Web Caching: a Memory-based Architecture

David Manuel Rodrigues Sora

Departamento de Informática – Universidade do Minho
4710 - 057 Braga, Portugal
david@ipb.pt

Abstract. Caching also applies to Internet content by distributing it to multiple servers that are periodically refreshed. Performance gains can be obtained of two orders of magnitude between the original process-based Web servers and todays threaded servers. AFPA (Adaptive Fast Path Architecture - a software architecture) dramatically increase Web and other network servers' capacity by caching content in RAM, and serving that content as efficiently as possible from the kernel.

1 Introduction

In order to keep the Web attractive, the experienced latencies must be maintained under a tolerable limit. One way to reduce the experienced latencies, the server's load, and the network congestion is to store multiple copies of the same Web documents in geographically dispersed web caches.

Caching [1] is a technique generally aimed at bringing parts of an overall data set closer to its processing site. It is applied for example in memory hierarchies where relatively small on-processor caches yield data access times a magnitude lower than memory-chip access times. The same principle, although on another size and latency scale, also applies to the file-system, where frequently used documents are kept in main memory instead of being slowly retrieved from disk. The Internet expands this memory hierarchy one further step outward from the processor.

Caching popular objects at locations close to the clients (Web caching) has been recognised as one of the effective solutions to alleviate Web service bottlenecks, reduce traffic over the Internet and improve the web scalability.

This communication is organised as follows: Section 2 gives an overview of Web Caching and why it is useful. Section 3 describes the main caching architectures, classifies some performance issues, and to establish a performance baseline, describes the "Adaptive Fast Path Architecture", a platform for building Kernel-mode network servers. The Intel Itanium Architecture appears in this Section as a improved solution aiming for high-performance with its 64-bit processor's features. Finally, Section 4 draws conclusions from the performance analysis and makes recommendations for future work.

2 Overview of Web Caching

Caching on the Web began with the introduction of client-local document stores either implemented as host-local disk backup store or as in-memory caches. The disadvantage of employing this approach alone became obvious as soon as too many different documents appeared on the Web, rendering the formerly known notion of locality set obsolete.

2.1 Functions of a Web Cache

When a Web-user requests specific information, such as a Web site, the cache feature [1] is used to direct the request to the origin server, then return the data back across the Internet. Web caching processes the delivered data and stores that content. When another request is made for that same data, there is no need to send the request all the way through the network; instead the response is sent from the cache memory. This way, duplicated requests are responded to more quickly while also preventing the network from being bogged down by multiple requests for identical information.

A Web cache also keeps track of whether a Web object is fresh or not. If a Web object is no longer fresh or past its expiration date, a Web cache would delete it from its storage or replace it with a newer object from the origin Web server.

Another important feature for caches is the amount of working cache size or amount of space available for storing cacheable Web objects. In most caches, a combination of disk and RAM are used to store Web objects. A cache can serve Web objects stored on its RAM much faster than if the Web objects are stored in the disk drive.

2.2 Web Caching vs. File System Caching

Most file system buffers store fixed-size blocks of data, while Web servers always read and cache entire files. The sizes of Web objects range from a few kilobytes (such as HTML and image files) to extremely large ones such as video files. It is of course possible to break up a large Web file into small pieces and cache just some of the pieces, but to our best knowledge, no research so far has indicated that this is practical and beneficial. Variable-sized Web objects complicate the memory management in Web server caches.

A file system buffer manager always requires that the requested data block be in the cache in order to fulfil a request. Thus, a cache miss on a requested file block invariably results in a cache operation for bringing the requested file block into the cache, and possibly a series of cache replacement operations if it is full. That is, caching of a requested block is obligatory. This is not the case for Web object caching. Unlike file accesses, access patterns observed by Web servers do not usually exhibit high temporal locality. In general, the same user rarely requests the same Web object twice from a Web server in a short time interval, since the requested object should already be in the Web browser cache, the client OS cache, or the proxy cache. Therefore, traditional LRU (Least Recently Used) replacement algorithms that are so popular in file caches are not suitable for Web document caching. On the other hand, multiple accesses from different clients in a short time interval do indicate high popularity of a document. Frequency-based replacement algorithms, such as Least Frequently Used (LFU) and its variants have been shown to be more suitable for Web caching [2,3,4,5,6].

2.3 Why Web Caching?

The most obvious beneficiary of Web caching is the user, who avoids some traffic problems when browsing. The network administrator and the remote Web site also benefit.

Large caches with lots of clients may field as many as 50% of the hits that would otherwise travel through a network individually to the origin site. A typical cache could easily field about 30% of the intended hits, says the NLANR's 1996 research [7]. Thus,

statistically speaking, a Web cache could eliminate at least 30% of the Web traffic that would normally be going out over a wide area network (WAN).

3 Web Performance

Highly accessed Web sites may need to handle peak request rates of over a million hits per minute. Web serving lends itself well to concurrency because transactions from different clients can be handled in parallel. A single Web server can achieve parallelism by multithreading or multitasking between different requests. Additional parallelism and higher throughputs can be achieved by using multiple servers and load balancing requests among the servers.

The Web servers would typically contain replicated content so that a request could be directed to any server in the cluster. For storing static files, one way to share them [8] across multiple servers is to use a distributed file system (DFS). Copies of files may be cached in one or more servers. This approach works fine if the number of Web servers is not too large and data does not change very frequently. For large numbers of servers for which data updates are frequent, distributed file systems can be highly inefficient. Part of the reason for this is the strong consistency model imposed by distributed file systems. Shared file systems require all copies of files to be completely consistent. In order to update a file in one server, all other copies of the file need to be invalidated before the update can take place. These invalidation messages add overhead and latency. At some Web sites, the number of objects updated in temporal proximity to each other can be quite large. During periods of peak updates, the system might fail to perform adequately.

3.1 Caching Architectures

Caches sharing mutual trust may assist each other to increase the hit rate. A caching architecture should provide the paradigm for proxies to cooperate efficiently with each other. Two common approaches to implement a large-scale cache cooperation scheme are hierarchical and distributed caching.

Hierarchical Caching. With *hierarchical caching* caches are placed at different network levels. At the bottom level of the hierarchy there are client caches. When a client cache does not satisfy a request, the request is redirected to the institutional cache. If the document is not present at the institutional level, the request travels to the regional cache, which in turn forwards unsatisfied requests to the national cache. If the document is not present at any cache level, the national cache contacts directly the origin server. When the document is found, either at a cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the request finds the document. There are several problems associated with a caching hierarchy: i) every hierarchy introduces additional delays, ii) higher level caches may become bottlenecks and have long queuing delays, and iii) several copies of the same document are stored at different cache levels.

Distributed Caching. With *distributed caching* no intermediate caches are set up and there are only institutional caches at the edge of the network that cooperate to serve each other's misses. In order to decide from which institutional cache to retrieve a miss document, all institutional caches keep meta-data information about the content of every

other institutional cache. Most of the traffic flows through low network levels, which are less congested and no additional disk space is required at intermediate network levels. In addition, distributed caching allows better load sharing and is more fault tolerant. However, a large-scale deployment of distributed caching may encounter several problems such as high connection times, higher bandwidth usage or administrative issues.

3.2 Performance Issues

The main performance measure is the expected latency to retrieve a Web document. It is debatable that which caching architecture can achieve the optimal performance. A recent research work [9] shows that hierarchical caching has shorter connection times than distributed caching, and hence, placing additional copies at intermediate levels reduces the retrieval latency for small documents.

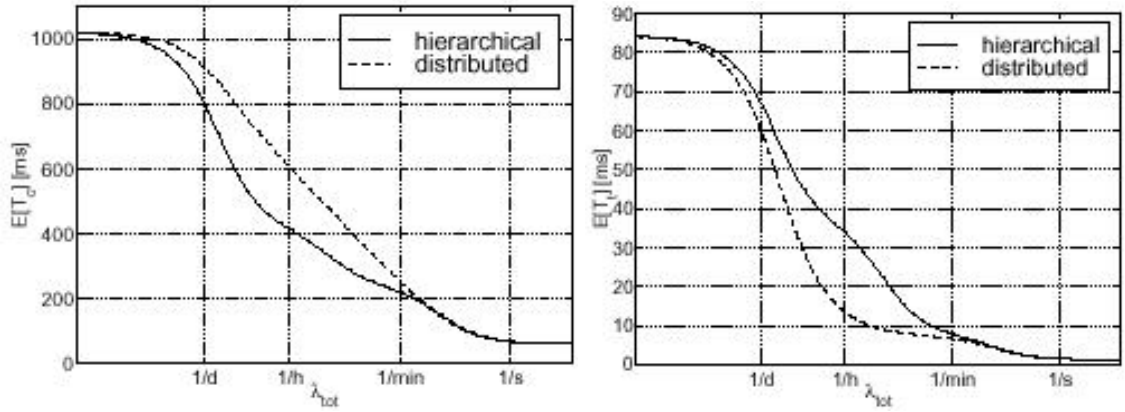

Fig. 1. Expected connection time $E[Tc]$ and expected transmission time $E[Tt]$, for hierarchical and distributed caching. Δ = 24 hours. (Courtesy of [9])

It is also shown that distributed caching has shorter transmission times and higher bandwidth usage than hierarchical caching. A "well configured" hybrid scheme can combine the advantages of both hierarchical and distributed caching, reducing both the connection and the transmission time.

A great number of incoming requests create large numbers of threads or processes, which must all contend for the same system resources. The resultant reduced throughput and slow response times decrease user satisfaction, potentially causing customers to leave.

Commercial and academic Web servers achieved much of the performance gains using new or improved event-notification mechanisms and techniques to eliminate reading and copying data, both of which required new operating system primitives.

More recently, experimental and production Web servers began integrating HTTP processing in the TCP/IP stack and providing zero copy access to a kernel-managed cache. These kernel-mode Web servers improved upon newer user-mode Web servers by a factor of two to six. [10]

Data Copies and Reads. Data copies can be difficult to avoid in user-mode Web servers where the data to be sent resides in the file system cache. In cases where data is already mapped into the user-mode address space, one or more copies will be performed before delivering the data to the network adapter. Even where data copies are eliminated, the additional overhead in reading the data to compute a checksum remains. Providing a

mechanism to send response data directly from the file system cache to the network interface solves the data copy problem. The checksum problem is solved either by precomputing and embedding the checksum in a Web cache object or by relying on network interface hardware to offload the checksum computation.

Event Notification. Event notification can be defined as the queuing of a client request by a server for response by a server task. To handle multiple clients, the server supports concurrency either by assigning a single task, *single process event driven* (SPED), from a pool of tasks to each client request, or by using asynchronous system calls to manage many requests with a few tasks, multiple process/thread (MP).

In the MP model, a server creates a new task for each new request. Because creating a new task can be time consuming, most MP servers reduce the overhead by pre-allocating a pool of tasks. However, pre-allocating a pool of tasks to avoid task creation still incurs unwanted scheduling overhead. Every request requires a reschedule to the task for that request. Ideally, the unnecessary scheduling inherent to the MP model is avoided in a design where a single task services requests on behalf of multiple clients.

In the SPED model, a few processes handle requests from multiple clients concurrently. The SPED model relies on asynchronous notification mechanism for notifying a server task of incoming network requests.

Communication Code Path. A third performance issue is the overall communication code path through the socket layer, TCP/IP stack, link layer, and network interface. The socket layer is not necessarily tailored to the needs of Web servers. Reducing the code path between the Web server and TCP/IP stack allows system calls and redundant socket layer code elimination.

User-mode. These user-mode Web servers rely heavily on the operating system to provide the refined primitives to reduce data movement, limit event notification overhead, and minimise the communication code path. One approach is to optimise existing interfaces and their implementations.

Kernel-mode. Kernel-mode Web servers have been implemented in the context of both production and experimental operating systems. Migration of services considered integral to a server's operation into the kernel is not a new idea. Delivery of static Web responses amounts to sending files on a network interface and does not require extensive request parsing. A kernel-mode Web server can fetch response data from a file system or kernel-managed Web cache. If the kernel-mode caching Web server determines that it cannot serve the request from its cache, it forwards it to a full-featured user-mode Web server. Kernel-mode Web servers can be characterised according to the degree of their integration with the TCP/IP stack and whether responses are derived in a thread or interrupt context.

3.3 AFPA (Adaptive Fast Path Architecture)

AFPA is a Kernel-mode platform for high-performance network servers. The main features are the support for a variety of application protocols; direct integration with the TCP/IP protocol stack (tight integration with the TCP stack enables better event notification and data transfer); and a Kernel-managed, zero copy cache (a zero-copy send interface reduces data transfer by allowing responses to be sent directly from RAM-based cache).

AFPA's central component, an in-Kernel RAM-based cache, is intended to store not only the most frequently requested items, but also the entire set of static content. Serving content from RAM eliminates disk latency and bandwidth constraints.

Memory management. To ensure that sufficient pageable storage remains available for normal system operation, AFPA limits the amount of storage it pins to 25% of real memory [10]. But, it does not prevent the file-system cache manager and virtual memory manager from using more or less than 25% of real storage for the file-system cache.

Real-memory management performed by the file-system cache manager allows AFPA to influence resource management without assuming complete responsibility. However, operating systems may allocate less memory to the file-system cache than is appropriate for a system whose primary purpose is caching.

AFPA explicitly allocates nonpaged RAM for pinned-memory cache objects. The primary questions regarding the implementation of such objects concern allocation and management of pinned storage. No explicit limit is imposed on the amount of pinned memory which AFPA requests. Pinned memory is simply allocated until further allocations are refused. The operating system fails calls for relatively large amounts of pinned memory before the system runs dangerously low. When a call to allocate pinned memory does fail, a file-system cache object is used instead.

Table 1: Web Server Characteristics

	Architecture	Cache	0 copy	Direct TCP
Apache	MP/user	filesystem	no	no
Zeus	SPED/user	filesystem	no	no
IIS	SPED/user	filesytem	yes	no
kHTTPd	SPED/kernel	filesytem	no	no
TUX	SPED/kernel	memory	yes	no
SWC	SPED/kernel	fs or mem	yes	yes
AFPA	Softint/kernel	fs or mem	Yes	yes

Table 1 enumerates and describes the Web servers tested on Linux and Windows 2000. The "architecture" column describes servers as MP, SPED, or *softint* (software interrupt) and kernel-mode or user-mode. The "cache" attribute defines whether the file system, memory, or both back the Web server's cache. The "0 copy" column indicates whether or not the Web server performs a copy to send a cache object. The "direct TCP" column indicates whether or not the Web server is directly integrated with the TCP/IP stack or uses the socket layer.

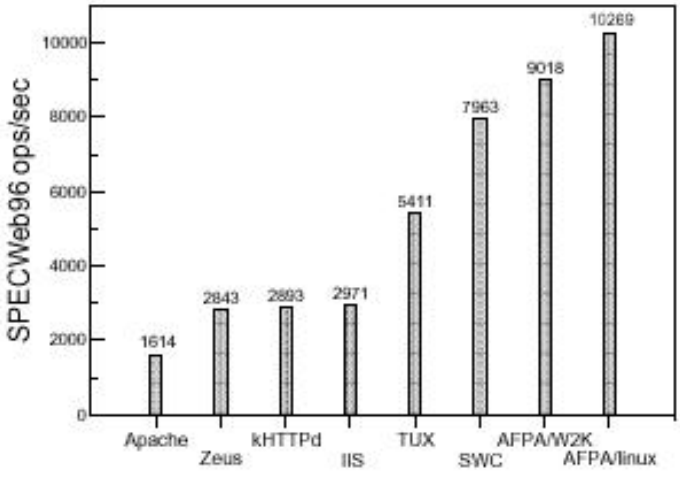

Fig. 2. SPECweb96 workload. (Courtesy of [10])

IBM use the first standard HTTP benchmark for their experiments: SPECweb96 [11]. The SPECweb96 working set comprises files that range in size from 100 bytes to 900 KB.

The results presented in Figure 2 show a significant gap in performance between kernel and user-mode Web cache implementations. AFPA on Linux achieves the fastest performance of the tested servers. The result of 10269 represents more than 1.2 GB/s server throughput. For example, on Windows 2000 AFPA is 50% faster on one CPU (9018 SPECweb operations per second) than IIS on four CPUs (6090 SPECweb operations per second).

3.4 Intel Itanium Architecture

Web server caching handles simultaneous requests from thousands, even millions of demanding customer at the same time. Demand for this kind of performance stimulates new improvements. Intel's new Itanium processors [12] offer another way to meet that demand. The Itanium processor's 64-bit address space, plus the architecture's advanced features, directly attacks the throughput problem in innovative ways.

Table 2: Intel Itanium Processor Benefits for Web Server Caching Technology

Caching Technology	Predication	Speculation	Increased No. of Registers	Parallelism
Web Server Caching	x	x	x	x

Predication. Web server caches are especially prone to heavy branching, and ordinarily this means a branch has to be evaluated before the process can continue, which slows things down.

Some advanced processors use predictive branching, to guess which branch to take and while this works, guessing wrong can stall the processor pipeline badly. Intel uses something called Predication instead, avoiding branching altogether. It processes both branches at the same time, then discards the wrong branch later.

Speculation. Speculation speeds things up by anticipating what code fragments or data bits the CPU might need next, and fetching it from main memory in advance. Other advanced processors have this kind of pre-fetching, but only after the last code branch. The Itanium architecture is under no such limit. In addition, the new generation of intelligent compilers for this architecture are actually capable of scheduling in the compiled code what speculative loads to make in advance, producing more finely tuned speculation results.

Increased Number of Registers. The system stores the intermediate results or code in CPU registers. If it runs out of registers, the intermediate results have to be stored in main memory, then reloaded when needed.

The Itanium architecture offers more registers: 128 integer, 128 floating-point, and 8 branch registers, increasing the overall performance to the end user.

Parallelism. The intelligent compilers being released for this architecture are able to find and exploit opportunities for parallelism. This means the predication and speculation capabilities of the architecture can be fully exercised, providing higher degree of efficiency in operations with this processor family.

4 Conclusions

Web caching is applied in today's Internet to improve the performance of Web accesses by avoiding redundant data transfers. Using new or improved event notification mechanisms and techniques to eliminate reading and copying data, Web servers achieved greater performance gains. IBM and Intel came out with improved solutions for Web caching. The basic difference between the two companies is Intel's focus on products while IBM's focus on basic research. The Intel Itanium architecture provides a set of functionality that should enable larger and more powerful cache server software. An overview of Adaptive Fast Path Architecture from IBM shows that AFPA more than doubles capacity for serving static content compared to conventional server architectures.

However, the gains achieved consider just the static content. It is important to predict the Network evolution and the future needs. The next step for researchers could be to deliver performance improvements for dynamic content while new line of processors with Web Caching features are developed.

References

[1] B. D. Davison.: A Web Caching Primer. IEEE Internet Computing, Vol. 5 (2001) 38-45

[2] Arlitt M, Williamson C.: Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers. Simulations (1997)

[3] Reddy N.: Effectiveness of Caching Policies for a Web Server. Proceedings of the 4th Int. Conf. on High Performance Computing, Cambridge, Massachusetts, USA, (1997)

[4] Robinson J, Devarakonda M.: Data Cache Management Using Frequency-Based Replacement. Proceedings of the 1990 ACM SIGMETRICS Conf. on the Measurement and Modelling of Computer Systems, Boulder, Colorado, USA, (1990)

[5] Williams S, Abrams M.: Removal Policies in Network Caches for World Wide Web Documents. Proceedings of ACM SIGCOMM '96, Stanford University, CA, USA, (1996)

[6] Rizzo L, Vicisano L.: Replacement Policies for a Proxy Cache. UCL-CS Research Note RN/98/13, Department of Computer Science, University College London, (1998)

[7] National Laboratory for Applied Network Research. http://www.nlanr.net/, (2001)

[8] T. T. Kwan, R. E. McGrath, D. A. Reed.: NCSA's World Wide Web Server: Design and Performance. IEEE Computer, November (1995)

[9] P. Rodriguez, C. Spanner, E. W. Biersack.: Web Caching architectures: hierarchical and distributed caching. Proceedings of WCW' 99. (1999)

[10] E. C. Hu, P. A. Joubert, R. B. King, J. D. LaVoie, J. M. Tracey.: Adaptive Fast Path Architecture. IBM J. RES. & DES. Vol. 45 2 March (2001)

[11] The Standard Performance Evaluation Corporation. SPECweb96 Benchmark, http://www.mindcraft.com/whitepapers/openbench1.html , (1999)

[12] Intel Corporation.: The Advantages of IA-64 for Cache Server Software. Version 1.0 (2000)