SE Computational Systems Seminar
SS 2004
June 15, 2004

Flash: An efficient and portable Web server

Vivek S. Pai Peter Druschel Willy Zwaenepoel

Abstract

This document is a survey on the paper "Flash: An efficient and portable Web server", appeared in Proc. of the 1999 Annual Usenix Technical Conference, Monterey, CA, June 1999.
It presents a new Web server architecture called asymmetric multiprocess event-driven architecture, and an implementation of this architecture: the Flash Web server.

Author:
Zenina Huskic (zehuskic@cosy.sbg.ac.at)

Academic Supervisor:
Univ.-Prof. Dr. Christoph Kirsch (ck@cs.uni-salzburg.at)

Correspondence to:
University of Salzburg
Department of Computer Science
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Contents

1 Introduction

The rise of Internet nowadays demands high-performance Web servers which can manage thousands of requests per second. In order to provide such huge throughput rates various Web server architectures have been developed and preferably used in the past years, like: single-process event-driven (SPED) architecture, multi-process (MP) architecture, and multi-threaded (MT) architecture. None of these architectures is perfect, thus each is characterized through some special features as well as drawbacks.

In order to overcome the drawbacks and to take advantage of the features incurred by the previously mentioned approaches a new server architecture was suggested by Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel at the Annual Usenix Technical Conference 1999. This new approach is called asymmetric multi-process event-driven (AMPED) architecture.

The goals which should be achieved by this new architecture are:

- High throughput, i.e. managing thousands of requests per second.
- Good portability, i.e. a Web server can be used on various platforms without larger (or any) modifications.
- Wide range of workloads, i.e. a good behavior on cached workloads as well as on disk-bound workloads for small and large sized files.
- Efficiency, i.e. a Web server has to make its job with as little waste of effort as possible.

In the following sections the basic functionality of a Web server and the problems associated with it will be presented. Afterwards, in the next chapter, previously mentioned approaches to implement a Web server, will be described with regard to their specific advantages and disadvantages.

What is also of importance to mention here is that the paper focuses on serving HTTP requests for static content. The reason is, that the dynamic content is managed in all server architectures in the similar way (an auxiliary application running as a separate process, is called to generate the requested content), thus leaving little space for performance improving.
Furthermore a UNIX-like operating system is being assumed.

1.1 Request Processing Steps

In order to communicate, i.e. to exchange requests and responses the HTTP clients and Web servers use the Transmission Control Protocol (TCP). A client opens a TCP connection to the server, and transmits an HTTP request header that specifies the requested content. By receiving a request the server executes the following steps (see Figure 1):

1. **Accept client connection**
   The server accepts a new connection by performing an `accept` operation on its `listen` socket.
2. **Read request**
   The server reads the HTTP request header and parses it for the requested URL and options.
3. **Find file**
   The server searches its filesystem for the requested file and checks if the client has appropriate permissions to access the file. It also obtains the file's size and last modification time in order to include them in the response header.

4. **Send response header**
   The server transmits the HTTP response header on the client connection's socket. The HTTP response header contains various information about the server itself and about the requested content.

5. **Read file**
   The server reads the file data from the filesystem. This may take several read operations for larger files.

6. **Send data**
   The server sends the requested content on the client connection's socket. As in the case of the "read step" the sending of larger files takes several send operations.

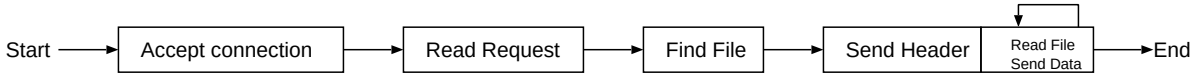

Figure 1: Simplified HTTP Request

1.2 Blocking steps

All of the processing steps involve operations which can eventually block:

- reading data or accepting connections from a socket may block if the expected data has not yet arrived from the client,
- writing to a socket may block if the TCP send buffers are full due to limited network capacity,
- testing a file's validity or opening the file may block until any necessary disk accesses complete,
- reading a file or accessing data from a memory-mapped file area may block while data is read from disk.

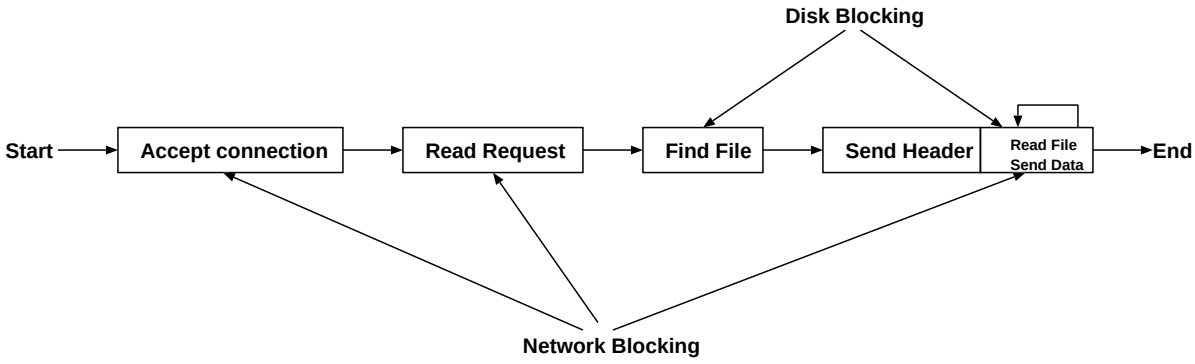

Figure 2: Blocking steps

1.3 How to cope with the blocking steps?

In order to cope with the blocking operations two common techniques have been developed and adapted in the past years:

- Interleaving CPU processing with disk accesses, and network communication, and
- Caching - storing of frequently requested content in the main memory, thus reducing the number of disk accesses.

Different Web server architectures implement these two techniques in different ways, thus the following section deals with the most popular approaches to design a Web server.

2 Server Architectures

In this chapter the existing server architectures as well as the newborn AMPED architecture will be described.

2.1 Multi-process (MP)

In this approach the master server process forks a new process to handle each HTTP connection, as illustrated in Figure 3.
The advantages of this model are:

- Concurrency – multiple processes handle multiple HTTP requests concurrently.
- OS support – for overlapping of disk activity, CPU processing and network connectivity.
- No synchronization – is required since each process has its own private address space.

Of course, this model has some disadvantages too:

- expansion of processes employed to serve the clients requests,
- additional overhead for context switching and interprocess communication (IPC), as well as
- the difficulty to implement some optimizations which are based on global information, like shared cache of valid URLs.

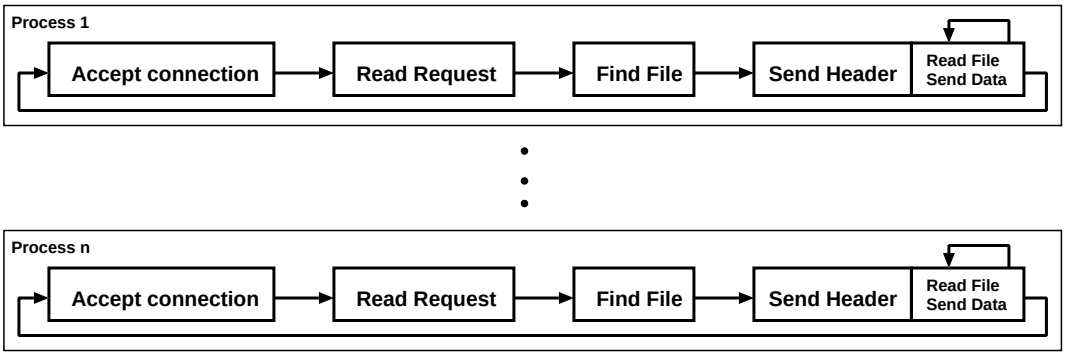

Figure 3: Multi-process model

The most popular Web server which implements this model is the Apache Web server on UNIX operating systems.

2.2 Multi-threaded (MT)

A multi-threaded server employs one single process with multiple threads of execution to manage all HTTP requests. Each thread executes all the steps associated to one request. Figure 4 illustrates the MT architecture. The major advantage of this approach is that all threads can share global variables, what opens the door to the

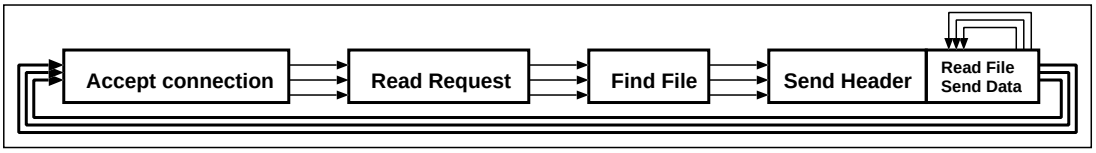

Figure 4: Multi-threaded model

use of optimizations that rely on shared state.
On the other hand this advantage implies the major disadvantage of this model, namely the necessity to coordinate the accesses to the shared data in some way.
Furthermore, the MT model requires that the OS supports kernel-level threads in order to prevent the blocking of a whole process if one of the threads blocks. This requirement makes an MT server less portable and less attractive, since there are still some operating systems widely used (e.g. FreeBSD 2.2.6) which provide only user-level threading without kernel support.
The MT approach is used by the Apache Web server on Windows operating systems.

2.3 Single-Process Event-Driven (SPED)

The SPED architecture uses a single process and non-blocking system calls to manage all HTTP connections concurrently as illustrated in Figure 5.
The main server process consists of the processing of a series of steps associated with the serving of the incoming HTTP requests. At any instant in time, a single step is being processed.
One can imagine the server process as an endless loop through the list of the actions associated with the request handling. In each iteration the server performs a `select` to check for completed I/O events. When an I/O event is completed, the server completes the corresponding basic step and initiates the next step associated with the request. This approach incurs the following advantages over the MP and MT architectures:

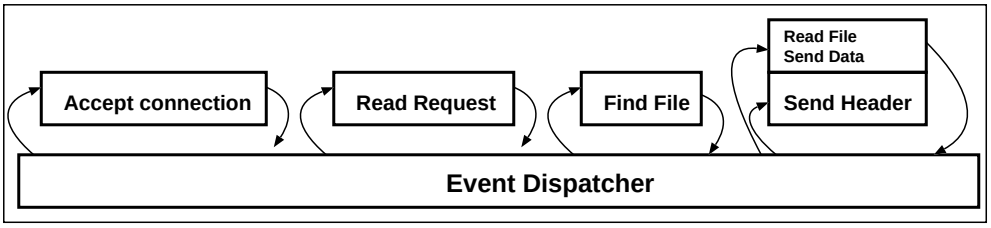

Figure 5: Single-Process Event-Driven model

- single address space, enabling an easy implementation of optimizations based on shared data,
- no context switching required, since only one process is being employed,
- no synchronization required, since the data is accessed and updated only by one process.

The major disadvantage of this approach is that many current available operating systems do not provide appropriate support for asynchronous disk operations. As a consequence, `read` operations on files may still block the main server process while disk I/O is in progress. This leads to some loss in concurrency and performance.
One of the Web servers which implement this architecture is the Zeus Web server, a commercial product of the Zeus Technology Ltd.

2.4 Asymmetric Multiprocess Event Driven (AMPED)

According to the advantages and disadvantages of the previously presented architectures the best way to construct an efficient and portable Web server would be to combine the existing approaches into a new one. This idea stands behind the AMPED approach. Figure 6 depicts this model. An AMPD server works as follows:

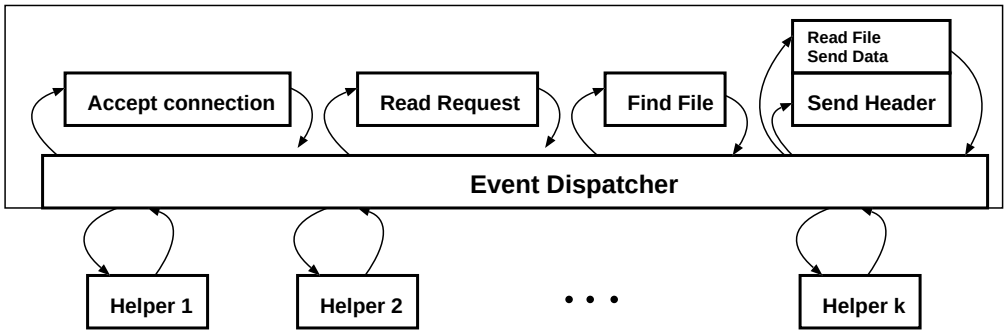

Figure 6: Asymmetric Multiprocess Event Driven model

- by default, the main event-driven process (event dispatcher) handles all processing steps associated with HTTP requests,
- when a file is to be read the main process checks if the requested file is in main memory (by using the `mincore` system call)
  - if the requested file is in main memory the main server process reads the file content from the main memory
  - if the requested file is not in main memory, the main server process instructs a *helper* process via an interprocess communication channel to perform the potentially blocking disk operation and does other work.
    The *helper* process reads the file into the main memory and notifies the main server process when the operation is completed. The server learns of this event like any other I/O completion event via `select`. It maps then the pages from main memory, which contain the requested file into its virtual memory by using the `mmap` system call.

The AMPED architecture combines the event-driven approach with the multiple *helper* processes (or threads) that handle blocking I/O operations. Thus, it incurs the advantages of the MT/MP approach and the advantages of the SPED approach. The only performance problems under this architecture suffers are low overheads due to:

- additional interprocess communication between the main server process and helper processes and
- memory residency checking.

2.5 Design Comparison

As presented in the previous section there are different approaches in implementing Web servers. Each of the approaches has its general advantages as well as disadvantages.
Furthermore each of the designs is characterized through different performance characteristics and each of them incurs either benefit or overhead by introducing some of the possible optimizations.

2.5.1 Performance characteristics

Performance of different server architectures is measured according to the following characteristics:

- Disk operations
  Does a disk activity cause all request processing to stop?
- Memory consumption
  How much memory does a server consume?
  If a server has large memory requirements, then there will be less space for the filesystem cache.
- Disk utilization
  Can server generate several concurrent disk requests in order to benefit from multiple disks and disk head scheduling?

	MP	**MT**	**SPED**	**AMPED**
Disk operations	+ only the process that causes the disk activity is blocked	+ only the thread that causes the disk activity is blocked	- the whole server process blocks during the disk activity	+ only the helper process that handles the disk activity is blocked
Memory consumption	- high memory requirements	+ single process memory requirements plus memory requirements for each thread employed	+ single process memory requirements	+ single process memory requirements plus additional memory for the helper processes
Disk utilization	+ one disk request per process	+ one disk request per thread	- one disk request at a time	+ one disk request per helper

Table 1: Performance characteristics; + desired behavior, - not desired behavior

2.5.2 Cost/Benefits of optimizations & features

The server architecture influences also the profitability of certain Web server optimizations and features:

- Information gathering – gathering of information about recent requests for accounting purposes and performance improvement.
  How much overhead does information gathering produce in different architectures?
- Application-level caching – caching of previously results in order to reduce computation, what further reduces the memory space available for the filesystem cache.
  In what way do the different architectures manage this cache and how much does it cost?
- Long-lived connections
  What does a long-lived connection cost, expressed in resources per connection?

	MP	MT	SPED	AMPED
Information gathering	- requires some form of IPC in order to consolidate data	- requires synchronization on global variables	+ centralized processing →simple information gathering	+ centralized processing →simple information gathering
Application-level caching	- each process may have its own cache	- single cache with synchronization	+ single cache without synchronization	+ single cache without synchronization
Long-lived connections	- process per connection	- thread per connection	+ file descriptor, application-level connection information and some kernel state per connection	+ file descriptor, application-level connection information and some kernel state per connection

Table 2: Overhead of optimizations & features; + low overhead, - relatively high overhead

3 Flash Implementation

In this chapter the Flash Web server, an implementation of the AMPED architecture, presented in Section 2.4 will be described.

The Flash Web server uses a single non-blocking server process and multiple helper processes (see Figure 7.) The server process is responsible for all clients, auxiliary (e.g. CGI-bin) applications and *helper* processes. The helper processes are responsible for all disk operations which may result in blocking. The processes are used instead of threads in order to provide the portability of Flash to operating systems that do not support kernel-level threads. Furthermore, Flash uses various caching techniques and optimizations to maximize its performance.

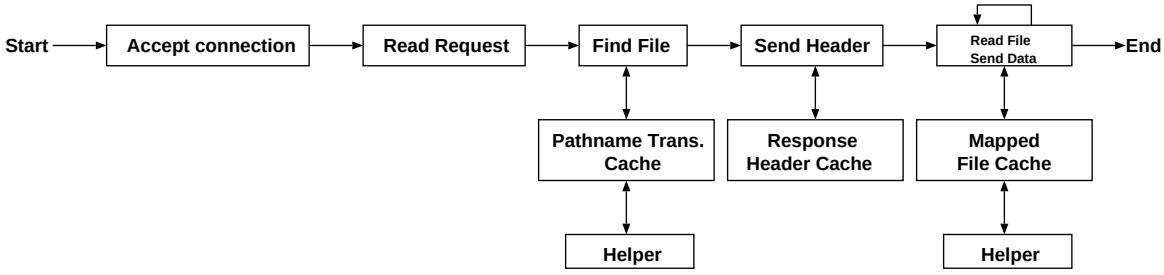

Figure 7: Caches and helper processes in Flash

3.1 Caches in Flash

As can be seen in Figure 7 Flash implements three types of caching:

Pathname translations caching – Caching of mappings between requested filenames and actual files on disk. E.g. "/~ zehuskic" is mapped to "/home/stud2/zehuskic/public_html/index.html"
Reason: avoid using the pathname translation helpers for every incoming request, thus reduce the processing required for pathname translations and the number of translations helpers.

Response header caching – Caching of response headers, that contain various information about the file and the server.
Reason: reuse the cached headers when the same files are repeatedly requested.

Mapped file caching – Caching of memory-mapped files.
Reason: reduce the number of map/unmap operations necessary for request processing, i.e. reduce copying of data from disk to main memory.

3.2 Byte Position Alignment

The Flash Web server uses the `writev` system call (as well as the other Web servers do) to send multiple discontiguous memory regions, i.e. response headers and file data, in one operation.
The problem is that the use of `writev` may cause misaligned data copying within the operating system, what in turn degrades the overall performance of a server. If the size of the HTTP response header is not a multiple of the machine's word size the misalignment occurs when the OS networking code copies the various memory regions (specified in a `writev` operation) into a contiguous kernel buffer. Thus the copying of all following memory regions is misaligned.

Flash solves this problem by aligning all response headers on 32-byte boundaries and padding their lengths to be a multiple of 32 bytes.

4 Conclusion

4.1 A successful story

According to the performance evolution, presented in the paper [2], the goals set to the new architecture have been achieved. An efficient and portable Web server has been developed.
The results of different experiments on synthetic workload, as well as on the realistic one show that the Flash Web server performs as well as (or even better than) the other Web servers. The Flash performance was compared to the performance of two widely-used Web servers: Zeus, a commercial Web server, that implements the single-process event driven architecture, and Apache, a freely-available Web server, that implements the multi-process architecture.
The Flash server's performance exceeds that of the Zeus Web server by up to 30% and it exceeds the performance of Apache by up to 50% on real workloads.
On cached workloads Flash nearly matches the performance of the SPED Web servers.

Better performance of the Flash Web server lies in the combination of the single-process event driven approach and the multi-process approach on one hand, and in the various optimizations (pathname translation caching, response header caching, and mapped file caching, byte position alignment), on the other hand.

4.2 Drawbacks

The major disadvantage (from my point of view) of this new approach is that it makes the things even more complicated than they already are. Both, event-driven programming as well as multi-process (or multi-threaded) programming are difficult enough if used separately. Thus, if used together it makes it much more harder to overview and to control the system in a reasonable and simple manner.
The improvements should be searched in one of the basic architectures.

A Glossary

CGI Common Gateway Interface.

HTTP HyperText Transfer Protocol.

IPC InterProcess Communication.

Kernel-level threads are threads, which are directly supported by the OS: The kernel performs thread creation, scheduling and management in kernel space. Thus, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. [1]

`mincore` system call that determines residency of memory pages.

`select` nonblocking system call that checks whether I/O is possible.

TCP Transmission Control Protocol.

URL Unified Request Locator.

References

[1] Silberschatz / Galvin / Gagne. *Operating System Concepts*. John Wiley & Sons, Inc, 2002.

[2] Vivek S. Pai / Peter Druschel / Willy Zwaenepoel. *Flash: An efficient and portable Web server*. 1999 Annual Usenix Technical Conference, Monterey, CA, June 1999.