. If the lease runs out, the proxy has to request another lease. Smaller lease duration reduces the server state space overhead, but increases the number of control (lease renewal) message exchange and vice versa.

### 7.3 Invalidates and Updates

With a server driven consistency approach, the origin server has to inform the registered proxies of any change to the data. This notification can either be an invalidation message or the changed data itself. While sending only an invalidation note has the advantage of inflicting only a small overhead, if the data has to be served by the proxy a delay occurs because the proxy has to fetch the data from the server.

This approach seems appropriate if the data is not accessed very often through the proxy. So for more popular objects it is better to transfer the changed data to the proxy rather than sending an invalidation, while for less popular data the invalidation mechanism should be used. Ideally, the server should be aware of this and send invalidates and updates based on the nature of the data it serves using statistical data and/or data size information.

## 7.4 Cachability Myths

For a long time designers thought that dynamic data is not cacheable. Jim Challenger et. al. [9] suggest that a great deal of data that is considered dynamic is indeed highly cacheable. The difficulty with frequently changing pages is controlling cache coherence. If frequently requested pages can be cached, this will lead to considerable savings, e.g. sport pages displaying the scores of sport events are often highly requested by users and can possibly be cached.

Of course, if the cost of caching exceeds the cost of generating a dynamic page, a page can be considered un-cacheable. An example for this would be an airline reservation system, as the important information changes too frequently and the number of pages affected is too great. On other websites it is tolerable that the information shown is not totally up-to-date. If caching is to be used server-side, it is important to consider the rate in which content changes on different web pages and how up-to-date an information needs to be for the user.

One technique for improving performance of dynamic data is to cache dynamic content the first time it is created. If the same dynamic content is requested, the cache can serve the data rather than the server, which would generate the page anew. To keep the data consistent, the server should manage the cache explicitly instead of trusting in expiration times. This is possible if the cache proxy can be managed through an API.

Pages that change with the user, so called personalized pages, normally create a significant load on the dynamic content generating facility. To reduce the load, caching can also be used. Instead of caching whole pages, pages are structured into fragments. Fragments that contain information not specific to the user can be cached, the others not. The different fragments need to be assembled, but the overhead for this process is smaller than generating the page from scratch.

Even in bank accounts or online stores, caching can be used. The idea is, that many users often view their “shopping kart“ more often than they add or remove items. A cache that is controlled by the server could be used to store pre-generated information about the user in the cache. That could activate a trigger in a database that changes the content of the cache if the user performs an action that changes a database entry. After logout or a given period of time without action, the information is removed from the cache. Especially web accelerators (see further below) could be used for such purposes.

News content on a website is normally highly cacheable because a lot of information is normally available about the frequency of news changes, so update rate and request rate can be estimated. In such cases it is better to actively push the website into the cache, before it is requested to minimize latency and problems with the expiration of those highly accessed sites from the cache.

If a site is accessed very frequently, it could be more efficient to pre-generate all pages and serve them as flat files rather than generating them on every hit. This is rather easy for pages with less dynamic content, but may also be viable for more dynamic pages. The key element is to rely on the database to generate the appropriate html-files when its content changes through database triggers available in modern database implementations. Database triggers lead to some problems, though. If a transaction has to be rolled back, this could lead to triggers already activated and to inconsistent pages. Triggering on each database update results in expensive redundant triggers that can be difficult to manage. Some of this problems can be solved by using a commit table, which summarizes all update triggers and is processed after the commit succeeded, solving the problem of rollback. Instead of database triggers, log following

programs or content generators can be utilized.

## 7.5 Caching through http header information

An easy and very general approach for caches is to rely on information of the http headers to decide if data from a web site is cacheable and how long it is able to remain in the cache without updating. Jun-Li Yuan et. al. [37] point out that many important headers of http documents are not sent or set to unnecessary inefficient parameters due to the configuration of the web server. That leads to cachable web pages that are not cached.

In a study, Jun-Li Yuan et. al. [37] analyze a well-known web caching system named Squid. This is a free, widely used open-source web caching system. The version that was used in the study was 2.4.STABLE3. After analyzing the algorithm used to verify if an object is cacheable or how long it can be cached, the author concludes that following http headers are of a main concern for caching: Cache-Control, Vary, Pragma, Expires, Last-Modified, Date and Content-Length. In order for a site to be efficiently cacheable, these headers should be set if possible. After analyzing traces of the National Laboratory of Applied Network Research in the USA, the author finds that the absence of the Last-Modified and the absence of the Expires header sum up 39% of the uncachable objects. Through further analysing of the traces it is possible to say that the absence of these headers is due to a improper setting of the Web server.

The Time-to-live (TTL) of an object in the cache was also a topic of this study. According to Squid, the TTL of an object is defined by a few headers such as "Expires" and "Cache-Control: max-age". If these parameters are absent, a heuristic is used to determine the TTL. By configuring Web servers properly to make them send objects with proper lifetimes, the rate of cache hits can be scientifically improved. Providing explicit lifetime headers as "Expires" are preferred. The "Last-Modified" header only result in a heuristic evaluation of the lifetime by Squid.

In the study, 78 % of the objects with explicit lifetime headers have not changed. This is in line with other studies that say that most of the time objects could be held longer in the cache. This leads to the idea, that especially static contents or embedded objects the lifetime should be significantly increased.

## 8 Performance Comparison of Middleware architectures for generating dynamic web content

This chapter relies on Emmanuel Cecchet1 et. al. [8] which compares commonly used middleware architectures about their performance. Dynamic web content is normally generated by a combination of 2 or 3 different parts: a Web server, a dynamic content generator and very often a database. Figure 3 shows this configuration and the protocols used to perform the interaction of these parts. The web server delivers all static content to the user requesting this file. If the user request a page containing dynamic content, the web server forwards the request to the dynamic content generator, which generates the page. To do that, the generator executes the code written for this specific business logic of the web site and makes use of other files or a connected database, normally using standard SQL Queries. The content generator puts all the information generated by code and the static content associated with the web page and puts



Figure 4: A typical configuration of a dynamic content web site

these parts together. Then it forwards the whole page to the Web server who delivers it to the user.

There are various ways to capture the business logic of a web site. Following script languages are used quite often: PHP [16], that executes as a module of the web server like Apache [29], Microsoft Active Server pages (ASP) [11] that are integrated in the IIS server [12], Java servlets [25] that execute in a separate java virtual machine and full application servers like the Enterprise Java Beans server (EJB) [24]. The rest of the chapter concentrates on comparing PHP scripts, Java servlets and EJB.

## 8.1 Language Overview

PHP is a scripting language which has the capability that SQL queries can be embedded in the scripts and forwarded to the database. In a similar fashion, Java Servlets can integrate SQL queries that can be send to the database. EJB works different, as the programmer defines so called “beans”. “Session beans” capture the application logic of the web site, while “entity beans” implement the persistence of data. Generally, an entity bean can be compared to a database table and a entity bean instance corresponds to a row in a table. Using container managed persistence, these objects are used to get information out of the database by calling bean methods, which in turn use SQL queries to communicate with the database. These SQL queries are generated automatically <sup>2</sup>.

An advantage of Java Servlets and EJB over PHP scripts is the possibility to use the integrated synchronization methods of java to offload some of the synchronization and locking typically done by the database. This may lead to performance gains, which will be discussed further in this chapter.

PHP executes as a module in the Web server, sharing the same address space. Java Servlets and EJB run in a java virtual machine that is separated from the Web server. This makes interprocess communication between the Web server and the java virtual machine necessary. Although this lessens the overall performance, the advantage of this separation is that the java virtual machine may run on a different hardware than the Web server. If the demand on the Web server is rather high, using a second machine can significantly reduce the strain on the Web server. As EJB introduces a significant overhead on the generation of dynamic content, it is better to separate the EJB server and the Java servlets from the Web server and putting these two on different machines.

PHP scripts are written into the html file directly. Any scripts found in a html page are forwarded to the module running on the server and executed there. Database connections are ad-hoc. An http-servlet is a java class that can be loaded dynamically by the servlet engine. The

<sup>2</sup>The alternative is bean-managed persistence which returns the responsibility of writing SQL queries to the programmer. This technique is not considered.

JDBC interface is used to communicate with the database explicitly. An EJB server provides a number of services like database access through JDBC, transaction management (JTA) or naming services (JNDI). An EJB server manages a number of EJB containers and generally java servlets are used to call bean methods. The advantage of this architecture is that it separates the application logic from any specific platform or infrastructure and database accesses are generated automatically as part of the bean methods.

## 8.2 Performance Comparison

Emmanuel Cecchet et al. [8] performed a study using 4 different architectures:

- Apache web server and a MySQL database [1] on a separate machine, working with PHP scripts.
- Apache web server and a Tomcat servlet engine on the same machine plus a machine with the MySQL server.
- 3 different machines, one with Apache server, one with a tomcat servlet engine and one with MySQL server.
- 4 different machines: A Apache server, a tomcat servlet engine, a JOnAS EJB server [10] and a MySQL server.

Those 4 architectures are used to perform 2 tests with different web sites. One simulated an online bookstore, which stresses the database, and the other simulates an Auction site, which stresses the Web server.

After finishing the study, the author concludes that the programmability of EJB and Java servlets is higher than of PHP because of available java tools and the safety properties of the java language. EJB is easy to use as it does not require SQL scripts to be written. Nevertheless, using this technique leads to the greatest number of lines of code compared to PHP and normal java servlets. Tools can be used to generate large portions of EJB code automatically.

PHP scripts are more efficient than Java servlets because they introduce a minimum overhead. PHP scripts have the disadvantage that they are tied to the Web server and provide limited functionality and runtime support. If the Web server is a bottleneck, using different machines for the servlet server can offload the web server. If the database is the bottleneck due to database lock contention, the build-in synchronization methods of java can be used to offload some stress from the database leading to performance gains.

EJB servers offers the most flexible architecture and offers many services to the enterprise beans, which capture the application logic. This usually is a trade-off, as this offers some important software engineering qualities such as modularity, portability and maintainability. Performance suffers, though, due to this big architecture introducing some overhead.

## 9 Miscellaneous Studies and Techniques about Enhancing Performance

The following section summarizes some other studies about web server performance and techniques to improve them.

Invariant	Name	Description
1	Success Rate	Approximately 88% of requests to the server result in the successful return of the document
2	File Types	HTML and image files account for 90-100% of requests
3	Mean Transfer Size	Mean transfer size $\leq 21$ kilobytes
4	Distinct Requests	Among all server requests, less than 3% of the requests are for distinct files
5	One Time Referencing	Approximately one-third of the files and bytes referenced in the log are referenced only once in the log
6	Size Distribution	The file size distribution is heavy-tailed; the tail of the distribution is Pareto with $0.40 < \alpha < 0.63$
7	Concentration of References	10% of the files accessed account for 90% of server requests and 90% of the bytes transferred
8	Inter-Reference Times	File inter-reference times are exponentially distributed and independent
9	Remote Requests	Remote sites account for $\geq 70\%$ of the accesses to the server, and $\geq 60\%$ of the bytes transferred
10	Wide Area Usage	Web servers are accessed by 1000's of domains, with 10% of the domains accounting for $\geq 75\%$ of usage

Figure 5: Summary of Invariants Found in Web Server Workloads [4]

## 9.1 Performance enhancements through Invariants in Web Server Workloads

Martin F. Arlitt et. al. [4] present a study about web server workloads and looks for invariants. These invariants are described in Figure 4. Based on these figures, we can see that file caching must have a great potential for improving Web server performance. A large number of references direct to a small number of documents (see 4).

Within this set of documents there is a concentration of references (7), which indicates that only a small number of files need to be cached to have a good cache-hit ratio. The author suggests – due to the fact that the average size of files is rather small – that the size of the cache does not need to be very great. As most of the requested files are only hit once during the duration of the viewed logs (5), caching these documents would lessen the functionality of the cache.

The author argues that even a reasonable sized cache (32 MB) could reduce the data volume transferred by the server by an order of magnitude. The study also tests different replacement policies in web caches and suggests using a Least Frequently Used (LFU) policy with aging to get the best results. In this policy the least frequently used cache entry is deleted from the cache to make room for a new entry. This contradicts the traditional approach of using a Least Recently Used (LRU) policy, where always the least frequently used cache entry is deleted.

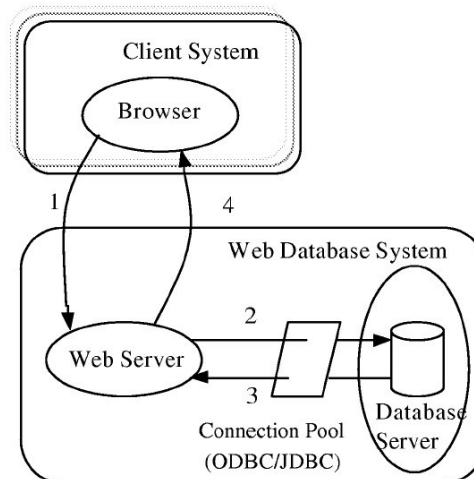


Figure 6: Processing a Web database query at a typical Web database system [23]

## 9.2 Performance Issues of a Web database

Ji Li et. al. [23] introduce the concept of a Web database. A Web database consists of a Web server and a database, seen as a coherent system. Figure 5 describes such a Web database and how a query is normally processed. Such a query is processed in 4 steps: First, (1) the query request is send by a client to web database system via the Internet. (2) Here the request is served by the Web server if it finds a suitable database query and dispatches the request to the database server through an interface. (3) The query is processed by the database server and the result is returned to the Web server. (4) The Web server then uses the result to create a response web page and returns this page the client.

Pooling technology is normally used in a Web database system to get better performance. There are 2 different types of pools. There is the thread pool in Web server and the other is a connection pool in the database server. Thread pools eliminate the overhead of creating a new thread<sup>3</sup> when another request is sent to the Web server. Instead an available thread out of the pool is used. The same applies to the connection pool. The ODBC pool is such a typical connection pool. ODBC is the standard interface to use different databases.

### 9.2.1 Additional Metrics

Traditionally, response time and throughput are the two most important performance metrics for a database and a Web server. Ji Li et. al. [23] suggest using different or more metrics to measure a Web database because of following reasons: If the result of a Web database query is 1 MB, a user may only need to receive the first page to read at first, expecting pages to follow will be available when he needs them. The user or client does not need the whole page first put only a small part first and the rest at times as long as he can read without interruption.

<sup>3</sup>Like thread pooling it is possible to create a Process pool for using processes repeatedly. The author also assumes a Thread based Web server. Different Web server architectures are described in the section 5

There it would be interesting to measure the time the client receives the first response byte from the server (TTFB) and the last byte (TTLB). Another interesting metric is the Throughput Ratio (TR) which indicates how the capacity of the Web server matches the capacity of the database. If their TR is equal to 1, their processing speed match each other and the Web database system can achieve its maximum throughput.

### 9.2.2 Relations between different Web database parameters

Ji Li et. al. [23] performed a study trying to find relations between different Web database parameters. First the Response time was measured against the query result file size. The test showed that when the result file size increases, the response time for queries to the database increases about 7.8 more than the script response time and the rate of increase of the script response time is about 1.5 times bigger than that of HTML. Therefore, one can ignore the size of HTML files but has to be aware of the fact that if query results grow in size the response time is seriously affected.

Second, the table size used from the previous test was greatly increased to see the implication of big tables in a database and its affects to the response time. Although the response time increased, the difference was a linear change due to the efficient implementation of todays databases in querying data.

Third, result files could be separated into different parts and sent to the client divided into several pages. The test tried to find out how much the response time increases if the number of records on a page was increased. For example, if the a page was sent with 15 records, the response time is 18% greater than sending only 8 records. So the number of records sent in a page should be defined by the limit of satisfactory service to the user and necessary response time.

### 9.3 Overuse of HTTP Services

Jim Challenger et. al. [9] discuss the overhead server extensions like SSI, JSP, PHP, Java Servlets or ASP inflict. These extensions can provide very useful services and web application cannot life without them. Nevertheless, some content generated by these services is not really necessary for the user but lead to worse performance. One example is the frequently seen “factoid“ or “thought of the day“, which are trivia or wisdoms chosen at random and presented to the user. Simple static pages that can be served quickly need to be parsed for such trivia and degenerate performance. Web sites that make use of such features should go ahead and test response time without this information to see if performance can be enhanced by eliminating these parts of a page, especially at highly frequented sites.

Very often scripts are sent to the client (e.g. Javascript scripts) but are not used in the content of the page. This may happen because of general headers and footers for each page. Even long embedded comments or blanks can increase the size of a page. Although the size is ignorable for a single page, if the same page is served 1 Million times a minute, 6 kb of unnecessary scripts and HTML code sums up to 100MB of additional bytes per second served unnecessarily.

While using SSL, the programmer has to take care that only the essential information is encrypted. For example, very often embedded images are encrypted too, which can increase the CPU overhead scientifically when a lot of images are transferred. Another examples are buttons or logos. If these objects or other objects that do not change during a session need to be

encrypted, their expiration time needs to be set to a proper value so that the browser can cache them. Generally, not only in SSL encrypted sessions, images should be served from the same URLs so that browser are able to cache them.

Web pages often include menus consisting of several images. As transferring one big images is cheaper than transferring multiple small images, a menu could be served as a single picture while using an image map to determine which item of the menu was clicked. Images seen on Web pages should only contain the necessary information to let them be seen in good quality on the screen. To guarantee that, only a resolution of 75 dpi is needed, which lessens the size of an image. So a proper design of a web page is significant to decrease Web server load as well.

## 10 Conclusion

Performance testing of websites is very important for sites which are going to have a lot of traffic. Although performance testing may cause a lot of work, investments in performance testing will pay in the long run. Important for testing performance is the right characterization of the workload the website will have to be able to master. While existing logs from previous systems can be used to estimate future workload, these logs often do not contain all relevant information. Characteristics of Web user behavior can be used to characterize workloads. Burstyness of WWW traffic has to be taken into account, too. While testing a web application, it is necessary to verify that the testing client is not the bottleneck of the tested system.

There are various ways to enhance server performance. Replication and Load Balancing is a necessity for many websites, but there are various ways to ensure an equal load distribution. Maybe the most efficient solution is to use multiple connection routers which are balanced through a RR-DNS server. Caching can enhance performance significantly, even if the site makes heavy use of dynamic content generation, if the proper techniques are used. Pre-caching of frequently accessed sites and site fragmentation are examples for such techniques. It is important to think about how up-to-date the information for the user of the website has to be. Even tuning the web server to let it send the proper http header values may lead to performance gains in client or server side caching.

There are many available possibilities to include business logic in a website. PHP, for example, is a scripting language used to integrate dynamic data in a html page. While it is rather easy to use, it suffers from the lack of runtime support. Java servlets or EJB servers provide this support but are generally slower. Databases are normally used for generating dynamic content and could easily become a bottleneck of the whole web server system. While table size of the database does not matter for performance issues, query size is relevant. This leads to the idea of sending query results only in small sized parts to the client.

# 11 Bibliography

## References

- [1] MySQL AB. Mysql. <http://www.mysql.com/>.
- [2] Oskar Andreasson. Linux kernel 2.4. <http://ipsysctl-tutorial.frozentux.net/chunkyhtml/tcpvariables.html>.
- [3] Florin Andrei. Optimizing squid for linux 2.4. <http://www.linux-kita.com/data/Optimizing-Squid-for-Linux-2.4-HOWTO.html>.
- [4] M. Arlitt. A performance study of internet web servers, 1996.
- [5] R. Braden. Request for comments 1122. <http://www.faqs.org/rfcs/rfc1122.html>.
- [6] R. Braden. Request for comments 1644. <http://www.rfc-archive.org/getrfc.php?rfc=1644>.
- [7] Maria Calzarossa, Luisa Massari, and Daniele Tessera. Workload characterization issues and methodologies. In *Lecture Notes in Computer Science, 1769 / 2000*, page p. 459. Springer-Verlag Heidelberg, 2000.
- [8] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite1, and Willy Zwaenepoel. Performance comparison of middleware architectures for generating dynamic web content. In *Lecture Notes in Computer Science, 2713 / 2003*, pages pp. 23 – 33. Springer-Verlag Heidelberg, 2003.
- [9] Jim Challenger, Arun Iyengar, Paul Dantzig, Daniel Dias, and Nathaniel Mills. Engineering highly accessed web sites for performance. In *Lecture Notes in Computer Science, 2016 / 2001*, page p. 247. Springer-Verlag Heidelberg, 2001.
- [10] ObjectWeb Consortium. Jonas. <http://jonas.objectweb.org/>.
- [11] Microsoft Corporation. Active server pages. [www.asp.net](http://www.asp.net).
- [12] Microsoft Corporation. Internet information server. <http://www.microsoft.com/windowsserver2003/iis/default.mspx>.
- [13] M. Crovella and A. Bestavros. Explaining world wide web traffic similarity. In *Proceedings of the 1996 SIGMETRICS Conference on the Measurement and Medeling of Computer Systems*, pages 160–169, 1996.
- [14] Marina del Rey. Request for comments 793. <http://www.rfc-archive.org/getrfc?rfc=793>.
- [15] National Center for Supercomputing Applications. Ncsa [httpd](http://hoohoo.ncsa.uiuc.edu/). <http://hoohoo.ncsa.uiuc.edu/>.
- [16] PHP Group. Php. <http://www.php.net/>.
- [17] Amazon.com Inc. [www.amazon.com](http://www.amazon.com).
- [18] Red Hat Inc. The tux webserver. <http://people.redhat.com/ mingo/TUX-patches/>.

- [19] InetDaemon.Com. An explanation of the transport control protocol. <http://www.inetdaemon.com/tutorials/internet/tcp/index.html>.
- [20] Object Technologies International and by Eastman Kodak Company. Jaws adaptive web server. <http://www.dre.vanderbilt.edu/JAWS/>.
- [21] Arun Iyengar, Erich Nahum, Anees Shaikh, and Renu Tewari. Enhancing web performance.
- [22] Alfred Kobsa<sup>1</sup> and Josef Fink<sup>2</sup>. Performance evaluation of user modeling servers under real-world workload conditions. In *Lecture Notes in Computer Science, 2702 / 2003*, pages pp. 143 – 153. Springer-Verlag Heidelberg, 2003.
- [23] Ji Li and Kevin L. Performance issue of a web database. In *Lecture Notes in Computer Science, 1873 / 2000*, page p. 825. Springer-Verlag Heidelberg, 2000.
- [24] Sun Microsystems. <http://java.sun.com/products/ejb/>.
- [25] Sun Microsystems. Java servlets. <http://java.sun.com/products/servlet/>.
- [26] Sun Microsystems. Sun java web server. <http://www.sun.com/software/jwebserver/>.
- [27] David Mosberger and Tai Jin. httpperf: A tool for measuring web server performance. In *First Workshop on Internet Server Performance*, pages 59—67. ACM, June 1998.
- [28] Vivek Pai. Flash web server. <http://www.cs.princeton.edu/~vivek/flash/>.
- [29] The Apache Projekt. The apache webserver. <http://httpd.apache.org/>.
- [30] IBM Research. Afpa web server. <http://www.research.ibm.com/afpa/>.
- [31] Rozanski, Bollman, and Lipman. Sieze the occasion: Usage-based segmentation for internet marketers.
- [32] William Stallings. *Operating System, 4th Edition*. Prentice-Hall, Inc., 2001.
- [33] Zeus Technology. Zeus web server. <http://www.zeus.com/>.
- [34] Filippas I. Vokolos and Elaine J. Weyuker. Performance testing of software systems. In *Workshop on Software and Performance archive*, 1998.
- [35] www.realnworks.com. [http://www.realnworks.com/resources/contentdelivery/server/releasenotes\\_020925](http://www.realnworks.com/resources/contentdelivery/server/releasenotes_020925)
- [36] www.unixguide.net. Solaris 2.4 faq. <http://www.unixguide.net/sun/faq/3.45.shtml>.
- [37] Jun-Li Yuan and Chi-Hung Chi. Web caching performance: How much is lost unwarily? In *Lecture Notes in Computer Science, 2672 / 2003*, pages pp. 22–33. Springer-Verlag Heidelberg, 2003.
- [38] Yuval. lists.balabit.hu mailing lists: tproxy. <https://lists.balabit.hu/pipermail/tproxy/2003-August.txt>.