Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches

by

Roland Peter Wooster

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED:

Advisor: Dr. Marc Abrams

Dr. Edward Fox

Dr. Scott Midkiff

December, 1996

Blacksburg, Virginia

Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches

by

Roland Peter Wooster

Project Advisor: Dr. Marc Abrams

Computer Science

(ABSTRACT)

This thesis investigates the possibility of improving World Wide Web (WWW) proxy cache performance. Most published research on proxy caches is concerned only with improving the cache hit rate. Improving only the hit rate, however, ignores the actual retrieval times experienced by WWW browser users. This research investigates removal algorithms that consider the time to download a file as a factor.

Our experiments show that a removal algorithm that minimizes *only* the download time yields poor results. However, a new algorithm is investigated that does provide improved performance over common removal algorithms using three factors — the speed at which a file is downloaded, the size of the file, and the number of references to the file (the number of hits).

Experiments are conducted with a modified version of the Harvest Cache which has been made available on the Internet from the Virginia Tech Network Research Group's (VT-NRG) home page. WWW traffic from the ".edu" domain is used in all of the experiments. Five different removal algorithms are compared: least recently used, least frequently used, document size, and two new algorithms. The results indicate that the new three factor algorithm reduces the average latency experienced by users.

ACKNOWLEDGEMENTS

I would to like thank Professor Marc Abrams, my advisor, for his hours of advice, proof reading, and direction throughout the research period. I would also like to thank the members of my thesis committee Professors Edward Fox and Scott Midkiff along with many others including Steven Williams, Ghaleb Abdulla, and Dan Aronson at America Online for their valuable comments and suggestions.

Anawat Chankhunthod, at USC, answered my questions related to the software of the Harvest Cache. Tommy Johnson ported my work to the RS6000 and wrote WebJamma to fire accelerated log files at the proxies running in parallel on the RS6000 machine. Andy Wick was always there to help when my machine died and refused to return to life.

The Computer Science department at Virginia Tech funded me throughout my graduate studies. IBM donated, to Virginia Tech, the RS6000 machine used in some of the experiments. Finally, Jennifer, my wife, was always there to give moral support when problems occurred.

To Jennifer, my wife.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The World Wide Web (WWW) has become extremely popular; there are now millions of users using the Internet for WWW traffic. Caching is an effective and cheap method of improving the performance of the WWW. This thesis investigates methods for minimizing the time a user waits for a WWW browser to load pages. This chapter discusses the incredible growth statistics of the WWW. Following this, explanation of what caching can do to improve performance and an outline of the thesis is given.

Unlike a CPU cache, where the blocks are homogeneous in size, a proxy cache contains documents of a widely varying size. Also each of the documents cached may have taken different amounts of time to download. Consequentially simple CPU cache removal policies are inappropriate for proxy caches.

1.1 World Wide Web Growth Statistics

Every research group that has analyzed WWW traffic growth has reported exponential growth [8, 9, 10, 16, 17, 18, 19, 22]. Extensive analysis of the Internet's growth used to be calculated by the National Software Foundation (NSF), however, their studies only include traffic until November 1994, when the Internet backbone was privatized. NSF statistics, as current as they are, are daunting. In November 1992 the total NSFnet backbone data traffic [10] was 28GB per month, and only 0.1% was WWW traffic. By March 1995 the traffic was 200 times greater, at 5.3TB per month with 90% WWW traffic. The actual WWW traffic on the NSFnet backbone increased from 39MB to 4.8TB during this period. This is an increase of 125,000 times, equal to a doubling every 50 days.

The number of connections, where a connection is a single TCP request for a URL, are also increasing exponentially. The National Center for Supercomputing Applications (NCSA) reported [9] 92,000 connections to its systems for the week ending July 3, 1993. Their peak reported load was 4,100,000 for the week ending March 4, 1995. This is a doubling of load every 110 days. The doubling period is actually significantly shorter than 110 days because the March 4th figure is a peak and not a point on an exponential curve. For an unknown reason the number of connections beyond this point did not exponentially increase, but decreased in the last three months of the data to between 2 and 2.5 million per week. The drop might be due to users switching to Netscape browsers, therefore decreasing the number of hits to the Mosaic home page. The University of Melbourne [22] reported an even faster rate of increase of HTTP connections to their servers, doubling in quantity every 38 days for a six month period. However, their sample size, 83,000 per week is small compared to that of NCSA's.

1.2 Why Save Bandwidth, Network Resources, and Reduce Server Load?

It should be clear that the bandwidth of the Internet can not be increased at the same rate as the growth in demand being placed upon it. NCSA's servers have crashed many times due to excessive load [16], due to exhaustion of resources in the operating system.

When an HTTP page is requested the cost to the network is the product of the number of bytes and the hop count. The hop count is the number of network segments between client and server [27]. It is also important to consider what fraction of the bandwidth of a link is used by a connection. There are two methods to reduce the bandwidth cost, either compress the data or increase the locality of the data to the client.

1.3 Methods to Save Resources

The number of Internet users continues to increase, especially with the efforts of companies such as America Online (AOL) and Compuserve. Both of these companies have

introduced millions of subscribers to the WWW. If the demand and supply from all the WWW users was equally[1] balanced then the Internet would probably have little need for analysis. The solution would be to increase the bandwidth of the Internet and use compression on all documents. The load, however, is not equally distributed; there are particular servers and documents that are in very high demand and others in low demand. This yields many possibilities for caching. Changes in the protocols used by WWW browsers could also improve performance.

1.3.1 Compression

Not all the documents on the Internet are compressed. Studies of the data transferred on the NSFnet were performed which indicated that 31% of FTP data was not compressed [8]. These studies were performed, during September 1992, at a time when FTP traffic used the greatest percentage of the NSFnet's bandwidth. FTP is now only a small percentage of the traffic. HTTP is now the largest percentage of the traffic on the Internet, and most HTTP text documents are not compressed. Thus most Internet traffic is uncompressed. It is quite likely that reducing the quantity of data transferred by storing compressed documents on WWW servers and decompressing the documents on receipt would yield a user perceived improvement. This improvement would be possible because most clients using a WWW browser have many idle CPU cycles when they are simply waiting for data to arrive; currently the bandwidth is the bottleneck. These spare CPU cycles could be used to decompress incoming compressed documents, which would arrive more rapidly than uncompressed documents.

[1]The notion of "equally balanced", where all network servers would be equally stressed and where all communication channels were also equally stressed, is impossible to achieve. The term "stressed" is the fraction representing current load over maximum load.

1.3.2 HTTP Protocol Changes

The HTTP protocol, layered over TCP/IP, is costly in terms of both number of TCP connections and the absorption of a network's bandwidth. The term "absorption" is used to reinforce the idea that a significant amount of network bandwidth is used for headers rather than data. Tests have shown that 95% of the data of some HTTP requests is redundant [25]. The problem with the HTTP/1.0 protocol is that it handles connection negotiation on a per URL basis. This means that for each image within a WWW page a TCP connection is made, rather than one connection for the whole page. To exaggerate the problem HTTP sends every possible option supported by the client in each new connection request packet. Many new protocols [17, 20, 25] have been developed to solve one or both of the above problems.

HTTP-NG [25] and HTTP/1.1 [11] use long lived connections. Thus a TCP connection can serve all the images and documents within a WWW page, rather than making multiple TCP connections for a single WWW page containing multiple images or documents. Not only does this reduce the number of TCP connections at the server and client, reducing the use of scarce resources on the server, but it also avoids the slow-startup delay for all but the first file. The slow-startup delay is due to the TCP slow start protocol and the connection negotiation. TCP was designed decades before the WWW; it was optimized for large data transfers hence the slow start protocol. However, most HTTP transfers are small and the delays introduced by the slow start protocol are a significant fraction of the total download time. HTTP-NG and HTTP/1.1 also attempt to avoid sending the vast array of redundant data in each HTTP request. The initialization data is only sent in the first request; the server stores this information and associates it with the TCP connection. Successive requests to that server need only send option information if the options are changed by the client.

Simon Spero [25] proposes a predictive form of HTTP with his HTTP-NG. Although prediction would reduce average response time for the client in theory, this idea may lead

to significantly more inefficient use of network resources. As soon as too many people start to use it the increase in traffic on the Internet would reduce performance to an even greater extent. The problem it would create is each HTTP-NG browser would make requests for all the "predicted" WWW pages. Many of the pages would be wrong predictions, and thus significant bandwidth would be wasted.

Only a few ideas of HTTP-NG were incorporated into HTTP/1.1, other HTTP-NG ideas may be ignored because they require modifications to either server or client software or both, rather than being transparent to the client and server. Unless Netscape and Mosaic include these new protocols in their client software it will be difficult if not impossible for these protocols to become of significant use. HTTP/1.1 addresses many of the transparency problems because it allows the client and server to the use the lowest common denominator: 1.0 or 1.1.

1.3.3 Locality of Data: Caching

The transparency, to the system, of any modification is essential for the success of any performance improving scheme for the WWW. Data caching exemplifies exactly this property. As mentioned before, the cost of an HTTP retrieval, and its time to display, depends on its byte count, available bandwidth of the links, and the number of links traversed. Caching copies of documents from the origin servers to machines closer to clients can reduce the distance significantly. In addition to reducing the retrieval latency a cache will also reduce the network load. This saves bandwidth on all the networks between the cache and the origin server indicating that it is better to position the cache as close to the client as possible. Unfortunately there are disadvantages to caching data [29]; depending on the system the cache can become a single point of failure. It can also serve out of date documents to the user and can not cache dynamic documents.

There are three locations for caches: at the client, at the server, and at the network. These caching methods are discussed in section 2.1. Significantly different cache hit-rates have been achieved by different groups. Equation (1.1) shows how the hit rate is calculated.

Equation 1.2 shows the calculation for weighted hit rate which is frequently used in this thesis. Hit rates consider only the number of URLs whereas weighted hit rates consider only the number of bytes transferred:

$$\text{HR} = \frac{\text{Number of URLs loaded from the cache}}{\text{Total number of URLs accessed}} \tag{1.1}$$

$$\text{WHR} = \frac{\text{Number of bytes loaded from the cache}}{\text{Total number of bytes accessed}}. \tag{1.2}$$

1.4 Problem Statement

Almost all published research on proxy caching, except one recent paper by Bolot and Hoschka [5], is devoted to maximization of hit rate or weighted hit rate. However, this may not yield the optimal response time. There is no benefit in caching local documents which may, in some instances, be retrieved faster from the origin server than from the proxy server. A local file is defined as one on a machine in the same domain as the cache; an example of a domain is "cs.vt.edu".

One objective of this thesis is to modify the Harvest Cache (ver1.4pl3) to offer a selection of removal algorithms. The modified cache is then experimentally tested to compare leading algorithms in the literature (LFU, SIZE), a popular algorithm in implementation (LRU), and two new algorithms developed in this thesis. Finally we aim to distribute the modifications so that anyone can use them.

1.5 Organization

The thesis follows the following organization. Chapters 2, 3 and section 4.1 are literature reviews. Chapter 2 discusses issues related to caching, such as the location of the cache; the different locations of caches are described. Chapter 3 discusses the issues relevant to removal algorithms for caches, such as which protocols to cache, object lifetimes, uncachable

documents, periodic trends, and one versus two way caching. Chapter 4 discusses the removal algorithms investigated in this thesis. Chapter 5 describes the system requirements, experimental methodology, performance variables, and methods of analysis. Chapter 6 describes the results of the initial experiment used to determine configuration parameters for the cache. Chapters 7, 8, and 9 describe the three experiments of the thesis. Chapter 10 concludes the thesis and describes possible areas of further study. Appendix A is the glossary of acronyms used throughout the thesis.

Chapter 2

Related Work

This chapter reviews the rapidly evolving work that is related to the thesis objectives. Five years ago it would have been difficult to find any published research on Internet or network traffic caching, let alone caching specific to HTTP traffic.

2.1 Location of the Cache

There are three logical locations for caches. One of the three, network caching, is the subject of this thesis. A brief synopsis of the other two, client and server caching, is included in this section. If the WWW is to continue growing "both clients and servers must aggressively exploit caching" [18].

2.1.1 Client Caches

All recent WWW browsers offer non-persistent and possibly persistent caching. The former type only stores recent documents in memory during the browser session in which they are loaded. In the latter type WWW pages are stored to disk ready for use in subsequent sessions. Most recent browsers use both types of caching.

For example, Netscape uses a default disk cache size of 5MB for MS Windows and UNIX; this is adjustable by the user. If this cache does not satisfy the request (a cache miss) then the request is forwarded to the network and onto the server. A study completed by O'Callaghan [22] at the University of Melbourne in Australia found that only 6.5% of the requests actually hit the client cache. This small hit rate could probably be improved significantly by increasing the size of the cache, but this is at the expense of disk space.

2.1.2 Server Caches

In contrast to caching at the client there may seem to be no obvious reason to cache at the server. Server caching does not reduce the transmission distance across the network so why cache at the server? The reason is simple: To reduce the load on the origin server by sharing it with a second cache machine. Not only is the Internet overloaded but so are many of the servers placed upon it. Some servers receive up to a million requests per day [21]. A single machine may struggle to perform this many connections, negotiations, and replies to HTTP requests.

The best example of a server cache is the NCSA server [4, 16, 17, 18], which in August 1995 consisted of eight workstations. NCSA's solution to an over-burdened single server was initially to increase the performance of the server; when this again became over-burdened they developed a new architecture for a high performance WWW server. Their server works with the Andrew File System (AFS) as its distributed file system [16]. NCSA's entire local network is served by three Sparc 10 servers, each with 120GB of disk space. To provide caching for remote HTTP requests they use a cluster of Hewlett-Packard (HP735) workstations which operate in a round robin fashion. Each server cache has 96MB of main memory and uses 130MB of disk space for its AFS cache.

The strength of the AFS system is that it is fault tolerant. Also it keeps documents in the caches consistent with the servers. All the cache servers are connected on a single FDDI ring, which was analyzed as being only 4% to 8% utilized. An arriving request is served by one of the cache servers. If none of them contains the requested file then the request is passed to the origin server. NCSA use a rotating DNS list to map requests to their server to one of the caches. If one of the machines crashes the system gracefully degrades its performance. However, for the machines that have set their DNS to the IP address of the crashed cache the performance degradation is not graceful — a timeout occurs.

It was found that the 50 most requested documents only required 600KB of memory to cache and that they satisfied 70% of all the requests [18]. To increase the hit rate to 95%,

800 documents had to be kept in each of the cache servers memories, requiring 60MB of memory. Although the hit rate is 95%, the weighted hit rate is only 80%. This indicates that the cache misses are for larger rarely selected documents. An unfortunate statistic is that the diversity of the selection of requested documents is increasing at a steady rate of 15% per month. Diversity indicates the range of documents selected. It is a measure of the number of different documents accessed within a specified number of URL requests. This will lead to either reduced hit rates or a requirement for increasing cache server memory sizes and disk space.

2.1.3 Proxy Caches

Luotonen and Altis claim that caching within the network "is more effective on the proxy server than on each client" [19]. It should be clear that the more disk space that is available for a cache the more effective it will be. There is, however, no advantage in a cache size larger than the maximum space needed to cache all the WWW documents accessed by the clients in the proxy cache group. If in the client caching system each client uses 5MB of disk space then 200 users will use 1GB of disk space. One study has shown that about 17% of the space is wasted by caching multiple copies of the same file [1]. In the above example 170MB would have been wasted. If this 1GB of disk space is used for a shared proxy cache that all the clients can access, the hit rate will increase enormously, at little cost in latency (assuming the network connecting the client to the proxy is lightly loaded).

Research groups [2, 19] have shown that 30% to 50% hit rates can be achieved by a proxy cache. This not only reduces latency for the clients, but significantly reduces the load on the network beyond the proxy [1]. The reason the server cache's hit rate is higher than a proxy cache's is simple; the server already has a hit on half of the {server, document} pair specified by a URL, whereas the proxy cache must hit on both of the pair to yield an actual hit. The next section compares the performance attainable from the original CERN proxy cache and a new high performance proxy cache.

2.2 Proxy Design

The CERN proxy cache, being the first to be widely deployed, is old and slow when compared to newer proxies such as the Harvest Cache [6]. Many research groups have analyzed the performance difference between proxies; all of them come to the same conclusion that the CERN cache is slow and inefficient. The "CERN proxy software can probably handle a sustained load of no more than 5 connections per second" [22].

The CERN proxy uses the UNIX *fork()* procedure for every proxy request. The forked process terminates after the HTTP request has been satisfied, either via a cache hit or from the actual server. The UNIX *fork()* procedure is very expensive, which is the primary reason for the poor performance of this proxy. "The operating system of a busy proxy site will spend too much time spawning new processes and not enough time relaying data" [29]. Studies have shown that using the CERN proxy will slow down a cache miss by an average of an extra 69% over the average direct latency[1] [20].

Collaborative research groups [6, 29] at the University of Colorado at Boulder and the University of Southern California have created a number of vastly improved caching proxies. The most significant of these is the Harvest Cache. In tests[2] the Harvest Cache served cache hits with a median time of 20 ms; this compares rather favorably with CERN's median of 280 ms. The average performance for hits was even more significant. CERN's average hit response time was 840 ms and Harvest's was a mere 27 ms.

The Harvest implementation uses a threaded architecture. Thus processes do not have to be continually forked and terminated with every HTTP request. This enables the Harvest Cache to serve up to 200 "small" requests per second on a Sparc 20 model 61, as compared to the maximum of 5 requests per second for the CERN cache [6]. Its method of document removal is discussed in section 4.3. The next chapter considers issues that must

[1] Direct latency is taken to be the time it would take to retrieve the HTTP request from the origin server without an intervening proxy.

[2] "Ten clients concurrently referenced 2,000 unique objects of various sizes, types, and Internet locations against an initially empty cache. A total of 2000 objects were faulted into the cache in this way. Once the cache was warm, all ten clients concurrently referenced all 2000 objects, in random order" [6].

be considered in the design of a proxy cache.

2.3 Summary

Of the three caching types, client, proxy, and server, only proxy caching is investigated in this thesis. The assumption is made that client caching and server caching are used in addition to proxy caching. The investigation uses a modified version of the Harvest Cache. The reason for choosing the Harvest Cache over the CERN cache is performance. The Harvest Cache operates at least an order of magnitude faster than the CERN cache.

Chapter 3

Caching Issues

This chapter discusses which protocols to cache, the types of documents to cache, object lifetimes, uncachable documents, periodic trends, and one versus two way caching.

3.1 Which Protocols — HTTP, FTP, and Gopher — to Cache

O'Callaghan's research [22] shows that using a shared cache space to store HTTP, FTP, and Gopher does not maximize cache hit ratio for a fixed disk size. His research consisted of caching all documents accessed by the central proxy cache for the University of Melbourne for a Time To Live (TTL) period of two months, or until space was insufficient. TTL is the maximum period for which a file is cached; when it has been in the cache for the TTL period it is removed. At this point the least recently used (LRU) document would be removed until enough space allowed the new document to be cached. He found that the weighted hit rate for HTTP was 27.3% whereas for Gopher and FTP it was only 6.2% and 5.5%, respectively. The average weighted hit rate was 22% and the average actual hit rate was 31%. O'Callaghan modified the cache removal algorithm by reducing the TTL to one week for FTP, leaving the TTL for both Gopher and HTTP at two months. The average hit rate was significantly improved for what appears to be a minor modification to the cache removal algorithm. The hit rate increased to 44% and the weighted hit rate to 32%. This significant increase in the hit rate may be due to one or two reasons: The first is that the choice of TTL has actually improved the hit rate, the second is that the effect of the variation[1] in the workload has increased the hit rate. This thesis only investigates

[1]The users in the second test may have simply referenced the same documents more frequently, thus increasing the hit rate.

HTTP caching.

3.2 File Type: Text, Image, Audio, and Video

A file's type has some correlation with its size. Video and audio documents are often many orders of magnitude larger than text and image documents. It may be necessary to limit the maximum size of cached documents. A 1GB proxy server will probably contain tens of thousands of documents, even close to a million documents if they are all small text or image documents. Many thousands of these documents would be removed to cache a single high quality audio file (e.g., six minutes of CD quality audio requires 60MB of space). A high quality video file would be much larger. Each of these would consume a significant proportion of the proxy's disk space. Quite obviously the hit rate will be significantly reduced if there are only a few huge documents in the cache. If, however, these audio or video documents are used a large number of times then the weighted hit rate will be extraordinary. This, however, does not often happen [2, 31]. The proxy server will require different, maximum cachable file size settings depending upon whether the weighted hit rate or the request hit rate is more important. In the workloads[2] used by the VT-NRG group less than 0.01% of files were over 4MB [2, 31]. This insignificantly small percentage of files allows a maximum file size of 4MB to be used for "cachable" files in the Harvest Cache.

Table 3.1 illustrates the enormous difference between the percentage of requests and percentage of bytes transferred in one workload at NCSA. Fifty-two percent of the requests are for text, however these only comprise 32% of the bytes transferred. Video possesses the opposite trait; only 0.35% of the requests are for video but a huge 16.5% of the bytes transferred are video.

The VT-NRG Group found that for some of their workloads almost none of the actual references were for video, but between 26% and 39% of the bytes transferred were video documents [31]. Although not typical, a similar trait for audio was exhibited in one of their

[2]The workloads used were the "local clients" and the "graduate clients".

Table 3.1: File Type Characteristics (September 7, 1994) NCSA's Server

File Type	Number of Requests	Percentage of Requests	Bytes Transferred (MB)	Percentage of Bytes Transferred
text	254487	52.03	2515.76	32.20
image	226750	46.36	2849.24	36.47
audio	4114	0.84	885.19	11.33
video	1692	0.35	1289.27	16.50
other	2064	0.43	273.73	3.50

workloads, where audio represented as much as 88% of the bytes transferred, but only 3% of the requests! File type is not only correlated to size, but also with how frequently it is updated; the next section discusses this.

3.3 Object Lifetimes - Expires and TTL

A further problem for caching is that some of the documents in the cache will be out of date. This occurs when the origin server's copy is updated, but not the cached copy. A response to an HTTP request includes metadata about the object, such as an "Expires" field and a "last–modified" field. FTP and Gopher do not provide this information, making the choice of TTL much harder.

Wessels [29] found that in one workload only six out of 28,000 HTTP replies included a complete "Expires" field. Thus the HTTP "last–modified" header field is what most cache servers use to estimate the TTL. A proxy cache stores the "last–modified" value with the object. When a request is made for an object contained within the cache server it will check to see if it has an "Expires" field. If the "Expires" field has not expired the cache server will forward the object to the client. If the "Expires" field is absent a "get–if–modified" request is sent rather than a "get". The "last–modified" value of the cached copy is sent within the request. If the header data returned with the document does not include a "last–modified" field then the cache server will use a default TTL.

The Harvest cache [6], other proxies, and HTTP/1.1 [11] significantly reduce the number

of requests because they do not check with the server on every retrieval. Occasionally stale documents will be sent to the client. A stale file is one which is cached but has more recently been modified at the server. The Harvest cache will first adhere to the "Expires" field, if it is present. If the "Expires" field is not present then it will set the TTL to 20%[3] of the time elapsed since the object's last time of modification when it was cached. The choice of 20% is called the staleness factor; it is discussed in section 3.4. If neither the "Expires" field nor the last modified date is known a default TTL time is set.

The lifetime of a document is the period between successive modifications to the document. The Harvest group found that the mean lifetime[4] of all HTML objects is 44 days. Each different type (text, audio, and video) of file is updated with a different mean time. For example, text documents are updated on average every 75 days, and images every 107 days. It also was found that 28% of objects were updated more frequently than once every 10 days, and that 1% were dynamically updated. This indicates that a global TTL is difficult to set without either serving many stale documents or making many unnecessary server requests.

The problem with the Lagoon cache [24] is that it does not use the "Expires" or "Last–Modified" fields, but relies upon a statically defined TTL. Different TTL's can be set for different types of documents or different domains. There is, however, no way of telling the difference between the GIF photograph of someone on their WWW page, which almost never changes, and a dynamically created GIF weather map. Clearly a fixed TTL is not appropriate for weather maps or many other situations.

3.4 Staleness

HTTP/1.1 [11] and the proxies by CERN [12] and Harvest [6] calculate the TTL to be a fraction of the "last-modified" date. It has been reported that a default TTL set to

[3]The value is user definable; HTTP/1.1 recommends 10%.

[4]The measurements were made on 4,600 URLs over 2,000 Internet servers, both local and remote, for a period of 3 months.

between 5 and 7 days will return approximately 20% [6, 12] stale documents. Wessels [29] states that twenty percent is considered unacceptable. He also states: " ... most Internet users would probably find a staleness factor of 10% acceptable." His method of calculation of the staleness is as shown in equation (3.1).

$$\text{staleness} = \frac{\text{now - last update}}{\text{last update - last modification}}. \tag{3.1}$$

Thus an object that has not been modified for 30 days at the time at which it was cached will be 10% stale after it has been in the cache for three days. Gwertzman and Seltzer [12] took 5% staleness as their goal for the development of their Alex server, stating:

"The Alex protocol provides the best of all worlds in that it can be tuned to:

- reduce network bandwidth consumption by an order of magnitude over an invalidation protocol[5],
- produce a stale rate of less than 5%, and
- produce a server load comparable to, or less than that of an invalidation protocol with much less book–keeping."

Obviously the TTL is only used to try to keep the file up-to-date; if the size of the cache is insufficient to store all the requested documents, as is usually the case, a more aggressive removal policy is required.

3.5 Uncachable Documents: CGI, Dynamic, and PPV

As the WWW evolves, more pages are being dynamically created or operate through a Common Gateway Interface (CGI). CGI documents are created dynamically from database searches for a set of user defined queries. Every reply may be different, even a reply to the

[5]In an invalidation system the cache always checks with the actual server to see if the cached file is the most up-to-date copy of the file; thus the cache never serves stale documents.

same query may be different because the database on the server may have been updated or it may be time dependent. No research seems to agree exactly on what fraction of documents are of this type. Gwertzman *et al.* [12] mention 10% of the documents as a typical figure. The VT-NRG Group [30] found the maximum percentage in any of their workloads was 40% of the documents. What is agreed upon, however, is that this value is continually increasing; this is a significant concern to the designers of caching systems as these documents can not be cached as easily, if at all, as normal URL documents. This problem would be significantly reduced if proxies could rely on the "Expires" field being correctly used, but this is unfortunately not the case.

Another concern is that some servers contain time critical data, such as weather, or stock prices. This can be solved by setting the cache's staleness value to a small fraction; thus the data will never be out of date for too long. Using a staleness value of 10%, a page that is updated every 30 minutes can only be cached, while stale, for a maximum of three minutes.

As the WWW expands, especially in the commercial arena, some companies will be offering Pay Per View (PPV) pages, with films, copyrighted documents, videos, or time sensitive data on line. Obviously, there will be legal issues with copyright if this type of data is cached, because a company could serve a cached copy to all of its staff. The question is whether the PPV is per person, or per request. If it is per person the cache would have to inform the origin server of the number of cache hits it has received so that the correct billing could be made, otherwise the company would be infringing on copyright laws. If it is PPV on a per-download basis then the company would not be acting illegally. In fact the company would save itself time and a significant amount of money, which over time could perhaps pay for the proxy server itself!

One growing commercial issue is that many organizations' funding depends upon the number of times their WWW page is hit [29], advertising being one key example. If a page is cached, then not all the cache hits will be recorded by the actual server, and consequently the server will be deemed far less popular than it really is. Solutions to this problem are

being discussed. One suggestion is for a cache to maintain a count of the number of hits and to increment the origin server with the number of hits received on a regular basis [28].

The proxy will not attempt to cache dynamic files. It determines whether a file is dynamic by looking for certain characters or character strings in the URL. Files containing the following: "cgi", "bin", "pl", "?", and "map" are considered dynamic.

3.6 Day of the Week

Pitkow and Recker [23] found that the hit rate of a *server* cache containing all the documents accessed within the last 24 hours could be increased by using a five day week rather than a seven day week. When using a seven day week the mean hit rate was 67%. When the cache was switched off at the weekend, so that only weekdays are used, (thus the day before Monday is taken to be the previous Friday) the cache hit rate increased to 69%. This would indicate that there is possibly a pattern of documents that are accessed in the week days, and a different pattern at the weekend. If it was possible to dump the Friday cache to tape and upload it for Monday, and likewise dump Sunday to tape and upload it Friday night ready for the next weekend then the average hit rate could possibly be improved yet further. The VT-NRG Group, however, found that there was no significant correlation with the day of the week and the hit rate achieved by a *proxy* cache [31].

Other studies [18] have shown that certain types of documents are more likely to be loaded in the daytime and others at nighttime. Again it might be worth researching this phenomena, using a tape drive to up-load and download documents at an appropriate time.

3.7 Periodic vs. On-demand Removal

There are two methods of cache removal. The first is the on–demand removal policy, as advocated by the VT-NRG Group [31] for proxies. The second as, advocated by Pitkow and Recker [23] for servers, is periodic removal. In the periodic removal system the cache is periodically emptied to a comfort level, such as 80% of the disk space. This has attractive

merits: If the algorithm to determine which documents to remove is complex it is clearly more efficient to calculate all the documents to remove simultaneously and remove them until the comfort level is reached. There is, however, a serious drawback overlooked by this system: the cache is never full. Thus a great deal of disk space is wasted which could be caching documents that may be generating cache hits.

The VT-NRG Group [31] argue that SIZE is the best key; a sorted list can be kept, maintaining at the head of the list the largest file. Using a sorted list, no extra load will be exhibited by an "on–demand" algorithm than a periodic removal algorithm, and a higher hit rate will result because the cache will always be full.

3.8 One Way versus Two Way Caching

One way caching is defined as caching documents that are requested in only one direction; only the documents requested by the clients on a campus or business network are cached in the proxy. In a two way caching system the documents that are requested *from* the local servers by remote clients are also cached. The questions that arise are: "Why would a business or University be interested in two way caching?" and "Why would they be interested in paying for disk space to cache for other people?" The reason is simple: It is analogous to using a server cache. If a network cache server caches in both directions the load on the subnet past the network cache will be reduced. Thus the campus or business subnet will experience reduced load, increasing the performance of the network for everyone within the subnet. Also, the servers on the subnet will experience a lighter load and be able to perform better for the requests that they do receive.

One study has shown that a weighted hit rate of an amazing 95% can be achieved for caching the incoming traffic from clients outside a campus site [31]. Clearly, this reduces the load on the campus or business subnet by a factor of 20, which is quite extraordinary. Although the researchers claim that this is not typical, their test was for a period of 57 days, during which 227,210 documents were requested. Thus, the sample is certainly large

enough to warrant attention. The sample would appear to be representative of a *very* popular server within a campus or business network. High hit rates are often achieved by popular local documents; it may however be inefficient to cache local documents. This is discussed in section 4.3.4

3.9 Negative Caching

WWW browser average latency is greatly reduced by documents that are not available because of server failures and name changes. A solution by the Harvest group [6] is that of "negative–caching." This is where the cost of repeated failures is reduced to that of a cache hit. The first time a request is made to a document that does not exist, or server that is not going to reply, may take up to 75 seconds before the client will time-out. The Harvest cache will store the fact that the document is unattainable. This negative data has a default TTL of only five minutes, because it is expected that the server or network will be repaired and begin to respond relatively soon in the future. Any future requests made for the document in the following five minutes will be informed that it is unattainable, in an average time of 20 ms rather than 75 seconds. Requests for the document after five minutes will try again to retrieve the document from the origin server.

3.10 Summary

In our study only HTTP traffic will be cached; FTP and Gopher traffic will not be investigated. The reason is that HTTP traffic has quickly become the majority of the traffic on the Internet. Also, studies have shown that, for a fixed disk space, HTTP caching yields higher hit rates and weighted hit rates than FTP and Gopher for the same disk space. Audio and video files will be cached, but only up to a limit of 4MB in size (all files are limited to 4MB in size) — this is the default configuration for the Harvest Cache. The default Harvest Cache TTL will be used; settings are given in Table 6.1. No attempt will be made to cache dynamic files. Negative caching will be used, using the default TTL value

of five minutes.

Chapter 4

Removal Algorithms Tested

This chapter describes the removal algorithms investigated in this thesis. Section 4.1 discusses the three most popular removal algorithms discussed in the literature. Section 4.2 discusses a pre-caching "removal" policy, which only caches documents if they appear to be worth caching. Section 4.3 discusses the new removal algorithms implemented in addition to the three discussed in section 4.1.

4.1 Popular Removal Algorithms

4.1.1 LRU and LFU

Least Recently Used (LRU) would appear to be intuitive to use; the longer it has been since the file was used the less likely it is to be used again. "Duplicate transmissions tend to occur within a small time window of one another" [8]. Likewise, Least Frequently Used (LFU) is intuitive because a file which has yielded many cache hits is likely to yield many more cache hits. This also benefits the cache for documents that are only used once, which are loaded into the cache and never used again. These documents displace other documents that may have resulted in cache hits. Studies have shown that as many as half of the referenced documents are never referenced again [8]. This would indicate that up to half the documents in a cache are useless, using a significant fraction of the cache and never yielding a cache hit.

Danzig *et al.* [8] found that LFU and LRU yield increasingly similar hit rates as the size of the cache increases: " ... the probability of seeing the same duplicate-transmitted file within 48 hours is nearly 90%. For this reason, LRU and LFU replacement policies are

nearly indistinguishable." They showed that with a cache of 4GB both LRU and LFU yield a hit rate of 45%. This is, however, contradictory to the VT-NRG Group's [31] discoveries (described in section 4.1.2). This contradiction in their results may be due to the simple fact that Danzig *et al.* studied server caching and the VT-NRG group proxy caching. Also they used different workloads.

Pitkow and Recker [23] in their studies of server caching found that "... recency proved to be a stronger predictor than frequency." Their studies showed that a mean hit rate of between 67% and 69% could be achieved by simply caching only the documents accessed one day ago. Their statistics are, however, for server caching and not proxy caching. Wessel [29] came to the same conclusion for proxy caching, that one day is a key caching point. He found that 10 minutes is also an important caching point. This would of course be included within the caching of documents from the last 24 hour period. Wessel's results indicate that documents that have not been accessed for a month are only rarely accessed again. Similarly the VT-NRG Group [3] found that for a server cache one hour and 28 hours are times at which there is a significant decrease in hit rate. This would indicate that if a user does not use a file within one hour they are unlikely to access it again in the same session. They may, however, re-use the file the next day at up to four hours later, but beyond this point they were unlikely to re-use the file.

In contrast with other studies Wessel's proxy cache size was a mere 15MB, yielding an explainably low 23.5% hit rate. In comparison to the 5MB Netscape persistent client cache hit rates of approximately 6.5% [22] this is a large improvement; for only a three fold increase in cache size, there is an almost four fold increase in the cache hit rate. However, the difference may be due in part to a different workload. This indicates the advantage, for a fixed disk space, of group caching in the proxy server over persistent caching at each client. There are two distinct disadvantages of proxy caching over client caching, the latency is increased and the scalability decreased.

4.1.2 SIZE

SIZE, although not an acronym, is written in upper case to be consistent with the other removal algorithm names. The VT-NRG Group [31] analyzed the performance of many different types of removal policy. They ranked hit rate performance in the following order (best first): SIZE, log(SIZE), LRU-MIN (which uses log(SIZE) as a primary key and LRU as a secondary key), LRU, Number of References (NREF). They found that using SIZE for the removal algorithm achieved up to 50% better hit rates than using NREF, achieving hit rates of 45% and 30% respectively. NREF (same as LFU), however, yields the highest weighted hit rate.

Cunha *et al.*'s [7] research found that there is a fixed latency due to the setup of an HTTP request; thus it is much more effective to cache small documents. The total latency to download a small file is affected more by the fixed overhead than the latency to download the actual file. For a large file this fixed overhead becomes small.

The research of the VT-NRG Group and Cunha *et al.* indicates that caching of small documents is clearly the most important factor to consider in achieving only the maximum *hit rate*. NREF may be the most important factor for achieving maximum *weighted* hit rate.

4.2 Pre-caching "Removal" Algorithm — Ignore First Hit

Client caches suffer enormously from caching documents that are only accessed once. Arlitt *et al.* found this to be a significant proportion of the documents in a cache: "This one-time referencing behavior means that, on average, one-third of a server cache could be cluttered with useless documents [4]". The VT-NRG group [30], in a sample of 50,000 URL accesses, found that more than 90% of the URLs were only referenced once. Obviously to cache these documents is wasteful for a cache with limited disk space.

A solution suggested by both the VT-NRG Group and Arlitt *et al.* is to keep a list of all documents that have been accessed, and to only retain them in cache when they have been requested a second time. Only the documents that are accessed at least twice would

be stored in the cache. The advantage in this system is that the number of documents that are only referenced once is disproportionally high compared to the number of documents that are only referenced twice. Equation (4.1) illustrates this feature.

$$\frac{\text{URLs referenced only once by the proxy}}{\text{All URLs referenced by the proxy}} >> \frac{\text{URLs referenced only twice by the proxy}}{\text{All URLs hit at least twice by the proxy}}. \tag{4.1}$$

It may be noted by the reader that this algorithm is similar to the LFU algorithm, but only up to the point of using the LFU algorithm to prevent caching URLs that are only accessed once. Henceforth this algorithm will be called the *Ignore First Hit* (IFH) algorithm.

4.2.1 IFH Implementation Design

A large list of a fixed size contains the set of most recently accessed URLs that are not in the cache. The size of this list is also a design issue. A fixed size array was chosen in preference to a linked list due to the high speed of access of hashed arrays. The length of the array is user definable in the *cached.conf* file. The default is 10000 nodes long. Each node in the array consists of five integers each of which are used to contain a hash value for one of five URLs. These are used for collisions in the array which effectively hash on the last 13 to 14 bits[1] of the 32 bit hash value. Also contained within each node is knowledge of which of the URLs was least recently accessed, and thus which to remove on the sixth collision. There is a trade off between the amount of memory used (e.g., a 10000 node list consumes 250KB of main memory), and the number of URLs that are maintained in the array. Once a sixth collision occurs the oldest URL will be removed from the list. This will then require a third hit on the URL before it is cached. This is, however, unlikely. What is more probable is that the URL was one of the many URLs referenced only once. A hit

[1] $2^{13} < 10000 < 2^{14}$

on a URL stored in the hash table means that the file is not currently cached, but should now become cached.

4.3 Post-Caching Removal Algorithms

The Harvest Cache ver1.4pl3 uses what initially appears to be a rather strange removal system. The system uses a hash table to store the documents so that access time per file is minimized. This makes it necessary to use a disk usage upper threshold level for the removal of documents. For example, if the cache is more full than the upper threshold limit, documents will be removed until the lower threshold level is reached.

Obviously keeping the cache below 100% utilization achieves suboptimal performance. However, this system is able to operate very quickly with reduced CPU demands. CPU cost is often far higher than disk space, thus this less than 100% utilization was not considered important; it was not changed in our modified Harvest Cache. Also there are benefits for certain removal algorithms *not* to have the cache utilization kept at 100%. For example, if the LFU algorithm is in use and every file that is currently stored in the cache has been accessed twice, no more documents can ever be added to the cache because once they are added to the cache, having been referenced only once, the next file that is added to the cache will remove the only other file from the cache that has been used only once. Thus documents that would have been hit many times never get the chance to increment their usage counter above one. The Latency (LAT) algorithm, described in section 4.3.4, and the Hybrid (HYB) algorithm, described in section 4.3.5 use complex evaluation methods to find the "value" of the file to the cache. These values can change over time. Keeping a sorted list would be too costly and removing only one file at a time too expensive.

Once the upper threshold has been reached the Harvest Cache takes 256 documents and sorts the top eight documents[2] for removal; these being the eight least valued documents. If the eight documents do not free up enough space then the next 256 documents are searched

[2]The choice of 256 and 8 are modifiable #defines in the file "store.c".

for their eight least valued documents. This is repeated until the lower threshold has been reached. This may not be the most efficient system but has been shown to be adequately fast; the Harvest group have reported impressive results, such as 200 requests per second. Consequently no modifications were made to the method to choose documents for removal. Only the removal algorithm was modified so as to allow the choice of replacement policy: LRU, LFU, SIZE, LAT, and HYB. These are each described in the following sections.

4.3.1 LRU

LRU is the default removal algorithm implemented in the original version of the Harvest Cache ver1.4pl3. This is included in the testing as a baseline. It is expected that the algorithms discussed in the following sections should perform better.

4.3.2 LFU

The VT-NRG group have found in their simulations that LFU often yields the highest weighted hit rate of all the removal algorithms [30]. This is included to see if LFU will provide an improvement in response time over LRU.

4.3.3 SIZE

SIZE yielded the highest hit rate of all the algorithms tested by the VT-NRG group [31]. SIZE is included to test its performance in response time experiments.

4.3.4 Latency, LAT

The Harvest Cache is user-configurable to prevent caching local documents, from machines in the same domain, such as ".vt.edu" or ".cs.vt.edu". There is sound reasoning behind this policy. Although caching of local documents may optimize hit rates, caching of local documents may actually cause a performance degradation. The proxy will be forced to serve the requests that may have been distributed across many local servers. To reduce

the average latency per byte, documents that download slowly are cached while quickly downloadable documents are unlikely to be cached, and local documents not cached at all.

Only recently has research using latency as a factor in the removal algorithm been investigated [5]. As mentioned previously, the latency to download a file is the product of its size divided by the link's bandwidth and the propagation delay in the network. Obviously it is more beneficial to reduce load on a network segment that is heavily over-loaded, than to reduce load on multiple lightly loaded network segments.

A hash indexed array stores server names along with the estimates of the connection latency (CLAT). Equation (4.2) uses the variable $clat_j$ to represent the CLAT for a particular server. In this chapter subscripts "i" and "j" are used to indicate files and servers respectively. Equation (4.3) uses the variable cbw_j to represent the CBW for a particular server. The value used for α, $\frac{1}{8}$, is the same as in the TCP smoothed estimation of Round Trip Time (RTT) [26]. The effect of this constant is that the new value reflects seven eighthes of the original value and one eighth of the most recently calculated value. The new value of cbw_j is calculated using the sample download time minus the $clat_j$ time, and the sample size minus a constant CONN.

$$clat_j = (1-\alpha)(clat_j) + \alpha(\text{new } clat_j) \tag{4.2}$$

$$cbw_j = (1-\alpha)(cbw_j) + \alpha(\text{new } cbw_j) \tag{4.3}$$

The calculations, shown in equations (4.2) and (4.3) are complicated slightly by the fact that we are unable to exactly determine the connection time. The constant CONN is used as the threshold size to determine whether the download latency of a document is to be considered for the CLAT or the CBW calculations. If the document is smaller than CONN, the time to download the document is used only for the CLAT calculations, otherwise it is used for the CBW calculations. This estimate is smoothed in a similar manner to TCP RTT latency estimates [14, 26], using both old and new requests to the server as factors of the new estimate. Equation (4.4) shows the LAT algorithm's formula, the variable lat_i is

the LAT value for a particular file. *Documents with the smallest value as calculated by lat are removed first.* The following code extract illustrates the actual calculation method.

$$lat_i = clat_j + \frac{size_i}{cbw_j} \tag{4.4}$$

```
#define BYTES_CONNECT_TIME_ONLY 2048
#define SMOOTH_COEF 0.125

/* int bytes   - the number of bytes in the URL file */
/* int time    - the milliseconds to download the URL file */

/* a separate msec_connect and CBW is kept for each server name
 * this code has been simplified for readability */

if (bytes < BYTES_CONNECT_TIME_ONLY)
  msec_connect += SMOOTH_COEF *  (time - msec_connect);
else  /* more than just a small connect time only packet */
      /* the multiplier of 1000 is required because the time is in
         milliseconds, the 1000 converts this to seconds */
  CBW += SMOOTH_COEF * (((1000 * (bytes - BYTES_CONNECT_TIME_ONLY))
                       /(time - msec_connect)) - CBW);
```

4.3.5 Hybrid, HYB

The results of experiment one, discussed in section 7.3, show that the LAT algorithm performs poorly. Bolot and Hoschka suggest that removal algorithms should consider many more factors than just the download time in ranking documents for replacement [5]. Their algorithm, on an artificial workload, illustrates that using latency *with* other factors can perform better than SIZE or LFU, see equation (4.5). The variables associated with their formula include: t_i, the time since the document was last referenced, S_i, the size of the

Table 4.1: Important factors and setting that makes a document less likely to be replaced.

Factor	Least likely to be replaced if factor value is:
Connection latency	high
Connection bandwidth	low
Size of the file	low
Number of references	high

document, rtt_i, the time it took to retrieve the document, and ttl_i, the time to live of the document. They also use four, seemingly arbitrary, constants, $w_1 = 5000b/s$, $w_2 = 1000$, $w_3 = 10000bs$, and $w_4 = 10s$. *Documents with smaller values of W are removed first.*

$$W(t_i, S_i, rtt_i, ttl_i) = \frac{(w_1 rtt_i + w_2 S_i)}{ttl_i} + \frac{(w_3 + w_4 S_i)}{t_i} \tag{4.5}$$

Using Bolot and Hoschka's suggestions to incorporate other factors led to the development of a new hybrid removal algorithm to test. *Documents with smaller values of hyb are removed first.*

$$hyb_i = \frac{\left(clat_j + \frac{W_B}{cbw_j}\right) \cdot nref_i^{W_N}}{size_i} \tag{4.6}$$

Four factors are considered important. These factors and their preferred ranges are listed in Table 4.1. Equation (4.6) illustrates how the four factors in Table 4.1 affect the value of the document. The values $clat_j$ and cbw_j are calculated using the same method as for the LAT algorithm. This hybrid (HYB) algorithm is used instead of the LAT algorithm in experiments two and three.

The constant W_B, whose units are bytes, is used for setting the relative importance of the connection time versus the connection bandwidth. The constant W_N, which is dimensionless, is used for setting the relative importance of *nref* versus *size*. As $W_N \to 0$ the emphasis is placed upon *size*. If $W_N = 0$ *nref* is not taken into account at all. If $W_N > 1$ then the emphasis is placed more greatly upon *nref* than *size*. The values of the constants,

Table 4.2: Comparison of values of *clat* in the HYB algorithm

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A,any file	**10s**	1KB/s	x	y	$\frac{(10+10)\cdot x^B}{y}$	Retained
B,any file	**1s**	1KB/s	x	y	$\frac{(1+10)\cdot x^B}{y}$	Removed

Table 4.3: Comparison of values *cbw* in the HYB algorithm

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A,any file	1s	**10KB/s**	x	y	$\frac{(1+1)\cdot x^B}{y}$	Removed
B,any file	1s	**1KB/s**	x	y	$\frac{(1+10)\cdot x^B}{y}$	Retained

Table 4.4: Comparison of values of *nref* in the HYB algorithm

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A,file 1	1s	10KB/s	**5**	y	$\frac{(1+1)\cdot 5^B}{y}$	Retained
A,file 2	1s	10KB/s	**3**	y	$\frac{(1+1)\cdot 3^B}{y}$	Removed

W_B, W_N, and CONN, are modified to determine their relative importance (see Chapters 8 and 9).

The examples in Tables 4.2, 4.3, 4.4, and 4.5 illustrate which documents would be retained and which would be removed under different conditions. Effects due to the value of *clat, cbw, nref,* and *size* are presented. Obviously any combination of these can be calculated, and the four examples are given simply to show the effect of each factor individually. In Table 4.5, if $W_N > 1$ the emphasis would be on *NREF*, and thus file 4 would be removed, if $0 \leq W_N < 1$ then the emphasis would be placed on *size* thus file 3 would be removed.

4.4 Removal Algorithm Data Structure Storage Requirements

All of the removal algorithms, except SIZE, require additional data space for their operation. This section discusses the quantity of memory required for their data structures.

Table 4.5: Comparison of *nref* and *size* in the HYB algorithm

Server	*clat*	*cbw*	*nref*	*size*	*hyb*	Result
A,file 3	1s	10KB/s	**5**	**50KB**	$\frac{(1+1)\cdot 5^B}{50}$	See text
A,file 4	1s	10KB/s	**3**	**30KB**	$\frac{(1+1)\cdot 3^B}{30}$	See text

4.4.1 LRU

Obviously LRU must maintain a record of the last time that the URL was referenced. This is stored within the data structure at the cost of a four byte integer per URL.

4.4.2 LFU

The structure required by the LFU removal algorithm is maintained at a cost of one four byte integer counter per URL. The counter is incremented every time the file is referenced. A short integer, two bytes long, cannot be used, because certain documents on particular servers have been shown to have been hit 50000 times in the duration of a test, which indicates the need for a four byte integer.

4.4.3 SIZE

Although the size is maintained at the cost of one four byte integer per URL, this is effectively cost free because the size of the URL must be known for the operation of swapping the file in and out of the cache. Thus there is no extra cost for this algorithm.

4.4.4 LAT

The size of the data structure for the LAT algorithm is very different than the previous three. The cost can not be attributed on a per URL basis, but on a per server basis. Thus if the number of documents cached per server increases then the cost of the data structure is proportionally decreased.

In our implementation we use a large[3] hash table which contains pointers, each costing

[3]This is a user definable size, typically of the order of 10000 or larger.

four bytes. Once a server has been associated with a pointer in the hash table a structure is created and pointed to by the appropriate pointer in the hash table. This structure is comparatively large relative to that of the four byte cost per URL in the previous three removal algorithms. The structure contains the server name as a string; this is usually of the order of 10 to 20 bytes; for example "csgrad.cs.vt.edu" and "www.netscape.com" are both 17 bytes long, including a byte for the NULL at the end of the string. The structure also contains two integers: one for the smoothed connection time estimation, and one for the smoothed bit rate estimation. Finally the structure also contains a four byte pointer. This pointer is usually set to NULL; it is used to point to the next structure if there is a hash collision.

4.4.5 HYB

The HYB algorithm uses data structures identical to those of all the LFU, SIZE, and LAT algorithms simultaneously. Eight bytes are used per URL for the LFU and SIZE counters, however the integer required by the SIZE algorithm is required by the proxy anyway. Thus the cost is four bytes per URL, and the cost of the LAT data structure.

4.5 Using a Second Removal Algorithm in the Case of a Tie

Occasionally two documents will appear to have equal "worth" as calculated by the primary removal algorithm. Using a second removal algorithm to decide which file to remove seems appealing. No study published, however, shows results that would conclude that using a second removal algorithm improves the hit rate significantly [31]. What is significant, however, is the cost of implementing a second sorting algorithm for the marginal performance improvement. The objective of this thesis is to improve response time for a large user base, where the CPU cost of removal is significant and thus must be minimized. In particular, as discussed in section 4.3, the removal system removes 8 documents from the set of 256. This set of 256 is chosen from a list in hashed order, effectively a random choice.

Thus the second-order removal algorithm we use is effectively a random removal algorithm.

4.6 Summary

Five removal algorithms are implemented and tested in the thesis, these being: LRU, LFU, SIZE, LAT, and HYB. The LAT algorithm uses the estimated download time to weight the files. The HYB algorithm uses a combination of the connection time, download time, file size, and number of past references to the file. Three of the removal algorithms, LRU, LFU, and HYB, use an additional four bytes of storage per URL. LAT and HYB also require storage of nine bytes plus the length of the origin server name in bytes, per origin server.

Chapter 5

Methodology

This chapter discusses the methodology of the experimental design, initially the hardware requirements are defined. Following this the experiment order and the analysis methods used for testing the removal algorithms are discussed.

5.1 System Requirements

All experiments in this thesis were performed using a modified version of the Harvest Cache, which is one of a group of five applications. The Harvest manual [13] does not, however, indicate what type of machine is required if only one of the applications is to be executed. The manual states: "A good machine for running a typical Harvest server will have a reasonably fast processor (e.g., Sun Sparc 5, DEC Alpha, Intel Pentium), 1-2 GB of free disk, and 64MB, or more, of RAM. ... If you do not have enough memory, your system will page too much, and drastically reduce performance." The modified Harvest Cache was executed on three different types of machines:

- a Sun Sparc 10 with 64MB of RAM running Solaris 2.4 (SunOS 5.4),
- a DEC Alpha running OSF/1 3.2 with 96MB of RAM,
- an IBM RS6000[1] quad processor machine running AIX 4.1, using PowerPC 604 chips at 112MHz with 512MB of RAM, model number 7013-J30, although it was upgraded to a functionally equivalent 7013-J40.

[1]Ported to the RS6000 by Tommy Johnson, Dept of Comp. Sci., VA Tech, tjohnson@csgrad.cs.vt.edu.

5.2 Order of Experiments

A simple verification of the removal algorithms was performed; details and results of this are given in Appendix B. Following this some simple initial tests (Chapter 6) were performed. These were used to set the parameters of the proxy for the subsequent experiments.

Experiment one, named the VTCS1 workload (Chapter 7), compares the performance of the LRU, LFU, SIZE, and LAT removal algorithms. The proxy was modified so that it could emulate four removal policies simultaneously. A document is fetched from the origin server if *any* of the removal algorithms would have required a fetch. In this manner it is possible to obtain a direct comparison of the performance of the four removal algorithms. The results of experiment one illustrate that a removal algorithm using *only* time to download (the LAT algorithm) yields very poor results. The problems with the LAT algorithm and thus the advantages of the Hybrid (HYB) algorithm are discussed in section 7.3.

Experiment two, discussed in Chapter 8, differs from that of experiment one in only two respects: the LAT algorithm is replaced by the HYB algorithm and the testing period is two weeks. This experiment was run twice using different constants in the HYB algorithm in each trial. The two workloads are named VTCS2 and VTCS3.

Experiment three, in Chapter 9, is completely different from the other experiments. In the previous experiments a single proxy is used to emulate four different removal algorithms. Due to the burden of emulating multiple removal algorithms performance degradation was introduced by the proxy itself. This is the reason for experiment three where four proxies running in parallel on a four processor IBM RS6000 are used rather than one proxy emulating four removal algorithms. Log files are used to allow a client application to make requests to all four proxies in parallel. Using this system, log files were replayed at significantly faster than real time speeds. Three different log files are used, one each from the following three domains: “cs.vt.edu”, “bu.edu”, and“lib.vt.edu”, these being from the Virginia Tech Computer Science Department, a small computer lab at Boston University, and the Virginia

Table 5.1: Experiment numbers and workload names.

Experiment	Workload
1	VTCS1
2, part 1	VTCS2
2, part 2	VTCS3
3, part 1	VTCS2
3, part 2	BULOG
3, part 3	VTLIB

Tech Newman (main campus) library. The log file from the Computer Science Department is the log file obtained from the VTCS2 experiment. The other two experiments will be denoted as BULOG, and VTLIB. The log file from Boston University was created without the use of client caching. Also tested in this experiment are four different sets of values for the constants in the HYB algorithm. In these tests, all four parallel proxies are set up using the HYB algorithm, each with different constants. In the event that a URL is no longer valid when being re-played it is aborted and its time is not included in the proxy logs; this is the same as when a user interrupts a connection in experiment one and two. Table 5.1 records the experiment number and the names of the log files.

5.3 Methods to Analyze the Results

This section discusses the performance measures recorded during the experiments: HR, WHR, and TIME. Each of these performance measures are used to generate graphs and confidence intervals to determine which removal algorithms perform best for each performance variable.

Graphs are generated for each performance measure, for every removal algorithm, and for each workload. All the graphs are smoothed over a 24 hour period, in an attempt to visually reduce the fluctuations in the results due to the workload. Time zero on the graphs indicates the point at which the cache is full of documents and has started to remove them. The smoothed y-axis value at every point on the graph is the mean of the previous 24 hours. Equation (5.1) illustrates the smoothing function $S(Y_k^m)$. In this chapter the subscript "k"

is the data point (hour), and the superscript "m" is the removal algorithm.

$$S(Y_k^m) = 100 \cdot \frac{\sum \left(Y_{k-23}^m + Y_{k-22}^m + \cdots + Y_{k-1}^m + Y_k^m \right)}{24} \tag{5.1}$$

The x-axis of the graphs, in Chapters 7 and 8, for experiment one and two is time, in hours; the proxy records data on an hourly basis. Experiment three (Chapter 9) is accelerated, consequently the x-axis is number of URLs referenced. A smoothing period is chosen to provide a similar smoothing characteristic as the 24 hour period for experiment one and two.

The fluctuations in HR, WHR, and TIME are often so large that the range of the y-axis on the graphs is so great as to hide the differences between the removal policies. Consequently normalized graphs are used, they illustrate the performance difference of each policy from the mean of the four policies at each smoothed data point. The normalizing function requires a function to determine the mean of the values returned for each removal algorithm from the smoothing function defined in equation (5.1). The mean-smoothing function, $\bar{S}(Y_i)$, is defined in equation (5.2).

$$\bar{S}(Y_k) = \frac{S(Y_k^{LRU}) + S(Y_k^{LFU}) + S(Y_k^{SIZE}) + S(Y_k^{HYB})}{4} \tag{5.2}$$

Equation (5.3) shows the formula for calculating the normalized value, $N(Y_k^m)$, which uses the mean of the smoothed value, which is obtained from equation (5.2).

$$N(Y_k^m) = 100 \cdot \frac{S(Y_k^m) - \bar{S}(Y_k)}{\bar{S}(Y_k)} \tag{5.3}$$

Confidence intervals are also obtained from the data to determine which algorithms, if any, are significantly better than the others. The method of obtaining replications for the significance tests is different for each of the different experiments. In experiment one a two factor F-test [15] was performed. No replications were used, and the two factors were

the removal algorithm and the hour the data was recorded at. In the second and third experiment only data after the cache has filled up and started to remove files is used. In the second experiment the mean of each 24 hour period was used as a replication for the confidence interval (CI) significance test, see equation (5.4). In the third experiment the mean of each 5000 URLs was used as a replication for the CI test, see equation (5.5). Any data at the end of the log file that does not form a 24 hour period or a 5000 URL block is discarded. These sets of data are then normalized using equation (5.3) before calculating the CI.

$$Y_0 = \text{hour at which proxy starts to remove files because it is full.} \tag{5.4}$$

$$\begin{aligned} \text{Replication 1} &= 100 \cdot \frac{\sum (Y_0 + \cdots + Y_{23})}{24} \\ \text{Replication 2} &= 100 \cdot \frac{\sum (Y_{24} + \cdots + Y_{47})}{24} \end{aligned}$$

$$Y_0 = \text{URL number at which proxy starts to remove files because it is full.} \tag{5.5}$$

$$\begin{aligned} \text{Replication 1} &= 100 \cdot \frac{\sum (Y_0 + \cdots + Y_{4999})}{5000} \\ \text{Replication 2} &= 100 \cdot \frac{\sum (Y_{5000} + \cdots + Y_{9999})}{5000} \end{aligned}$$

5.4 Workloads Used

This section analyses the workloads of the three experiments. First the number of clients using the proxy in each workload is described. Then the file type distribution is described.

Table 5.2 presents experiment numbers, workload names, the number of different IP addressed clients (browsers) using the proxy during this period, the minimum number of

Table 5.2: Number of clients in experiment one and two.

Experiment number	Workload name	Clients with different IP addresses	**Minimum** number of users	Number of different servers	Period of log file
1	VTCS1	62	100	3100	Sep 4 - Oct 3, 1996
2, part 1	VTCS2	178	210	3000	Oct 31 - Nov 14, 1996
2, part 2	VTCS3	170	210	3100	Nov 15 - Dec 3, 1996
3, part 2	BULOG	30	Many	530	Nov 1 '94 - Feb 28 '95
3, part 3	VTLIB	34	Many	1300	Nov 19 - Nov 27, 1996

different users actually using the proxy, the number of different servers accessed, and the time period of the experiment. The first part of experiment three uses the log file obtained from experiment two, part one, the VTCS2 workload. In all cases URLs in the local domain were not cached, in the case of the BULOG the URLs of the local domain were removed before the log file was used in experiment three; the local domains are: "cs.vt.edu", "bu.log", and "lib.vt.edu" for the three workloads, VTCS, BULOG, and VTLIB, respectively.

The reason for the difference between the numbers, of IP addresses and actual users, is multiuser machines were used. A multiuser machine appears as a single client to the proxy. One machine in particular, *csgrad.cs.vt.edu*, is known[2] to have at least 40 users using the proxy. The lab in the Boston University trace has over 30 public machines, the library trace has 34 different IP addresses most of which are machines used publicly by many users.

Table 5.3 illustrates the percentage of each file type in the VTCS1 workload. The "DIR" entry includes the URLs that end in "/", these are usually ".html" files. The "Script" entry includes all URLs with "?", "map", or "cgi" as part of the entry. Tables 5.4 and 5.5 illustrate the percentage distribution in the VTCS2 and VTCS3 workloads. The percentage distribution of the BULOG and VTLIB workloads are illustrated in Tables 5.6 and 5.7. All the Virginia Tech workloads, VTCS1, VTCS2, VTCS3, and VTLIB illustrate similar traits. Graphics are approximately 67% of the URL requests, and 52% of the bandwidth. Text/html/directories are approximately 25% of the URL requests and bandwidth. Video

[2]Email correspondence with all *csgrad* users was used to discover the actual number of proxy users.

Table 5.3: Analysis of file types accessed VTCS1 workload.

Document type	Number	Size (MB)	% NREF	% Bytes
Graphics	76934	481.86	69.5	51.4
Text/HTML	18439	225.18	16.7	24.0
DIR	9830	63.96	8.9	6.8
Script	1939	8.19	1.8	0.9
Audio	297	13.49	0.3	1.4
Video	28	28.19	0.0	3.0
other	3157	116.60	2.9	12.4
Total	110624	937.48	100.0	100.0

Table 5.4: Analysis of file types accessed in VTCS2 workload.

Document type	Number	Size (MB)	% NREF	% Bytes
Graphics	60050	385.85	67.2	52.7
Text/HTML	13759	103.89	15.4	14.2
DIR	10101	52.58	11.3	7.2
Script	1517	58.40	1.7	8.0
Audio	169	33.71	0.2	4.6
Video	71	46.99	0.1	6.4
other	3671	51.14	4.1	7.0
Total	89338	732.56	100.0	100.0

and audio confirm traits found by other studies, showing that they use a disproportionally high bandwidth for the number of references. The Boston University log, Table 5.6, refers to a higher percentage of graphics files, 85% of the requests and 61% of the bandwidth used. The reason for the higher percentage of graphics files in the BULOG trace is because small icons are used multiple times within a HTML document, which are usually cached by the browser, as in the VTCS and VTLIB workloads, are not cached by the browser in the BULOG workload.

Table 5.5: Analysis of file types accessed in VTCS3 workload.

Document type	Number	Size (MB)	% NREF	% Bytes
Graphics	57306	349.44	67.6	47.7
Text/HTML	13961	111.02	16.5	15.2
DIR	8911	56.23	10.5	7.7
Script	1042	5.92	1.2	0.8
Audio	131	11.82	0.2	1.6
Video	48	68.79	0.1	9.4
other	3417	129.13	4.0	17.6
Total	84816	732.35	100.0	100.0

Table 5.6: Analysis of file types accessed in BULOG workload.

Document type	Number	Size (MB)	% NREF	% Bytes
Graphics	40543	59.20	85.5	61.6
Text/HTML	4008	19.82	8.5	20.6
DIR	2514	6.32	5.3	6.6
Script	85	0.38	0.2	0.4
Audio	32	5.97	0.1	6.2
Video	12	2.33	0.0	2.4
other	209	2.09	0.4	2.2
Total	47403	96.12	100.0	100.0

Table 5.7: Analysis of file types accessed in VTLIB workload.

Document type	Number	Size (MB)	% NREF	% Bytes
Graphics	25096	127.05	63.2	61.4
Text/HTML	6756	40.13	17.0	19.4
DIR	5379	17.95	13.5	8.7
Script	1235	4.69	3.1	2.3
Audio	30	0.35	0.1	0.2
Video	9	8.14	0.0	3.9
other	1218	8.61	3.1	4.2
Total	39723	206.91	100.0	100.0

Chapter 6

Initial Tests

6.1 Initial Test Configuration

An initial test was performed to calculate what would be suitable configurations[1] for the proxy before running it to analyze the performance of the different removal algorithms. The initial experiment was primarily to find an appropriate cache size for the proxy. The size of the disk cache is important if the system is to yield useful results. If the size is too small, documents will constantly be replaced before they have yielded many hits. Studies have shown that if the size of the disk cache is extremely small (say 1%) compared to the quantity of traffic the best removal policy will be different than when the disk space is 10% to 20% of the traffic. These are not the conditions being used in this thesis. If the size is too large too few replacements will occur for the algorithms to show any significant difference in performance. The following parameters were used in the preliminary test:

- cache disk size 10MB,
- maximum size of cachable documents 4MB,
- cache disk upper threshold 90%,
- cache disk lower threshold 75%,
- testing period 1 week,
- no caching of local (cs.vt.edu) documents, and

[1] Configurations as set in the " HARVEST_HOME/lib/cached.conf" file.

Table 6.1: Table showing how TTL is set in the Harvest Cache.

Regular Expression matching URLs	Absolute TTL (days)	staleness factor	Maximum TTL (days)
^http://	1	20%	30
^ftp://	0	0%	0
\.gif	2	50%	30
/cgi-bin/	0	0%	30
Other	7	N/A	N/A

- TTL values are set as shown in Table 6.1. If the document's header data contains a creation date such that its age can be calculated, a percentage of its age is used as the TTL, up to the maximum illustrated in the table. If the age of the file can not be determined a default value is used.

6.2 Results of the Initial Test

It was found that when the proxy was used for the VTCS workloads 10MB for the proxy size was totally inadequate. The volume of traffic experienced by the proxy was of the order of 50MB per day. Thus the cache size was set to 50MB for the VTCS and VTLIB workloads, the BULOG workload used a cache size of 10MB due to the small size of the traffic in the log. The disk cache upper and lower thresholds were changed to 98% and 92% respectively. An upper threshold of 98% was used in preference to 100% because some overhead per document is required (see section 4.4). At 98% the proxy will use 49MB for data, leaving 1MB for file headers and storage information. The lower threshold of 92% was chosen to be 4MB (for a 50MB cache) less than the upper threshold, 4MB being the size of the largest cachable document, as discussed in section 3.2.

Chapter 7

Experiment One: LRU, LFU, SIZE, and LAT

This chapter discusses the results of the first performance test of the removal algorithms. The duration of this experiment was four weeks. Conclusions of the performance of the LAT algorithm are made at the end of the chapter. The mean HR, WHR, and TIME for the removal algorithms, in this experiment, are shown in Table 7.1.

7.1 HR and WHR

This section discusses the results of the first experiment with respect to the HR and WHR for the four algorithms. The HR varied from 18% and 52%, and the WHR from 13% to 52%. Due to the quantity of data on the graphs, even when smoothed over 24 hours, normalized graphs are also illustrated.

Figures 7.3 and 7.4 show the disappointingly poor results that LAT is always worst. The results with respect to the other three algorithms are consistent with the VT-NRG group's findings [31, 30]. The SIZE algorithm yielded the highest HR, and LFU yielded the highest WHR. The other aspect that can be seen in the two graphs is that LFU and LRU are very similar.

Table 7.1: Mean HR, WHR, and TIME values for Experiment 1

Algorithm	HR %	WHR %	TIME Sec/File
LRU	36.8	33.7	2.44
LFU	36.8	33.8	2.45
SIZE	37.4	33.4	2.44
LAT	35.4	32.3	2.47

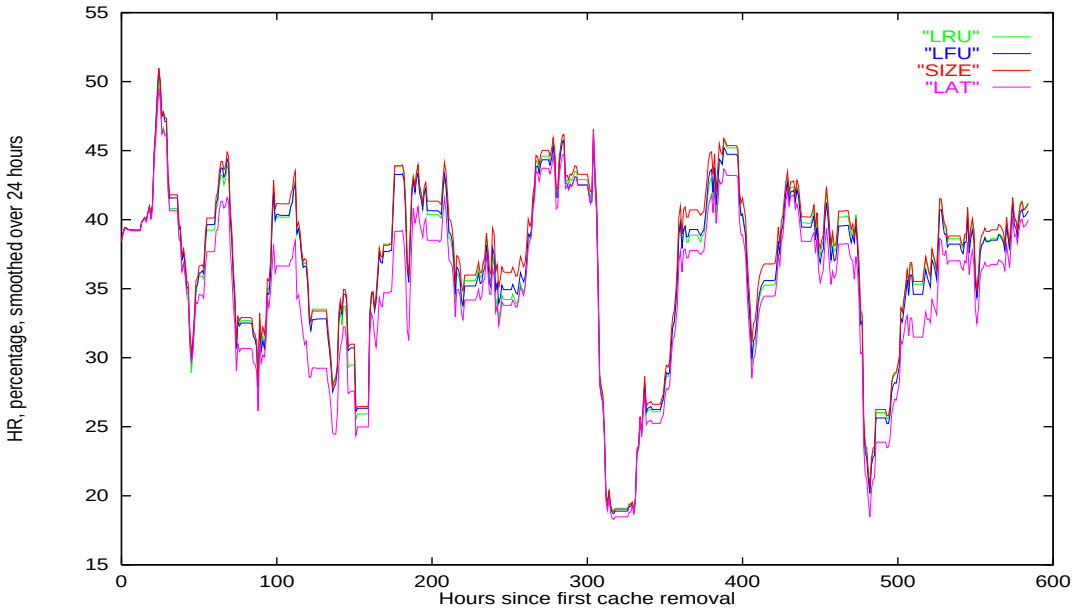

Figure 7.1: Experiment one: Hit rates, VTCS1.

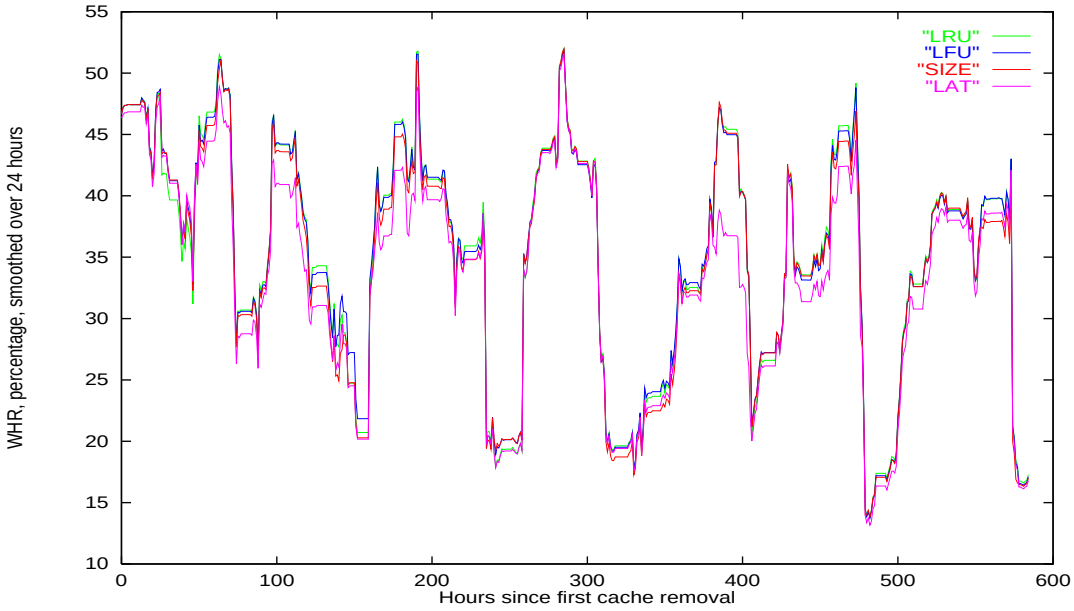

Figure 7.2: Experiment one: Weighted hit rates, VTCS1.

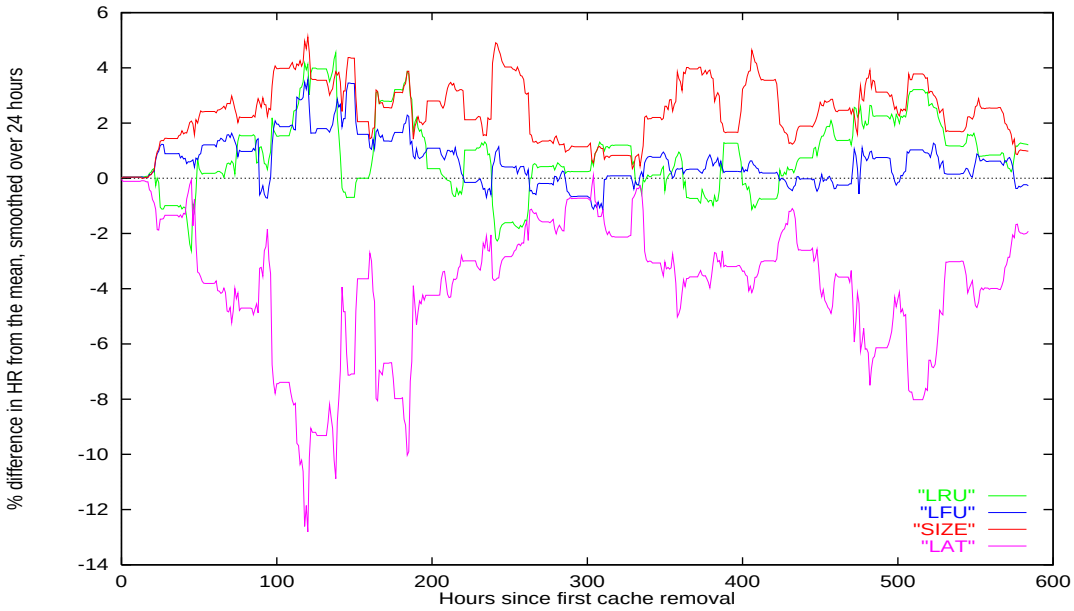

Figure 7.3: Experiment one: Hit rates, normalized, VTCS1.

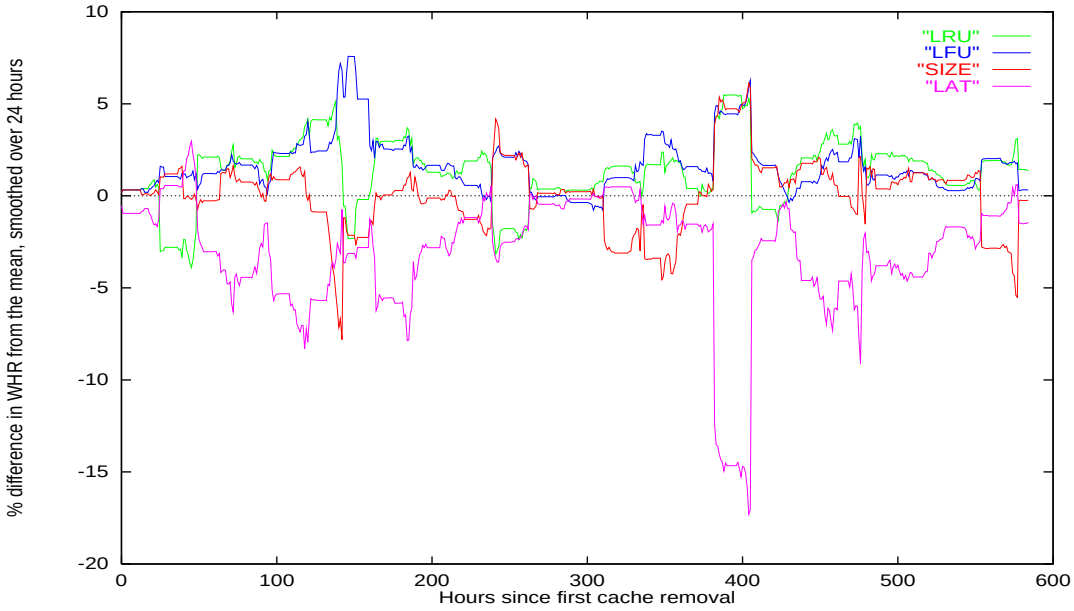

Figure 7.4: Experiment one: Weighted hit rates, normalized, VTCS1.

7.2 TIME and CBW

Figures 7.5 and 7.6 also show that the LAT algorithm performed worse than the other three algorithms. It is, however, very interesting to note the trends in both of the graphs. Figure 7.5 illustrates the almost continuous *increase* in the time per file. The fact that Figure 7.6 shows that the CBW declines as time increases indicates that a user behavior change towards larger files has not occurred. Figures 7.1 and 7.2 illustrate that the HR and WHR has not reduced. The only possible reason is that either the performance of the proxy or the Internet has caused this degradation.

There is no reason to believe the Internet has reduced in performance so much, if at all. The reason is that because the proxy was simulating all four algorithms it became over loaded and caused the performance degradation itself. This reduction in performance is not believed to have affected the relative performance of the removal algorithms. Thus the normalized graphs in Figures 7.7 and 7.8 are more useful as they illustrate the relative performances of the algorithms. This degradation in performance due to the proxy itself is the reason for experiment three.

7.3 Conclusions of Experiment One

The "infinite cache size" (ICS) of the log can be determined. The ICS is the disk space required so that no replacements are made. The ICS for this experiment was 496MB, the cache — set at 50MB — was therefore 10% of the ICS. Also the infinite hit rate (IHR), and infinite weighted hit rate (IWHR), can be calculated. These are the hit rates for the cache with the "infinite" disk space, over the entire duration of the trace: they are the maximum achievable hit rate and weighted hit rate. In this experiment the IHR is 54%, and the IWHR 45%. The HR and WHR of the proxy using only 10% of the ICS is 37% and 34% respectively.

The results from this first removal algorithm experiment, shown in the previous graphs and Table 7.1 are disappointing. However there are clear reasons why the algorithm does

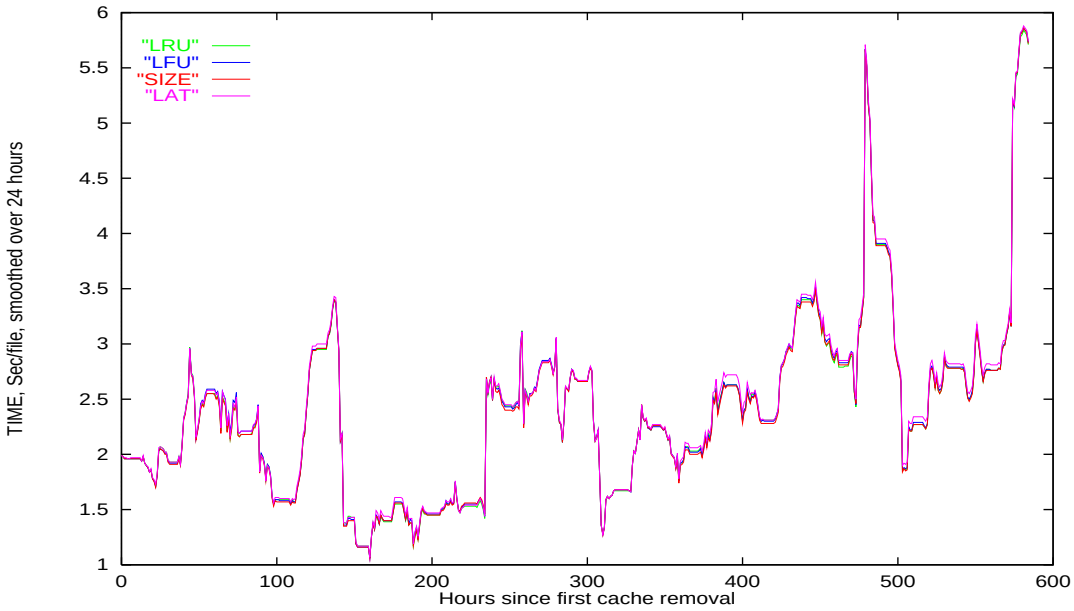

Figure 7.5: Experiment one: Average time to download per file, VTCS1.

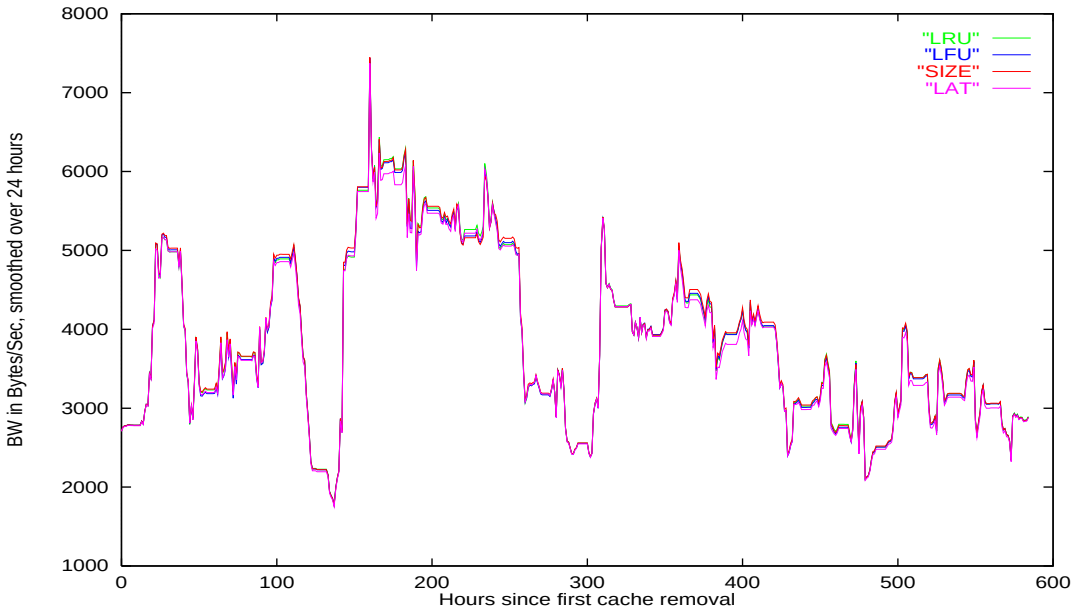

Figure 7.6: Experiment one: Connection bandwidth, VTCS1.

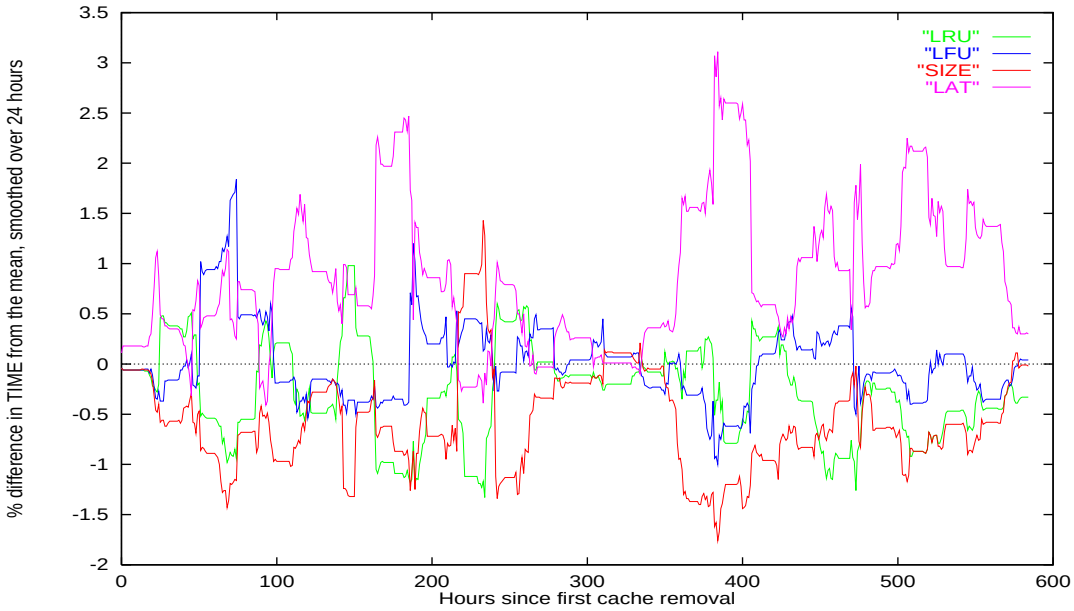

Figure 7.7: Experiment one: Average time to download per file, normalized, VTCS1.

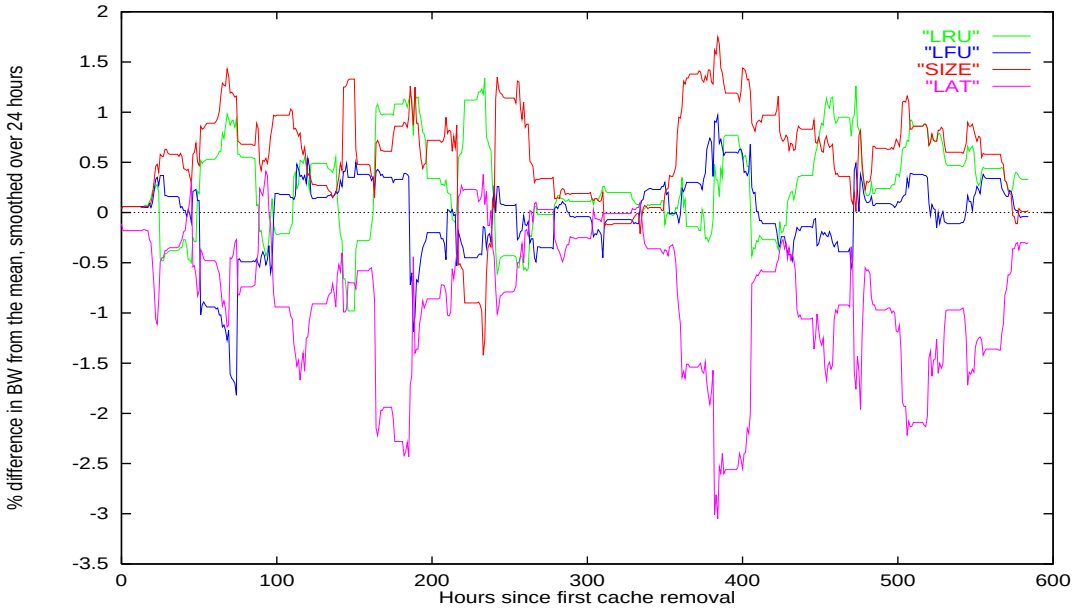

Figure 7.8: Experiment one: Connection bandwidth, normalized, VTCS1.

not work. The LAT algorithm, for a single server, favored the largest documents from that server. This resulted in a removal algorithm that worked in the *opposite* manner to the SIZE algorithm! As the SIZE algorithm was shown in [31] to be the best performer for at least HR it is logical to conclude that the LAT algorithm should do very badly. The LFU algorithm was the best performer for WHR. The LAT algorithm does not take into consideration how many times a file has been hit. The LAT algorithm is thus also likely to yield a poor WHR.

A two factor F-test was performed on the results to determine if the difference in removal algorithm performance was significant at the 95% level. The first factor is the removal algorithm, and has four levels. The second factor was workload, in particular the hour at which the log file was recorded. Thus this factor has 682 levels, representing just over four weeks. Using the 95% confidence interval the removal algorithm *factor* is significant for all performance measures, and the workload (hour) factor is also significant. The fact that the algorithm *factor* is significant and that LAT is consistently worst for HR, WHR, TIME, and CBW, in each of the graphs, would imply that the algorithm is poor - without the need for a significance test on the actual levels within the algorithm factor.

Chapter 8

Experiment Two: LRU, LFU, SIZE, and HYB

This chapter, the first of two testing of performance of the HYB algorithm, discusses the experimental design, results, and conclusions of experiment two.

8.1 Experimental Design

Two tests were performed using all four algorithms, LRU, LFU, SIZE, and HYB. The two tests differed in their choice of the constants in the HYB algorithm. Each test was performed for two weeks. Of the three constants, W_B, W_N, and CONN, used in equations (4.2, 4.3, and 4.6), discussed in sections 4.3.4 and 4.3.5, CONN was not believed to significantly effect the performance of the HYB algorithm. Thus it was not varied, in an attempt to reduce the complexity of this experiment. An experiment where the value of CONN is changed is discussed in Chapter 9. The two tests use the values illustrated in Table 8.1. This is a 2^{2-1} design. However, no analysis of the percent variation due to W_B and W_N is made. Analysis of the percent variation due to the choice of constants is left until Chapter 9. The reason for changing the values of W_B and W_N was to determine if they made such a major effect to the HYB algorithm so as to change its position when ranked against the other algorithms performance in the two parts of this experiment. Table 8.2 present the mean HR, WHR, and TIME achieved by each of the removal algorithms in the two experiments.

Table 8.1: Choice of constants used in the HYB algorithm for experiment two.

Part	Workload	W_B	W_N	CONN
1	VTCS2	8KB	0.9	2KB
2	VTCS3	16KB	1.1	2KB

Table 8.2: Mean HR, WHR, and TIME values for experiment two

Workload	Algorithm	HR %	WHR %	TIME Sec/File
VTCS2	LRU	24.6	21.4	2.93
VTCS2	LFU	25.3	21.8	2.91
VTCS2	SIZE	26.0	18.4	2.96
VTCS2	HYB	26.8	20.7	2.88
VTCS3	LRU	23.1	17.4	3.31
VTCS3	LFU	23.4	17.6	3.29
VTCS3	SIZE	24.3	16.0	3.29
VTCS3	HYB	24.3	16.9	3.26

8.2 Results of Part One — VTCS2 Workload

The following two subsections include graphs from the results of part one, the VTCS2 workload. The first subsection presents the results on HR and WHR, and the second subsection presents the TIME results.

8.2.1 HR and WHR

Figures 8.1 and 8.2 illustrate the HR and WHR results obtained from the four removal algorithms: LRU, LFU, SIZE, and HYB on the VTCS2 workload. The constants in the HYB algorithm are those of the VTCS2 workload displayed in Table 8.1. The graphs show the HR ranges between 25% and 30%, and the WHR ranges between 20% and 25%. The differences between the four algorithms is comparatively small compared with the range of hit rates. Figures 8.3 and 8.4 illustrate the normalized graphs.

8.2.2 Average Time to Download

Figure 8.5 illustrates the average time per file in seconds. As described in the conclusions of this chapter, section 8.4, the performance degrades over time. Hence the normalized graph illustrated in Figure 8.6 is much more useful in determining which removal algorithm performs superiorly.

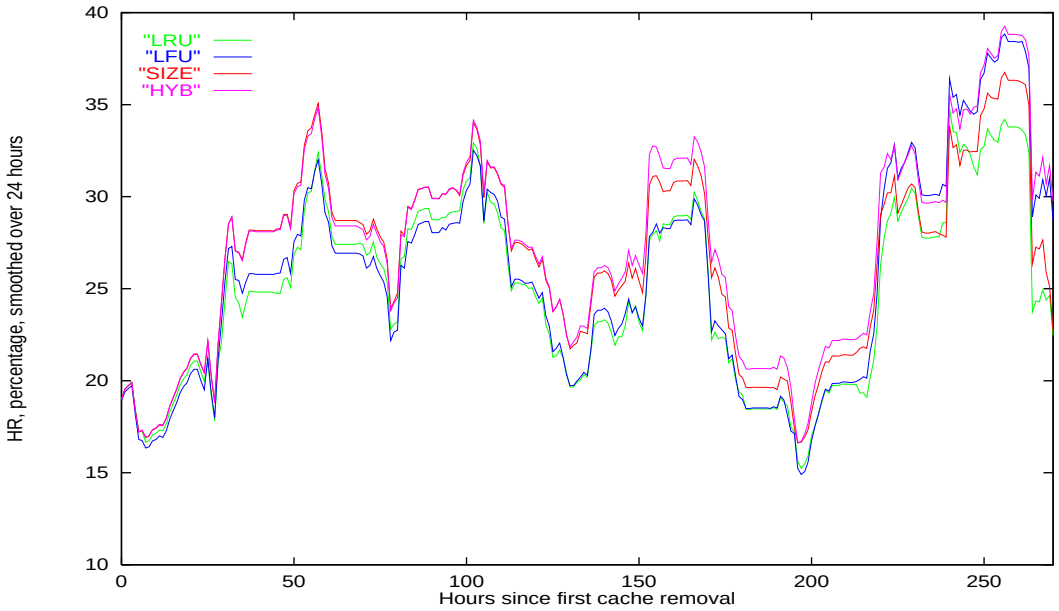

Figure 8.1: Experiment two: Hit rates, VTCS2.

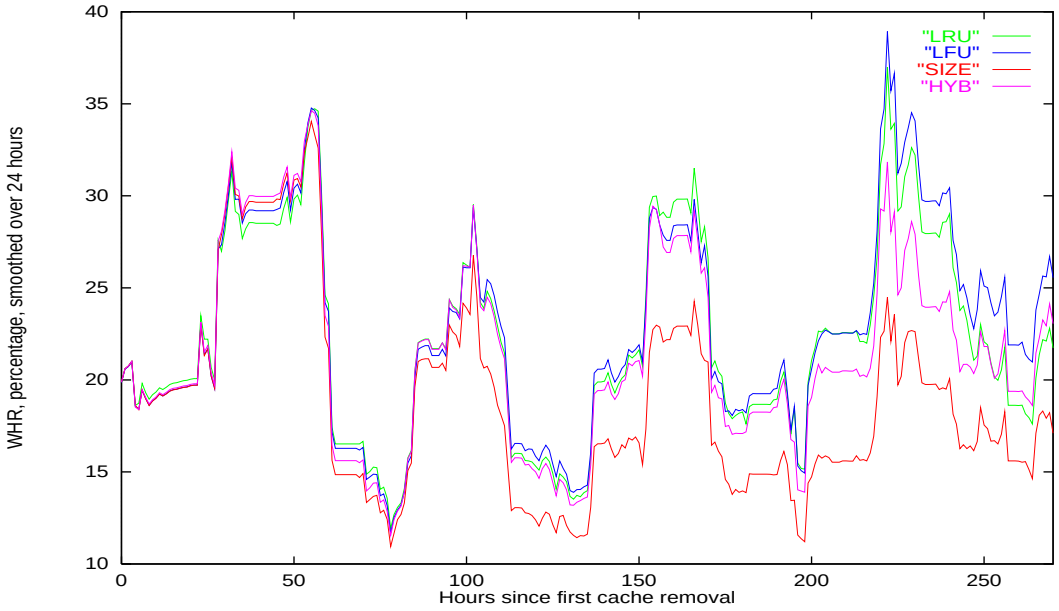

Figure 8.2: Experiment two: Weighted hit rates, VTCS2.

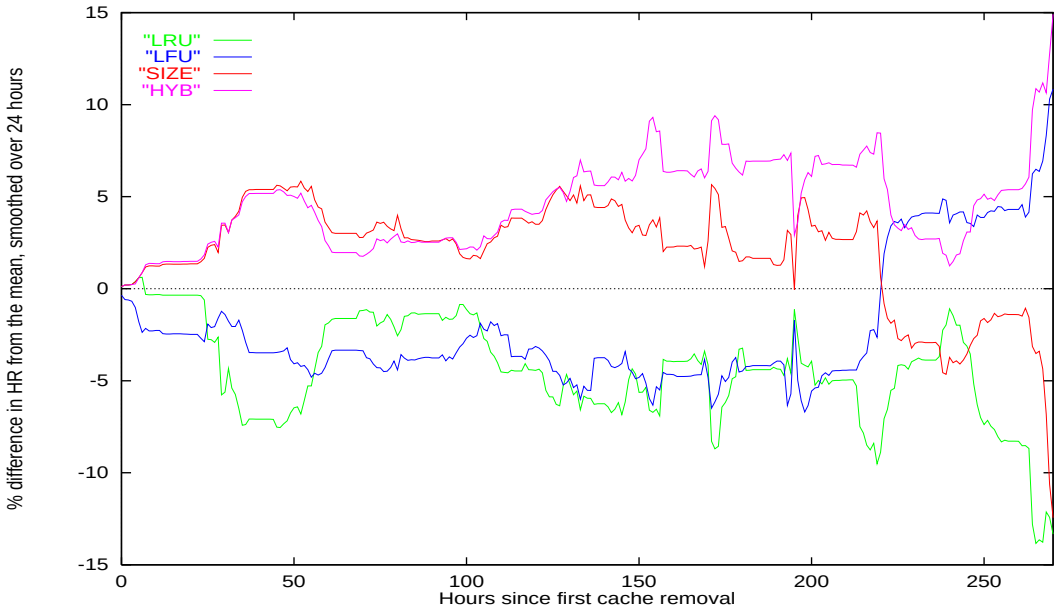

Figure 8.3: Experiment two: Hit rates, normalized, VTCS2.

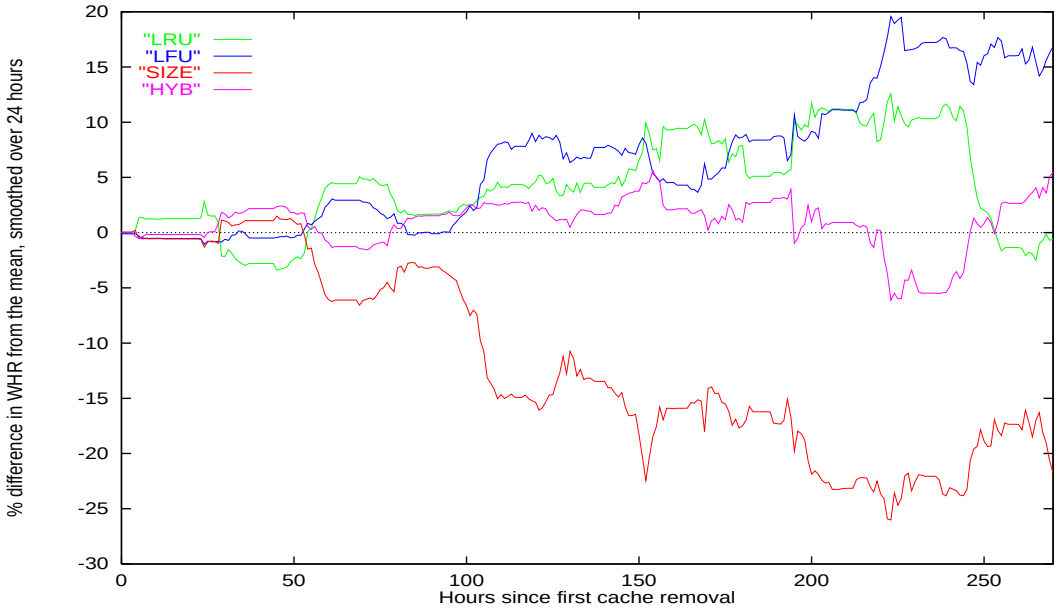

Figure 8.4: Experiment two: Weighted hit rates, normalized, VTCS2.

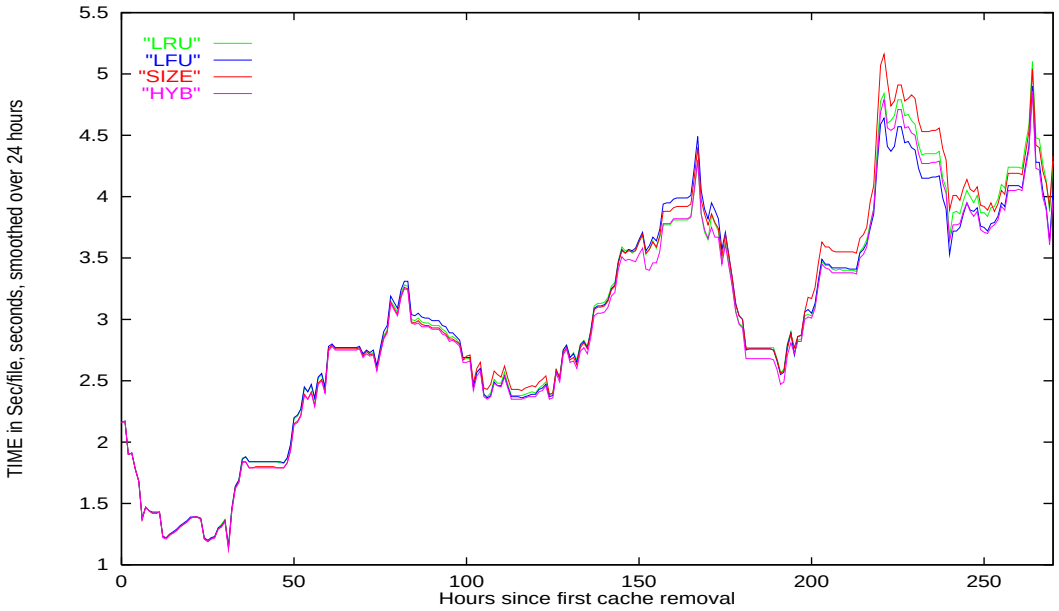

Figure 8.5: Experiment two: Average time to download per file, VTCS2.

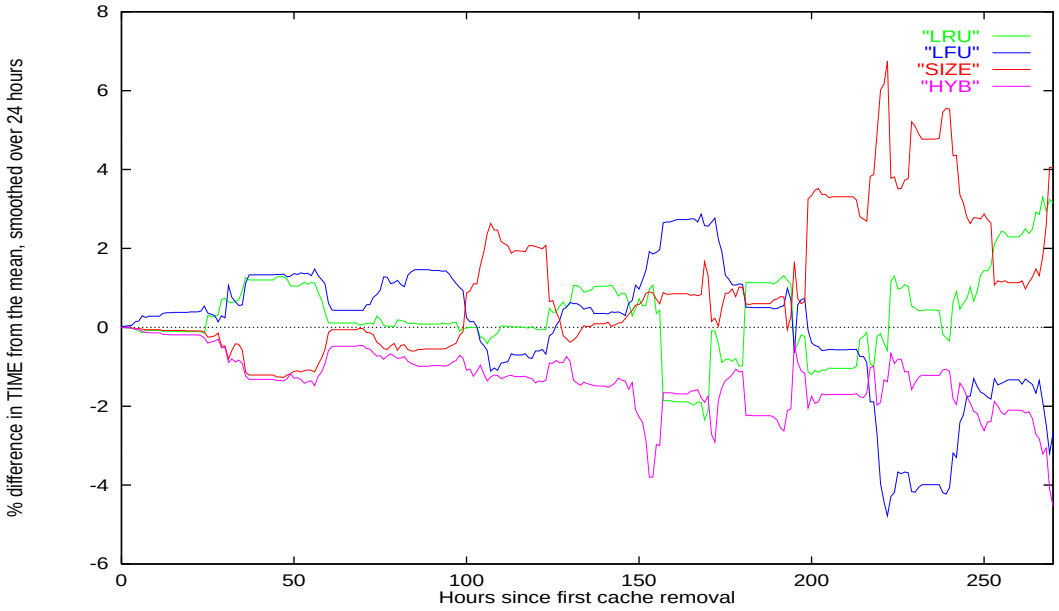

Figure 8.6: Experiment two: Average time to download per file, VTCS2, normalized.

8.3 Results of Part Two — VTCS3 Workload

The following two subsections include graphs from the results of part two, the VTCS3 workload. The first subsection illustrates the results from the HR and WHR, and the second subsection the TIME results.

8.3.1 HR and WHR

As in the previous section graphs of the results of HR (Figure 8.7) and the WHR (Figure 8.8) are shown. Following these normalized graphs of the HR (Figure 8.9) and WHR (Figure 8.10) show the percentage differences between the removal algorithms.

8.3.2 Average Time to Download

Figure 8.11 illustrates the average time per file in seconds. Again the performance degrades over time, and again the graph in Figure 8.12 illustrates the differences between the algorithms.

8.4 Conclusions of Experiment Two

Table 8.3 compares the ICS, IHR, and IWHR to the best values obtained in the four removal algorithms for HR, and WHR. In both parts the cache was set to 50MB and thus the cache size was 11% and 10% of the ICS for the VTCS2 and VTCS3 workloads respectively. The actual values for each removal algorithm are shown in Table 8.2.

Table 8.3: Comparison of the IHR and IWHR with the best HR and WHR.

Workload	ICS (MB)	Cache size (MB)	Cache Size as a % of ICS	IHR %	IWHR %	HR %	WHR %
VTCS2	465	50	11	37	37	27	22
VTCS3	519	50	10	47	42	24	18

The results of this section, and of Chapter 7, illustrate that the performance achieved by the proxy degrades over time. This is possibly due to the fact that the proxy was attempting

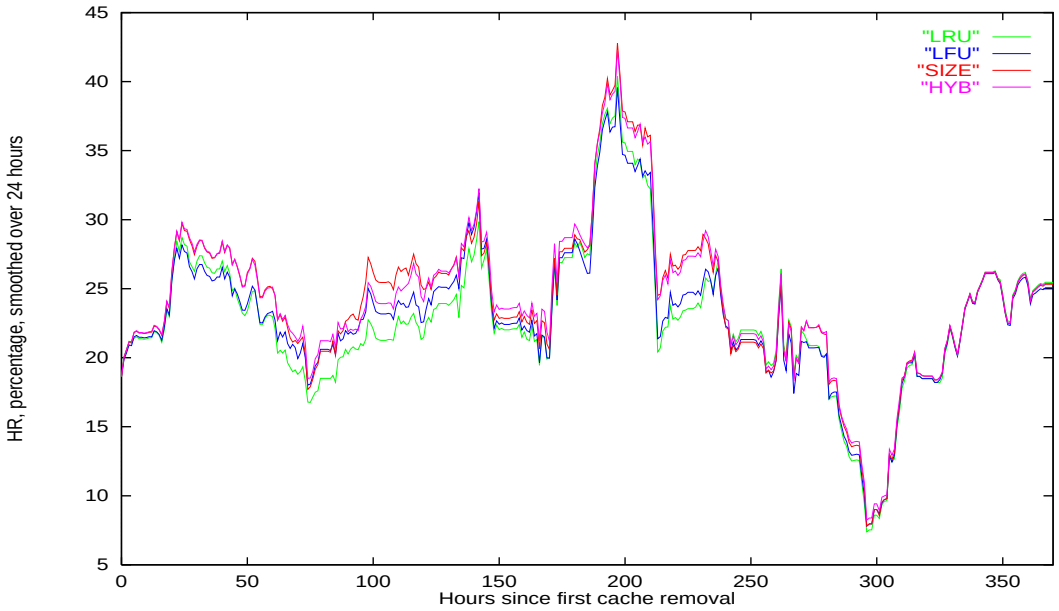

Figure 8.7: Experiment two: Hit rates, VTCS3.

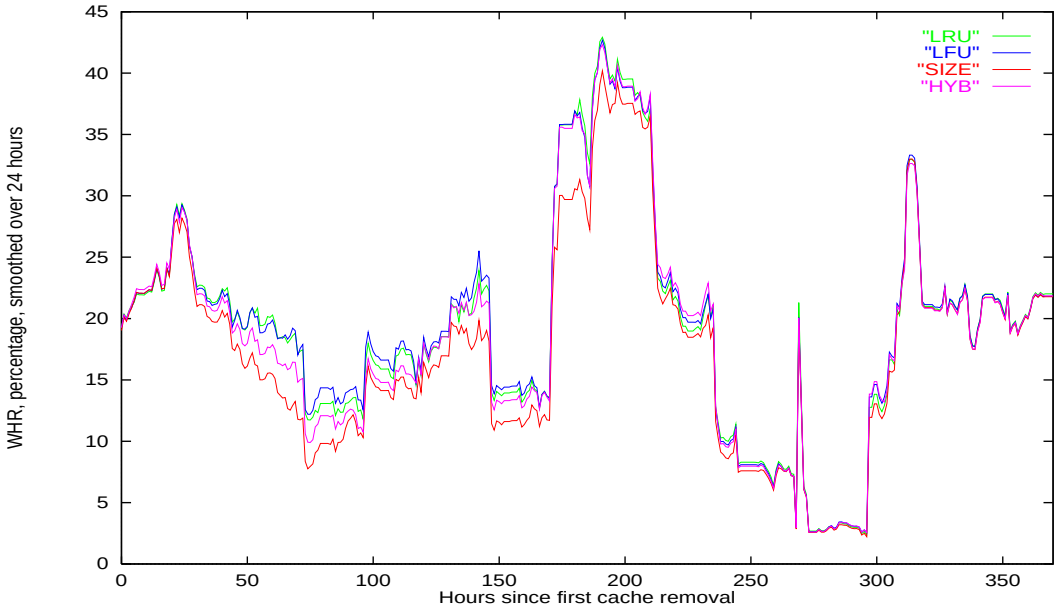

Figure 8.8: Experiment two: Weighted hit rates, VTCS3.

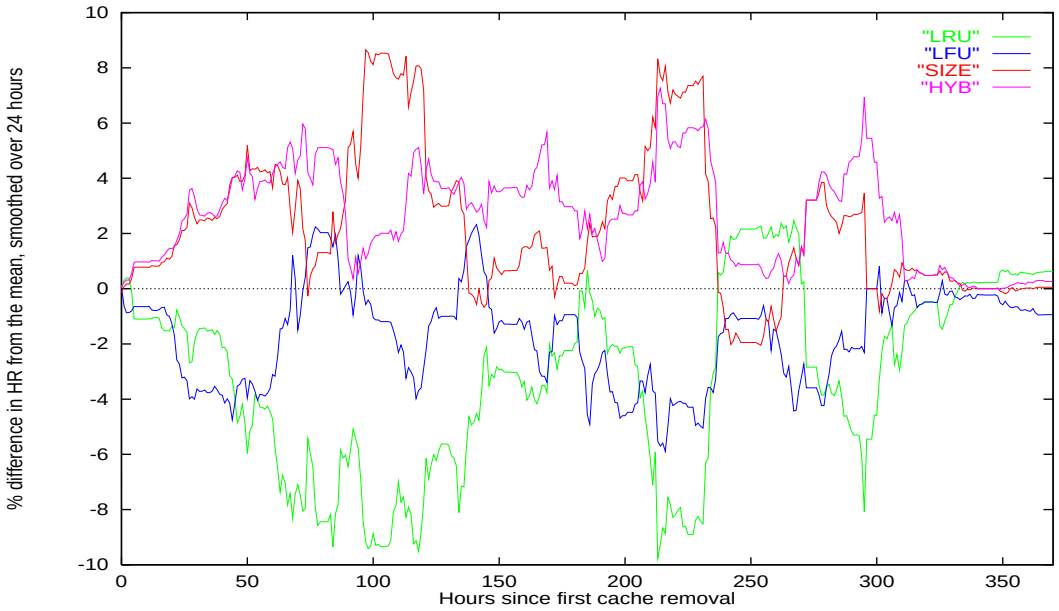

Figure 8.9: Experiment two: Hit rates, normalized, VTCS3.

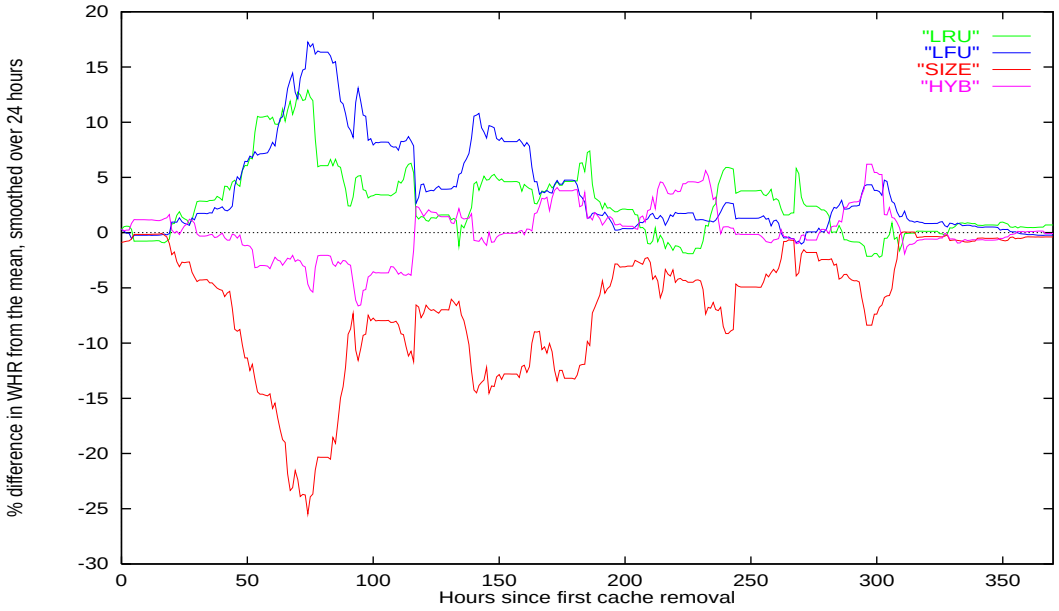

Figure 8.10: Experiment two: Weighted hit rates, normalized, VTCS3.

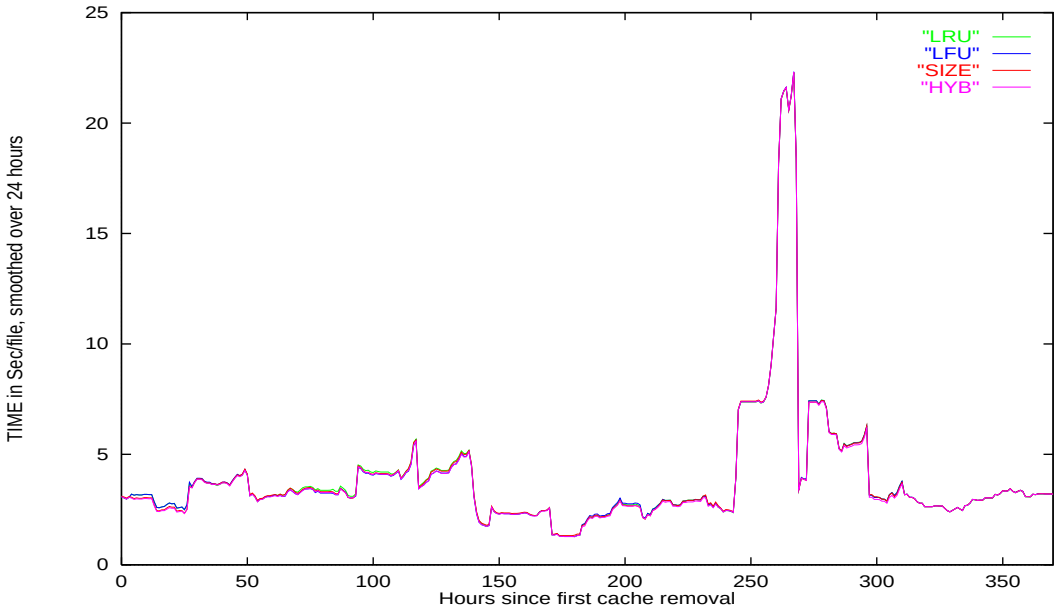

Figure 8.11: Experiment two: Average time to download per file, VTCS3.

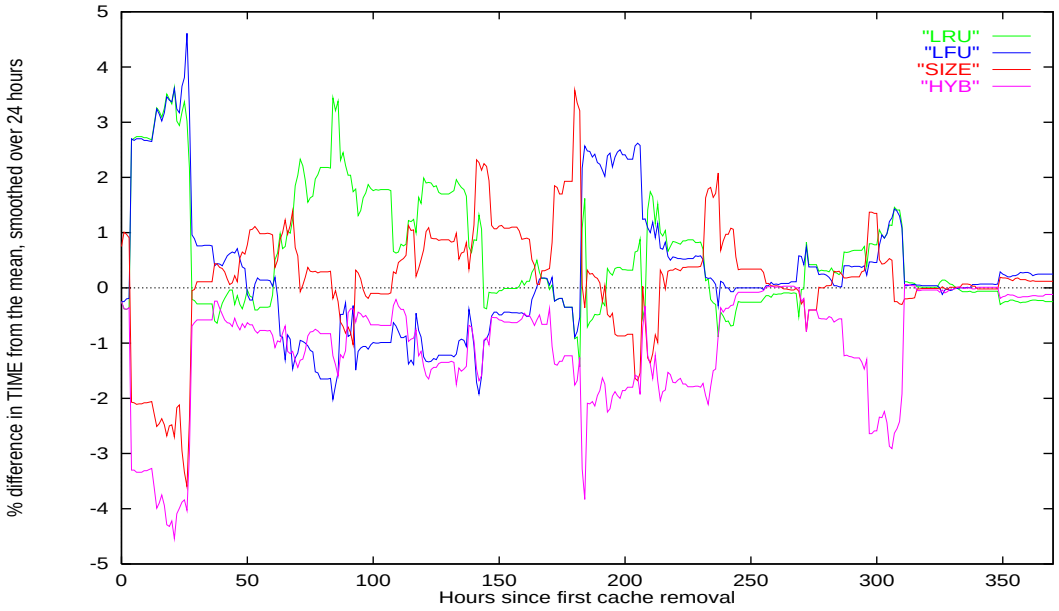

Figure 8.12: Experiment two: Average time to download per file, VTCS3, normalized.

Table 8.4: Confidence interval ranges at the 90% level for VTCS2.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
LRU	(-6.33, -3.23)	-4.78	(1.69, 7.25)	4.47	(-0.37, 1.00)	0.31
LFU	(-3.69, -0.59)	-2.14	(4.13, 9.68)	6.91	(-0.65, 0.73)	0.04
SIZE	(0.44, 3.54)	1.99	(-15.18, -9.62)	-12.40	(0.34, 1.72)	1.03
HYB	(3.37, 6.48)	4.93	(-1.75, 3.81)	1.03	(-2.07, -0.69)	-1.38

Table 8.5: Confidence interval ranges at the 90% level for VTCS3.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
LRU	(-4.23, -2.05)	-3.14	(1.11, 4.55)	2.83	(0.15, 1.04)	0.59
LFU	(-2.92, -0.74)	-1.83	(2.13, 5.58)	3.85	(-0.20, 0.68)	0.24
SIZE	(1.29, 3.47)	2.38	(-8.43, -4.98)	-6.70	(-0.30, 0.58)	0.14
HYB	(1.50, 3.68)	2.59	(-1.71, 1.74)	0.02	(-1.42, -0.53)	-0.97

to simulate all four algorithms from one executable. Unfortunately an enormous amount of disk thrashing occurred. At some points the thrashing became so bad that the proxy could not recover and had to be terminated. This is *not* a problem that the proxy experiences when running in "single removal algorithm" mode, but is an artifact of our experimental method. Although the performance degrades over time the relative performance of the removal algorithms is not affected. Chapter 9 illustrates the amazing performance achievable by the proxy when running in "single removal algorithm" mode.

The results, shown in shown in Table 8.2, do, however, illustrate the effectiveness of the HYB algorithm. As intended the HYB algorithm shows that it performs with the lowest latency of all the removal algorithms tested. LFU and LRU do very well for WHR, and for HR the HYB and SIZE algorithms are the best. Chapter 9 discusses further tests performed on the HYB algorithm.

The values in Tables 8.4 and 8.5 are used to generate Table 8.6 which shows which

Table 8.6: Illustration of the significantly better removal algorithms in experiment two, using 90% confidence intervals.

Workload	Measure	Best	Medium	Worst
VTCS2	HR	HYB/SIZE		LFU/LRU
VTCS2	WHR	LFU/LRU	LRU/HYB	SIZE
VTCS2	TIME	HYB		LFU/LRU/SIZE
VTCS3	HR	HYB/SIZE		LFU/LRU
VTCS3	WHR	LFU/LRU	LRU/HYB	SIZE
VTCS3	TIME	HYB		SIZE/LFU/LRU

algorithms are significantly superior (at the 90% level) for each type of performance measure: HR, WHR, and TIME. The actual raw data for each algorithm is given in the tables in Appendices C.1 and C.2. A data point was taken to be the average of 24 hours; only data after the cache had been filled was used. The removal algorithms are sorted in Table 8.6 from best to worst (left to right), for each performance measure. For example: for WHR in VTCS2, the best policy is LFU, although this is not significantly different than LRU. The next best is LRU and HYB, which are not significantly different, although the ordering implies that LRU has a higher mean than HYB. The layout of the table indicates that although LRU is not significantly different from LFU and HYB, LFU and HYB are significantly different with LFU being better. The next cell in the table is SIZE, which is significantly worse than all three of the other algorithms. To take another example: for TIME in VTCS2, here the HYB algorithm is significantly superior than all the others, and the other three are not significantly different from each other.

The goal of the research has been achieved. The HYB algorithm is the highest performer for reduction of retrieval times for both of the tests. Not only is it the best performer for the retrieval times, but also the best performer for HR. LFU and LRU are the best algorithms for WHR, and the HYB algorithm is significantly better than the SIZE algorithm. The next chapter shows the performance of the HYB algorithm with different workloads using a different experimental design.

Chapter 9

Experiment Three: Accelerated Log Files

This chapter discusses the design, results and conclusions of the third experiment. The results and conclusions, because of the large number of graphs, are split into four sections. After the experimental design the next two sections present graphs of the results of the comparisons between the removal algorithms, and the hybrid algorithms with different constants. The next section determines which algorithms and constant combinations are significantly better than the others. Finally conclusions are drawn from the graphs and significance test sections.

9.1 Experimental Design

This experiment uses a different machine for the proxy and a different approach to evaluating performance. This section discusses the design of the third experiment; it uses the algorithms: LRU, LFU, SIZE, and HYB. This experiment uses the VTCS2, BULOG, and VTLIB workloads, collected at different times and from different sources, and plays them through four "single removal algorithm mode" proxies, running in parallel, at a highly accelerated rate. In this manner we are able to test large log files in relatively short periods of time.

In our experiment, four proxies are started on the 4-processor RS6000. A test application called "WebJamma"[1] is used to re-play the logs at high speeds to the parallel proxies. WebJamma works by spawning 40 child processes, each given an child-id (CID) from 0 to 39, then reading the log file one URL at a time. Forty is used rather than four to increase

[1]WebJamma was written by Tommy Johnson, tjohnson@csgrad.cs.vt.edu, and is available from "http://-www.cs.vt.edu/~chitra/www.html".

the number of connections that the proxy opens simultaneously, thus each proxy will on average have 10 HTTP connections open at all times. The URLs are put in a pipe. Each of the 40 children processes of WebJamma take a URL from the pipe when they are free. CID sends URLs it takes from the pipe to proxy number (PrN) PrN=(1+(CID mod 4)) first and then to the other three in order, the order being defined by the proxy number. In this manner each of the proxies receive a URL request first, second, third, and fourth for exactly a quarter of the URLs in the log file. This, on average, removes any unfair disadvantage to a proxy for having to fetch the URL first. There are many reasons why it might be more costly to be the first proxy to have to retrieve a URL before the other proxies. One such reason is that network and server caching will not help the first proxy, but may reduce retrieval times for the others.

The machine used in this experiment is the IBM RS6000, described in section 5.1. This machine is fast enough to run four totally independent proxies in parallel. We also were able to play the log files through at very highly accelerated rates. The proxy was not the limiting factor on the performance; the retrieval times from the Internet is the performance bottleneck. The load exerted on the machine was almost negligible[2].

Due to the accelerated speed at which the log files are "re-played" we shortened the TTL of the documents within the cache, so as to simulate a similar expiry removal policy. The acceleration of the proxy is roughly 50 fold real time; the TTL values illustrated in Table 6.1 were consequently reduced 50 fold[3]. The factor of 50 is not critically important, the relative performance of the removal algorithms will still be identical. The TTL times are decreased to 1% so that the results of the experiment will be similar, but not identical, to those of experiment two (Chapter 8). Other configuration parameters, such as cache size, and the upper and lower cache thresholds were set as described in section 6.2.

Each of the three different log files, VTCS2, BULOG, and VTLIB, used in this experiment is first used to drive the parallel proxies, which each run one of the removal algorithms

[2]The load was noted occasionally by using the UNIX command *uptime*.

[3]The two week log from experiment two, test one, took only seven hours to run in this experiment.

Table 9.1: Factors of the $2^{k-p}r$ design test.

Symbol	Confounding	Factor	Level -1	Level +1
W_B	W_N· CONN	CLAT vs. CBW emphasis	8KB	16KB
W_N	W_B· CONN	NREF vs. size emphasis	0.9	1.1
CONN	$W_B \cdot W_N$	Connection Packet Size	512B	2048B

Table 9.2: Levels for the $2^{k-p}r$ design test.

Experiment	W_B	W_N	CONN
1	-1	-1	1
2	1	-1	-1
3	-1	1	-1
4	1	1	1

(LRU, LFU, SIZE, and HYB). Then the log file is tested with another four parallel proxies, each running the HYB algorithm but with four different sets of constants (W_B, W_N, and CONN) in the HYB formula.

A $2^{k-p}r$ [15] design was used to test the performance of the HYB algorithm with different values for each of the three constants used in the removal algorithm, shown in equations (4.2, 4.3, and 4.6). Four proxies running in parallel on the RS6000 enabled us to use a $2^{3-1}r$ experimental design, a half replicate three factor test, using replicas. Each successive set of 5000 URLs in a log file beyond the point at which the cache fills up for the first time, is used as a single replica. Table 9.1 illustrates the three constants used with the HYB algorithm, their symbols, levels and their confoundings. Table 9.2 illustrates how the levels are set for the four tests. This design is an $I = ABC$ design, meaning the resolution is R_{III}, the highest quality of design for a $2^{3-1}r$ test. Table 9.3 presents the mean HR, WHR, and TIME for each removal algorithm for the three workloads.

The measured response for each response variable (HR, WHR, and TIME) is calculated as a percentage of the average of the four different algorithms. The accelerated proxies complete the log files of 45,000 to 85,000 URL requests in 3 to 10 hours, depending on the log file length and the percentage hit rate. The proxies record their activity every two minutes; these data records are used in plotting the graphs in this chapter.

Table 9.3: Mean HR, WHR, and TIME values for experiment three

Workload	Algorithm	HR %	WHR %	TIME Sec/File
VTCS2	LRU	34.7	23.5	2.00
VTCS2	LFU	33.8	22.6	2.03
VTCS2	SIZE	36.2	19.6	1.97
VTCS2	HYB	35.8	20.6	1.96
BULOG	LRU	66.5	53.2	0.37
BULOG	LFU	66.2	50.8	0.36
BULOG	SIZE	67.1	49.6	0.33
BULOG	HYB	67.0	51.0	0.34
VTLIB	LRU	31.2	28.2	1.04
VTLIB	LFU	30.6	27.6	1.05
VTLIB	SIZE	32.1	26.4	1.04
VTLIB	HYB	32.0	28.1	1.05

9.2 Results: LRU, LFU, SIZE, and HYB Test

This section describes the results of the performance test of LRU, LFU, SIZE, and HYB on the log files, VTCS2, BULOG, and VTLIB. In this experiment the values of W_B, W_N, and CONN were fixed at 8KB, 0.9, and 2KB respectively. The results, of the HYB constants test, indicate that a value of 1.1 might have been a better choice for W_N. However, the difference is marginal. Graphs of the performance measures, HR, WHR, and TIME, are illustrated along with normalized graphs of the same data. The normalized graphs illustrate the percentage difference from the mean, at each data point, for each of removal algorithms. Table 9.4 lists which graphs illustrate which results.

Table 9.4: Experiment three, removal algorithm test, test results and graph numbers.

Log file	Performance measure	Smoothed (S) or smoothed & normalized (SN)	Figure number
VTCS2	HR	S	9.1
VTCS2	HR	SN	9.2
VTCS2	WHR	S	9.3
VTCS2	WHR	SN	9.4
VTCS2	TIME	S	9.5
VTCS2	TIME	SN	9.6
BULOG	HR	S	9.7
BULOG	HR	SN	9.8
BULOG	WHR	S	9.9
BULOG	WHR	SN	9.10
BULOG	TIME	S	9.11
BULOG	TIME	SN	9.12
VTLIB	HR	S	9.13
VTLIB	HR	SN	9.14
VTLIB	WHR	S	9.15
VTLIB	WHR	SN	9.16
VTLIB	TIME	S	9.17
VTLIB	TIME	SN	9.18

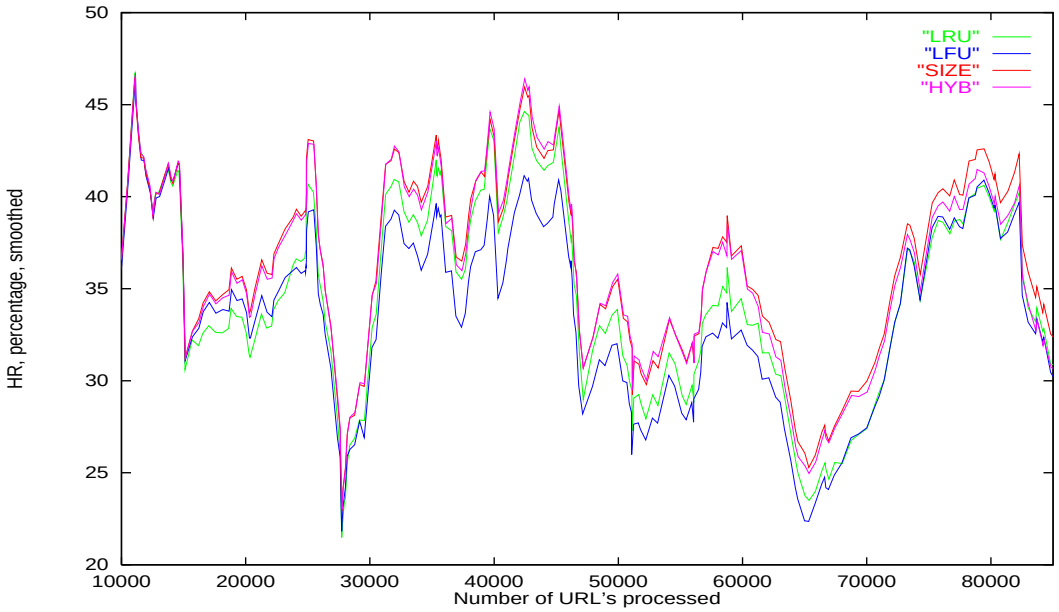

Figure 9.1: Experiment three, VTCS2, Removal Algorithms, HR.

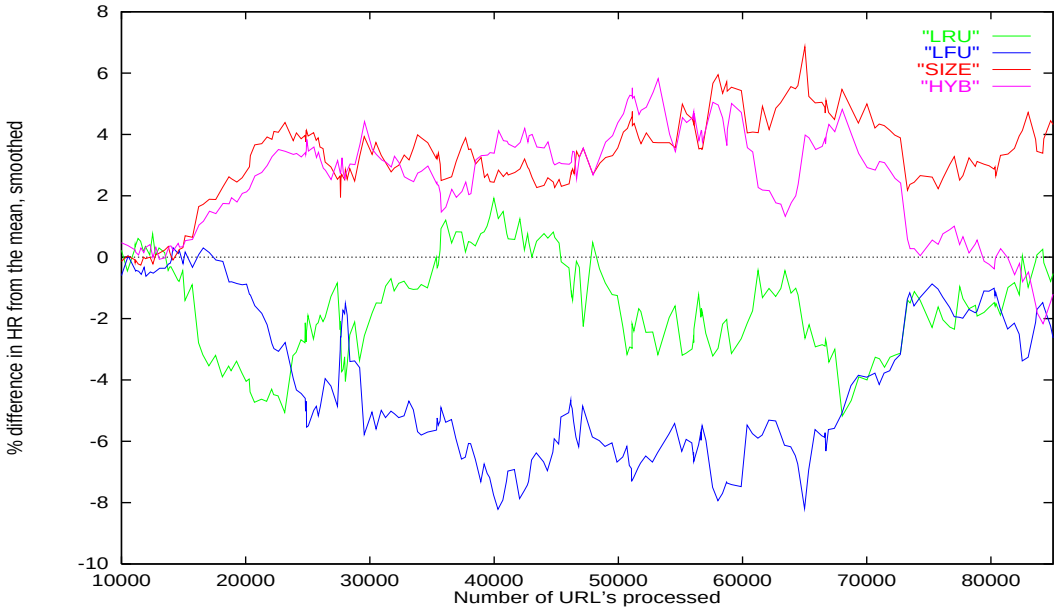

Figure 9.2: Experiment three, VTCS2, Removal Algorithms, Normalized, HR.

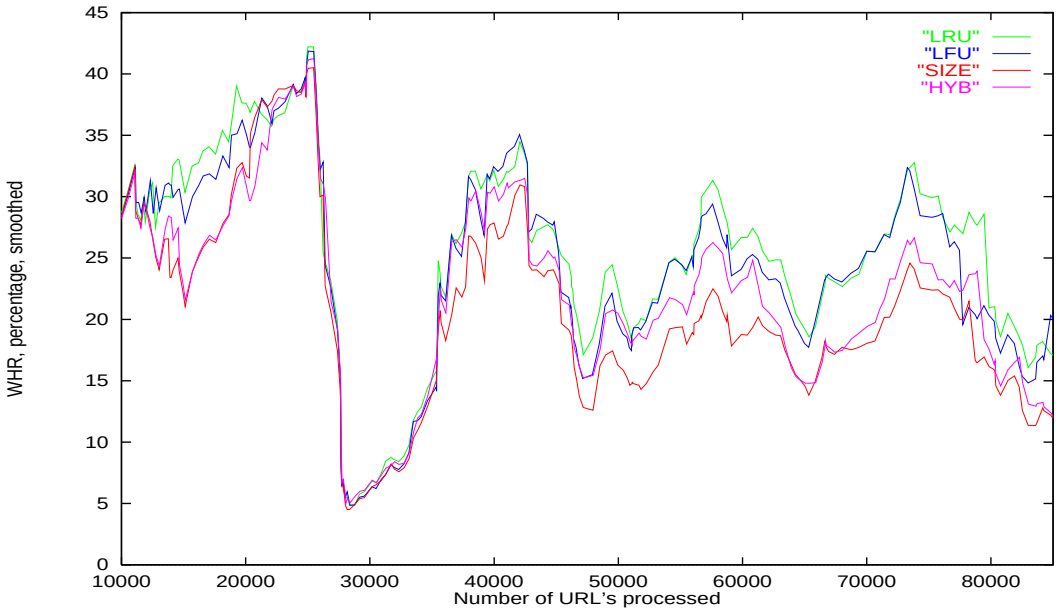

Figure 9.3: Experiment three, VTCS2, Removal Algorithms, WHR.

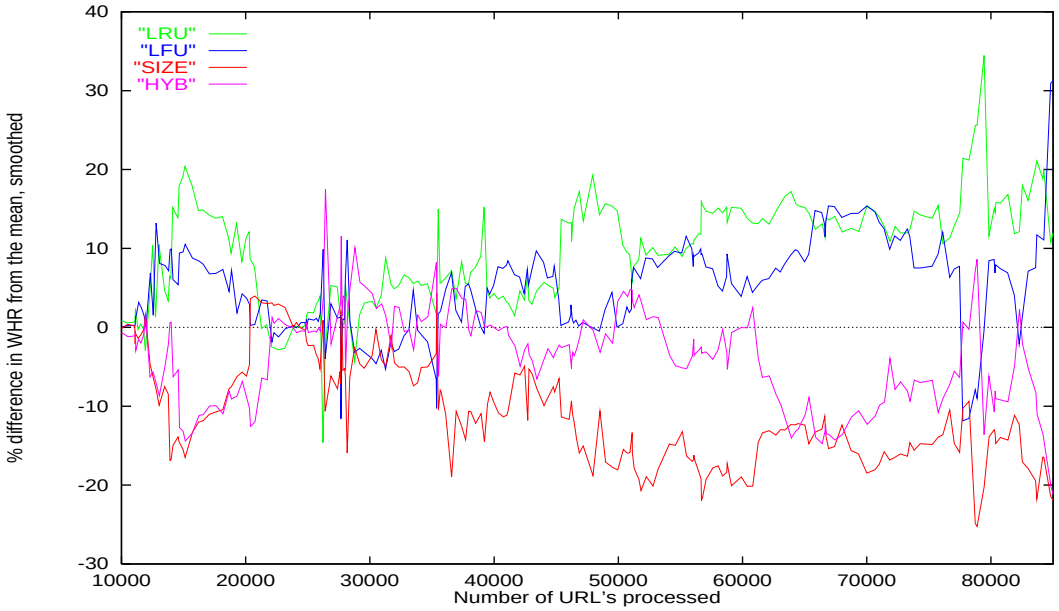

Figure 9.4: Experiment three, VTCS2, Removal Algorithms, Normalized, WHR.

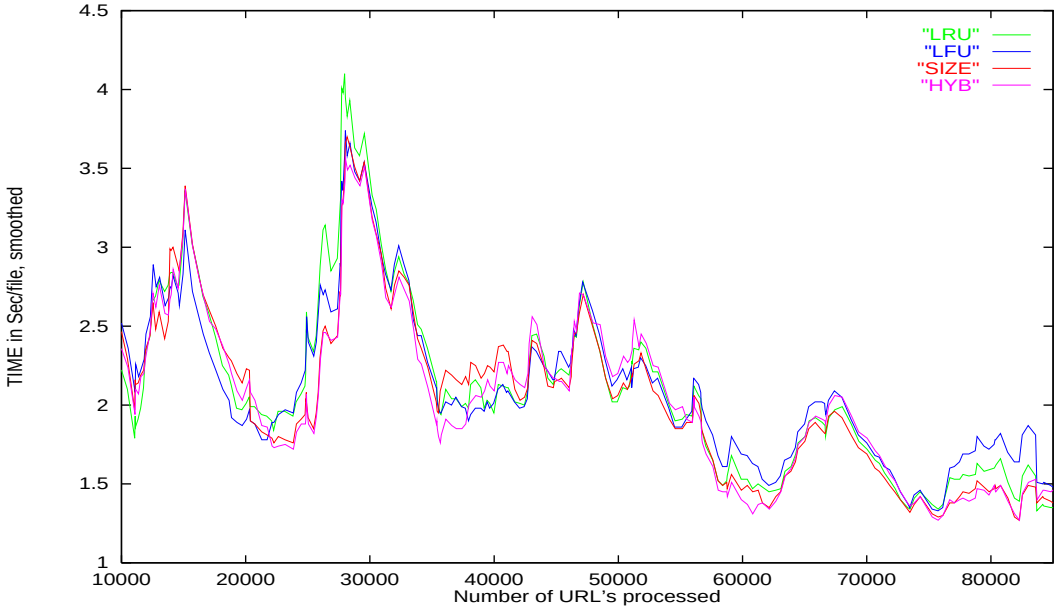

Figure 9.5: Experiment three, VTCS2, Removal Algorithms, TIME.

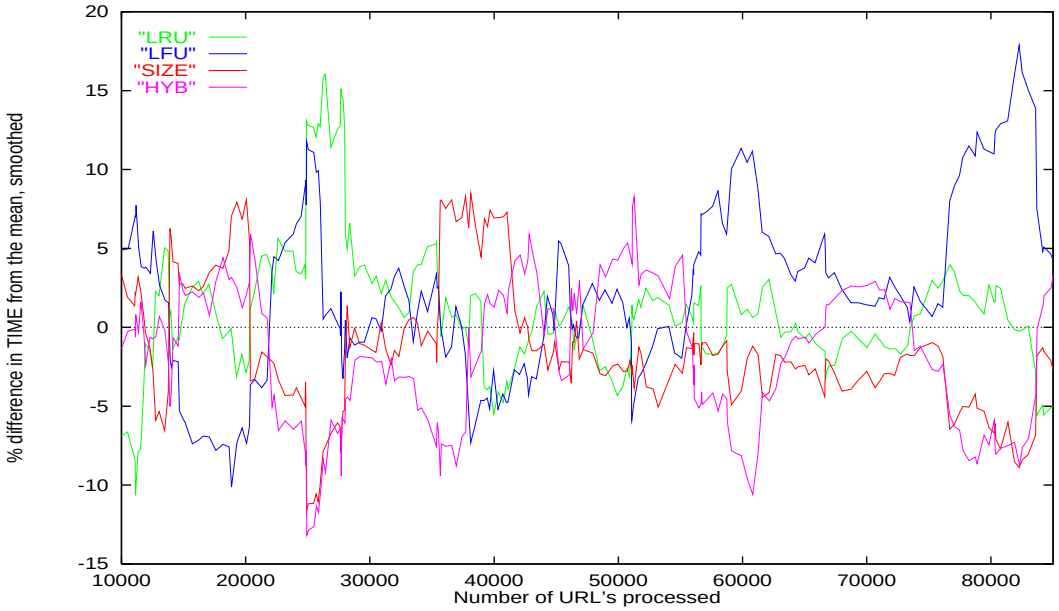

Figure 9.6: Experiment three, VTCS2, Removal Algorithms, Normalized, TIME.

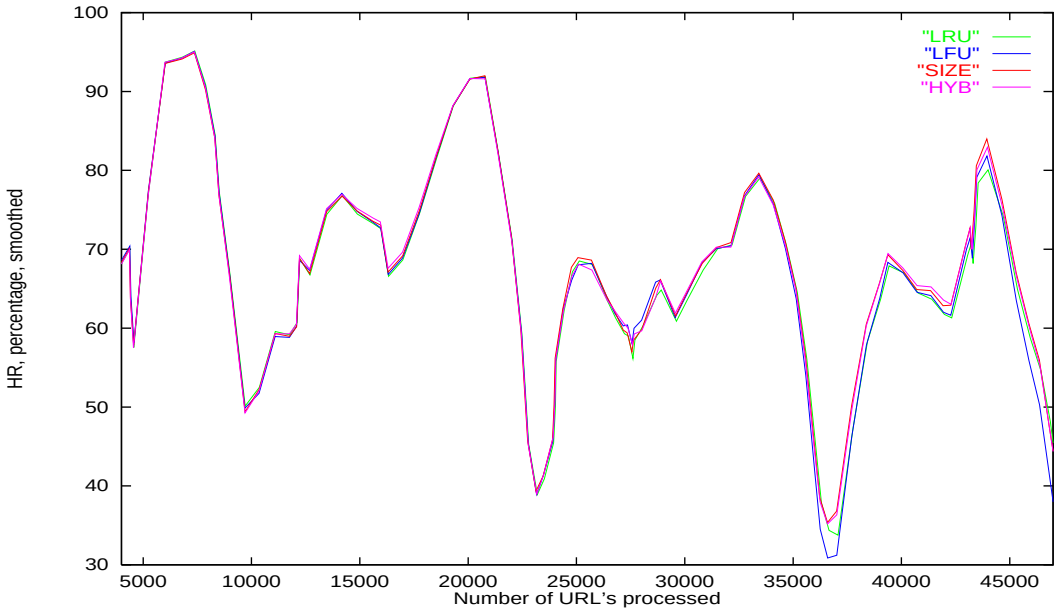

Figure 9.7: Experiment three, BULOG, Removal Algorithms, HR.

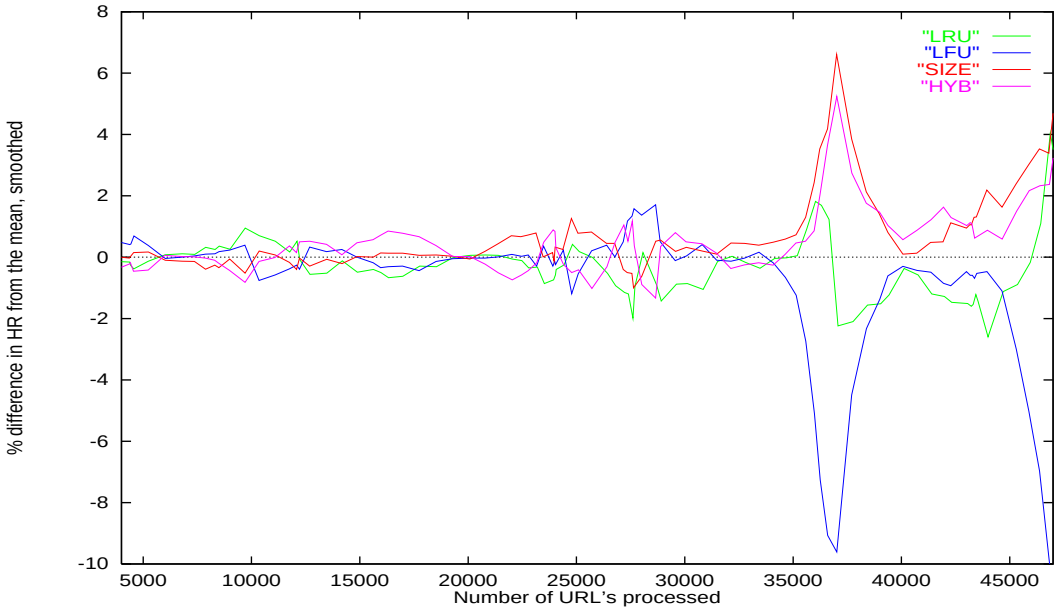

Figure 9.8: Experiment three, BULOG, Removal Algorithms, Normalized, HR.

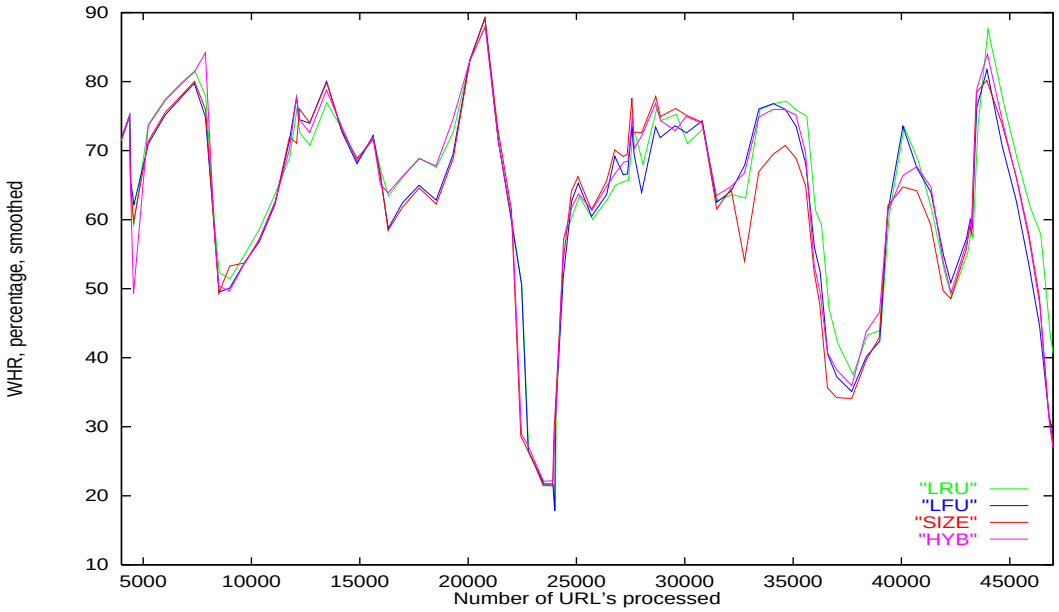

Figure 9.9: Experiment three, BULOG, Removal Algorithms, WHR.

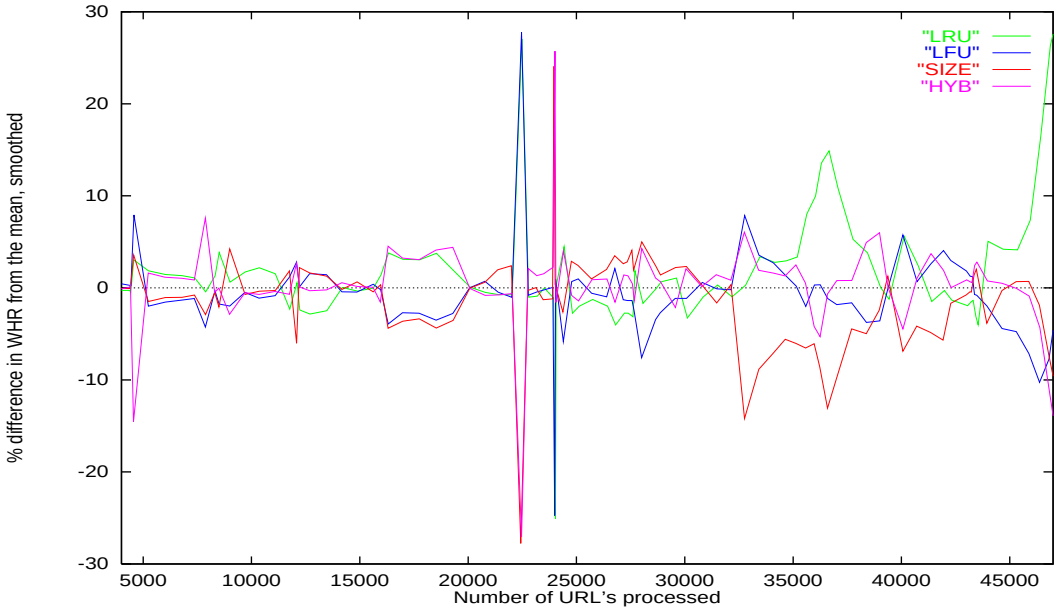

Figure 9.10: Experiment three, BULOG, Removal Algorithms, Normalized, WHR.

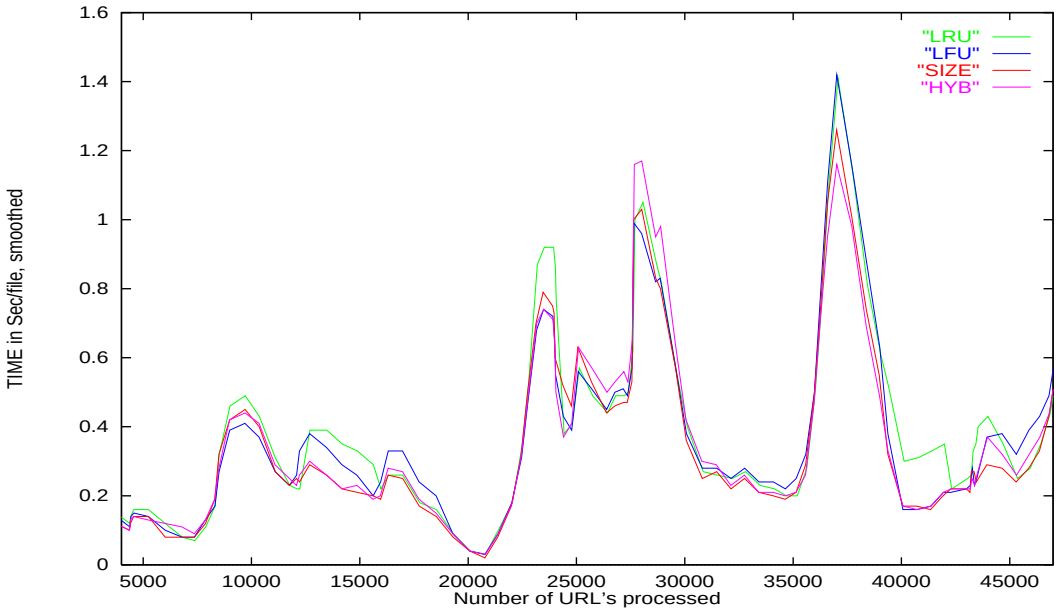

Figure 9.11: Experiment three, BULOG, Removal Algorithms, TIME.

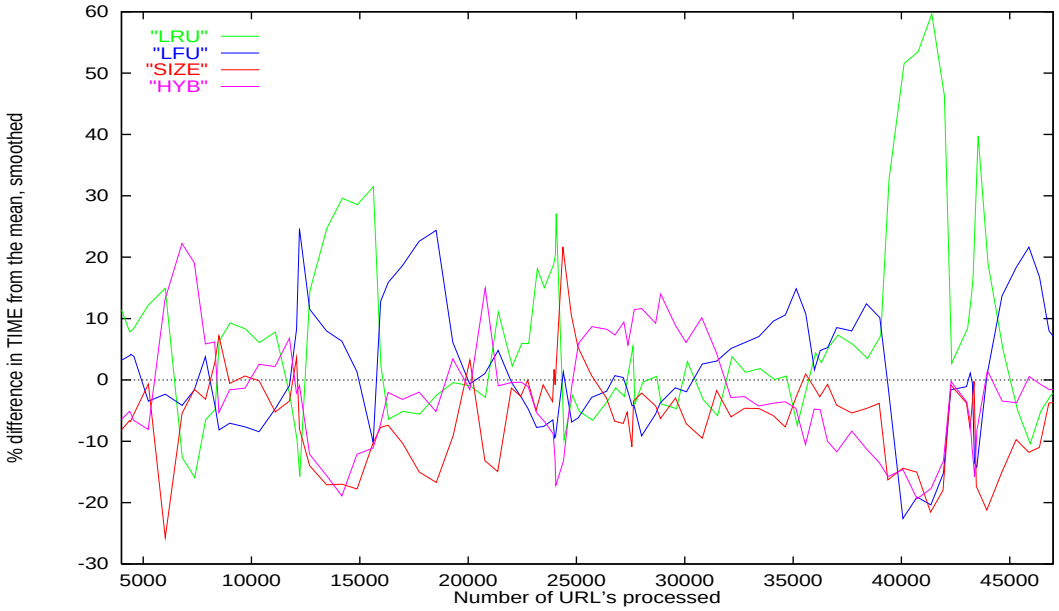

Figure 9.12: Experiment three, BULOG, Removal Algorithms, Normalized, TIME.

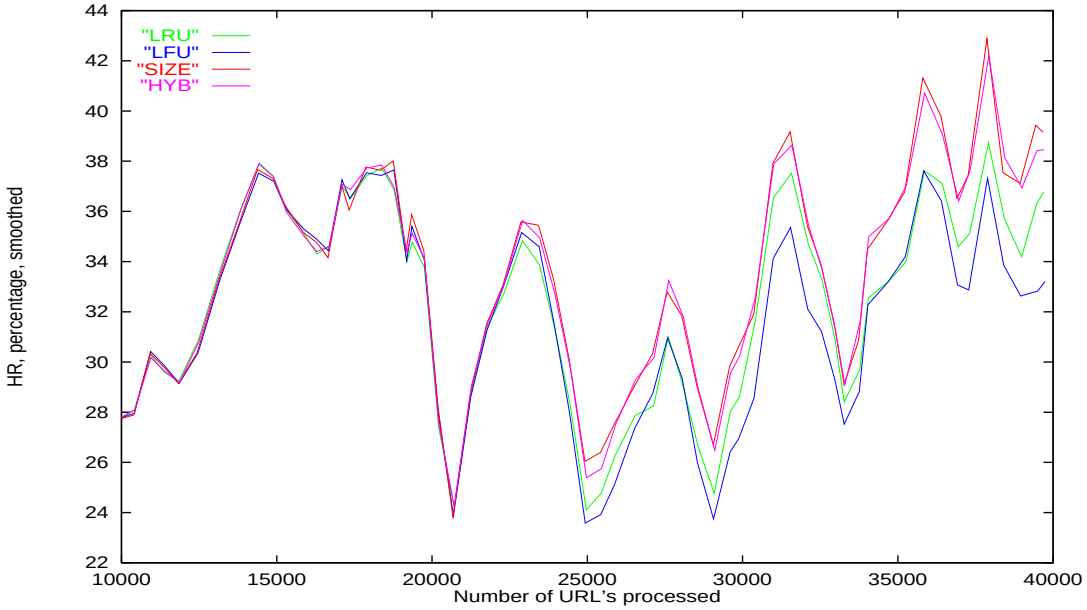

Figure 9.13: Experiment three, VTLIB, Removal Algorithms, HR.

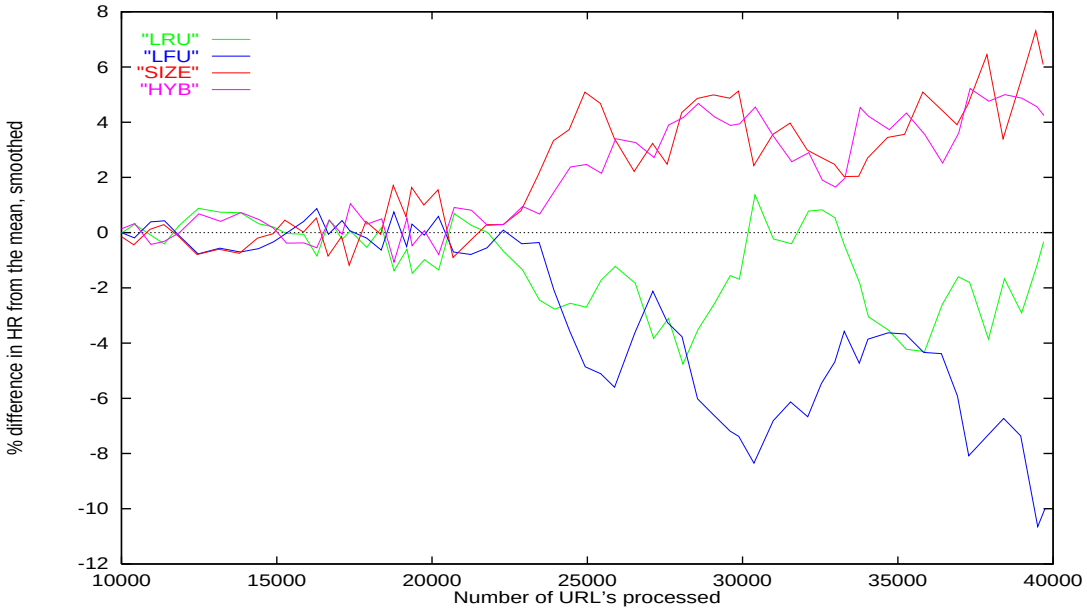

Figure 9.14: Experiment three, VTLIB, Removal Algorithms, Normalized, HR.

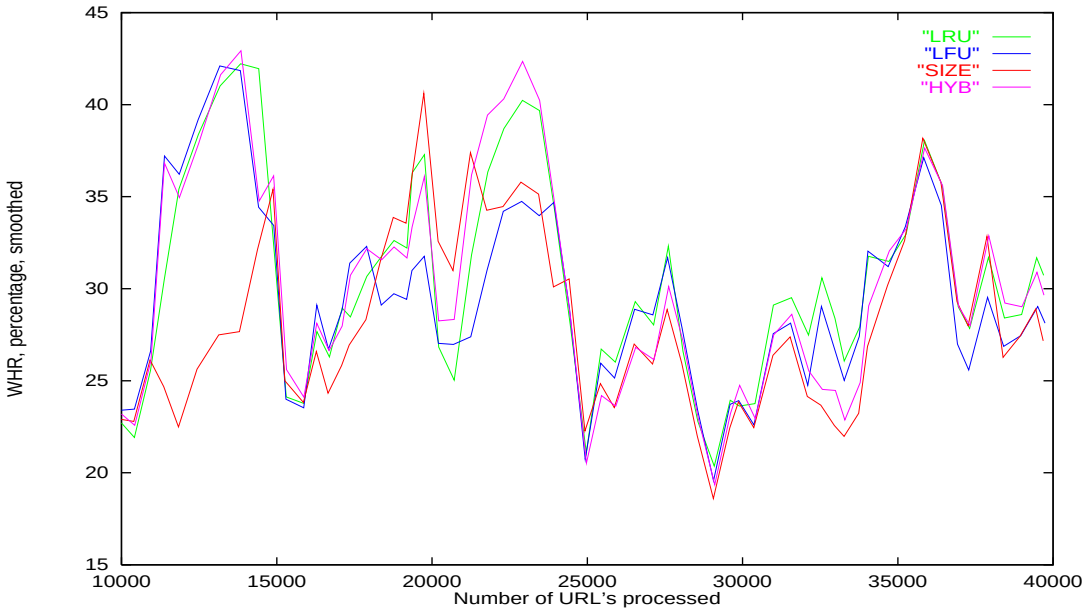

Figure 9.15: Experiment three, VTLIB, Removal Algorithms, WHR.

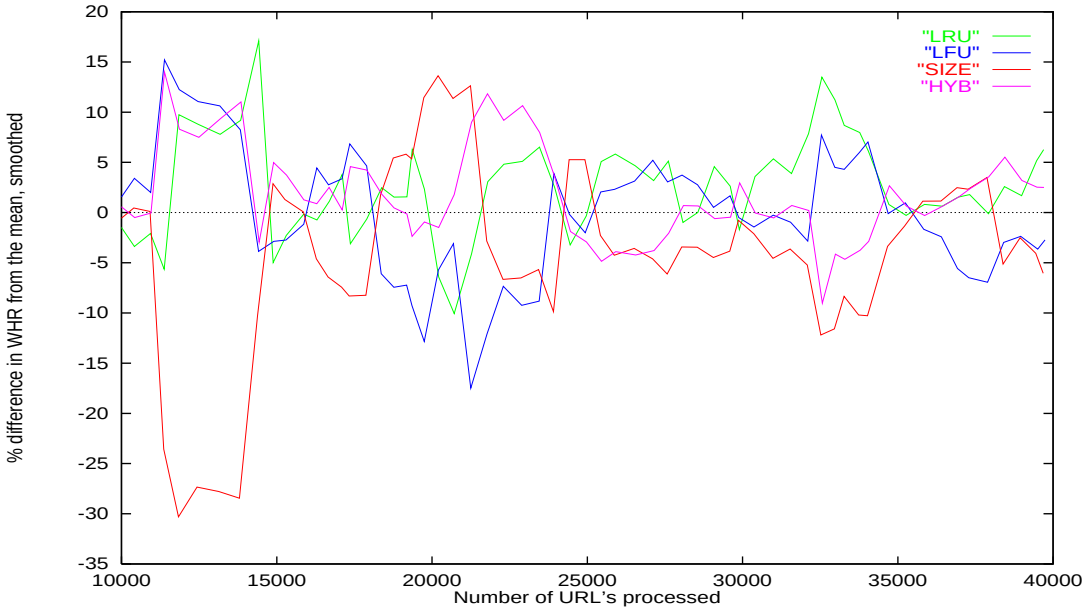

Figure 9.16: Experiment three, VTLIB, Removal Algorithms, Normalized, WHR.

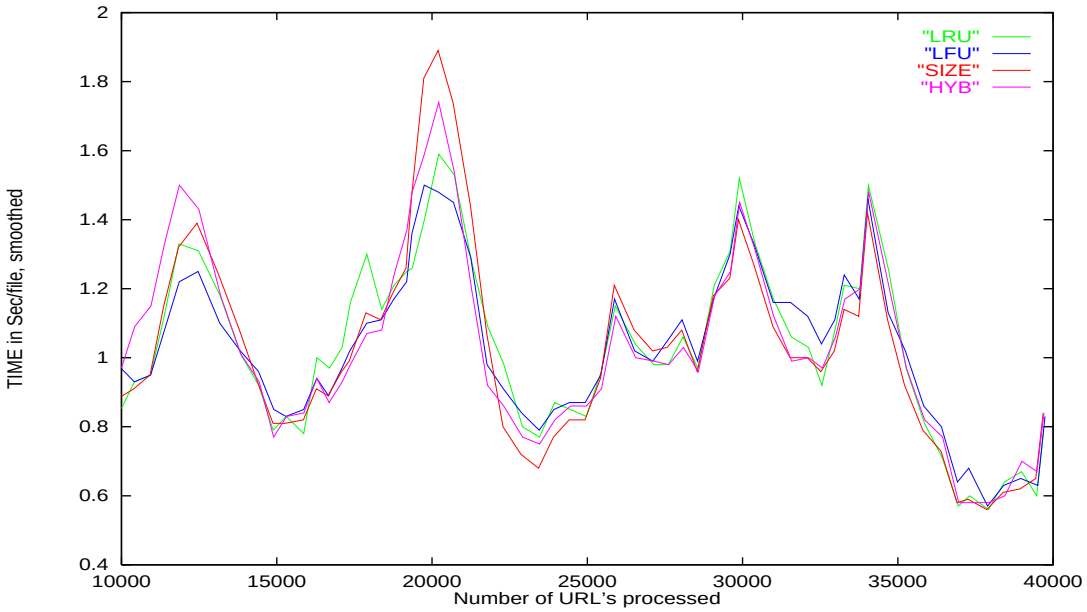

Figure 9.17: Experiment three, VTLIB, Removal Algorithms, TIME.

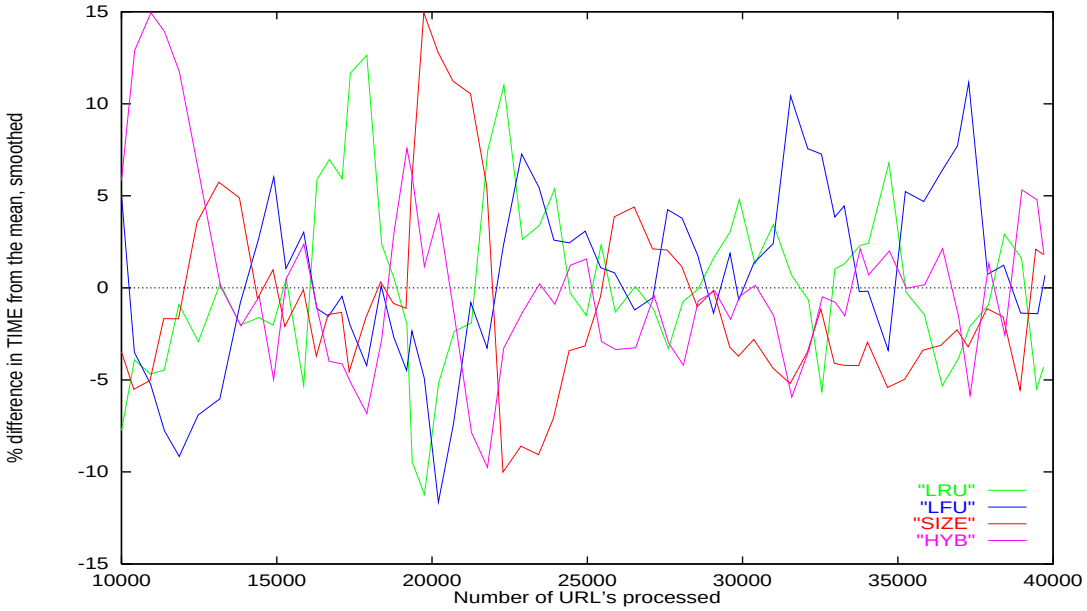

Figure 9.18: Experiment three, VTLIB, Removal Algorithms, Normalized, TIME.

Table 9.5: Experiment three, HYB constant test, test results and graph numbers.

Log file	Performance measure	Smoothed (S) or smoothed& normalized (SN)	Figure number
VTCS2	HR	S	9.19
VTCS2	HR	SN	9.20
VTCS2	WHR	S	9.21
VTCS2	WHR	SN	9.22
VTCS2	TIME	S	9.23
VTCS2	TIME	SN	9.24
BULOG	HR	S	9.25
BULOG	HR	SN	9.26
BULOG	WHR	S	9.27
BULOG	WHR	SN	9.28
BULOG	TIME	S	9.29
BULOG	TIME	SN	9.30
VTLIB	HR	S	9.31
VTLIB	HR	SN	9.32
VTLIB	WHR	S	9.33
VTLIB	WHR	SN	9.34
VTLIB	TIME	S	9.35
VTLIB	TIME	SN	9.36

9.3 Results: HYB algorithm — Constants Test

This section describes the results of the performance test of different constants within the HYB algorithm; all four proxies use the HYB algorithm but with different constants. As with the previous section graphs of the performance measures, HR, WHR, and TIME, are illustrated along with normalized graphs of the same data. Table 9.5 lists which graphs illustrate which results.

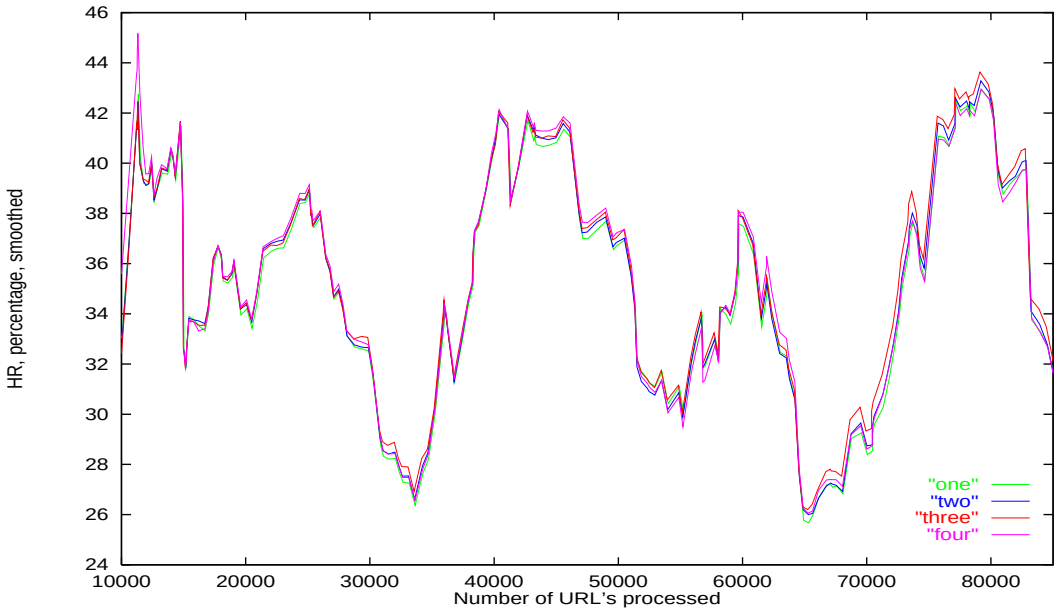

Figure 9.19: Experiment three, VTCS2, Hybrid Algorithms, HR.

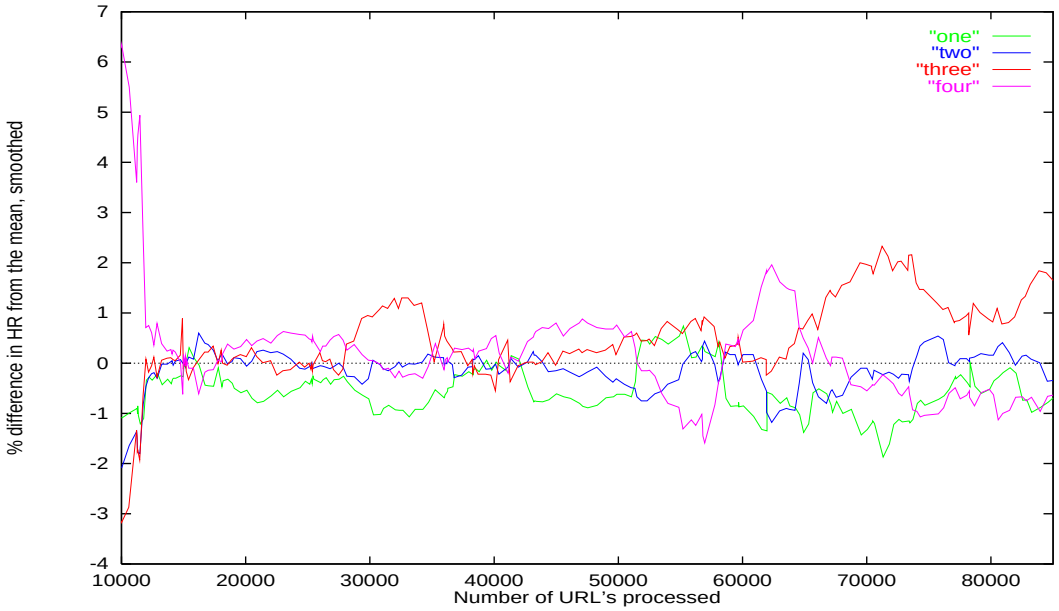

Figure 9.20: Experiment three, VTCS2, Hybrid Algorithms, Normalized, HR.

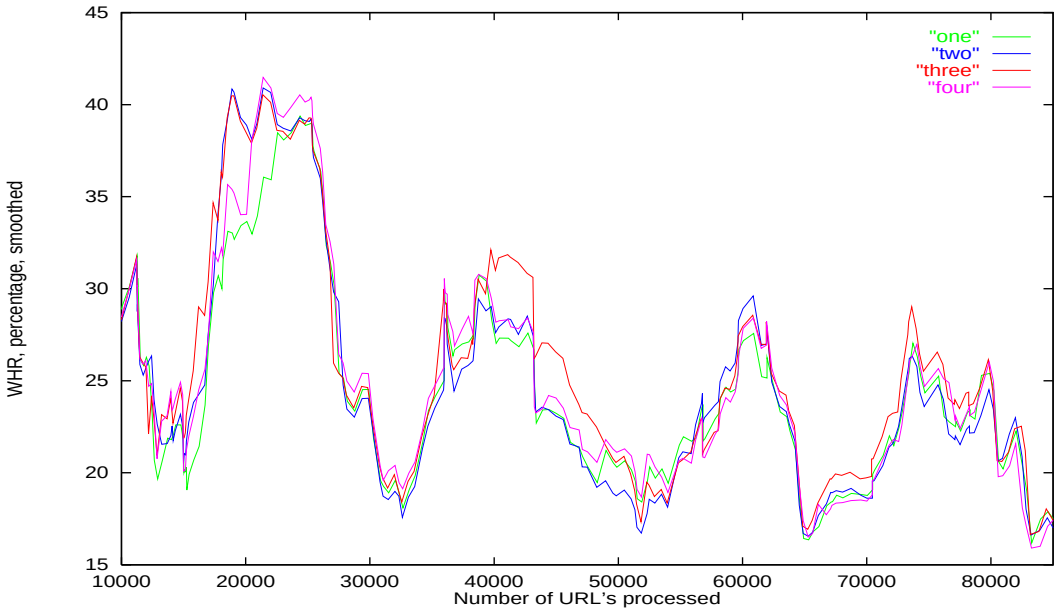

Figure 9.21: Experiment three, VTCS2, Hybrid Algorithms, WHR.

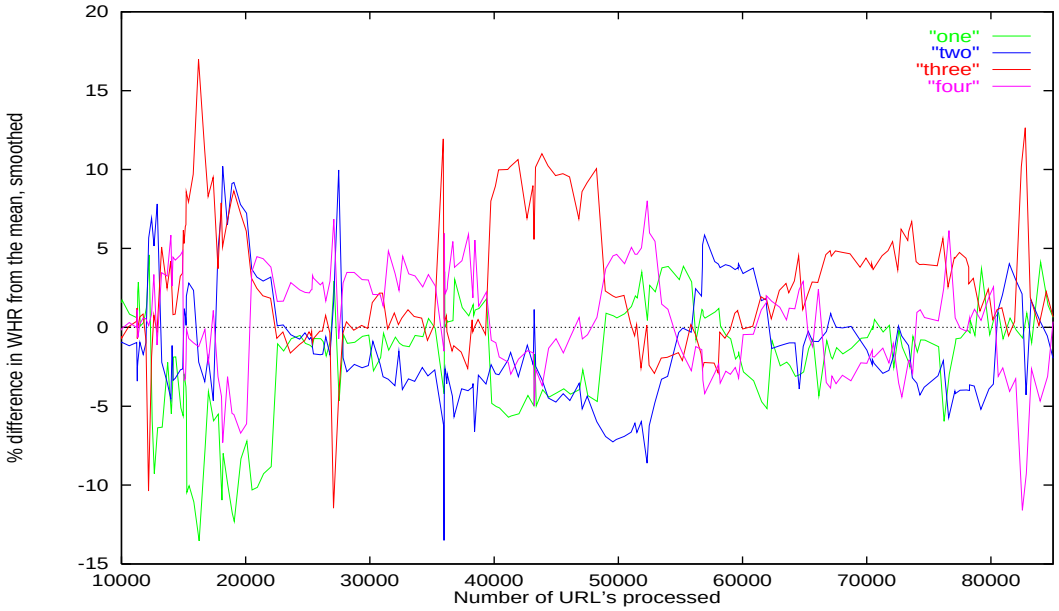

Figure 9.22: Experiment three, VTCS2, Hybrid Algorithms, Normalized, WHR.

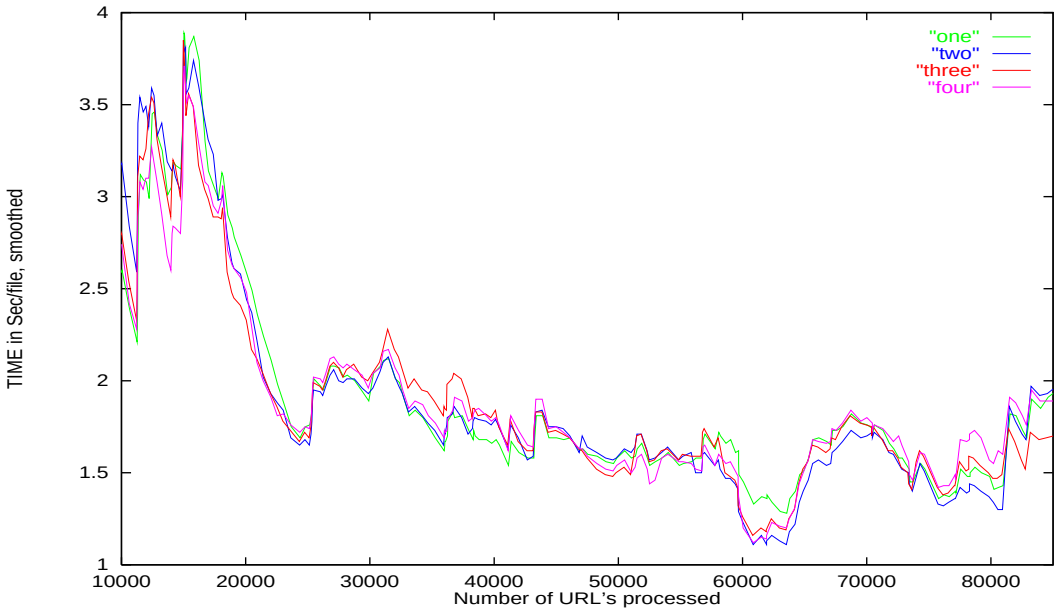

Figure 9.23: Experiment three, VTCS2, Hybrid Algorithms, TIME.

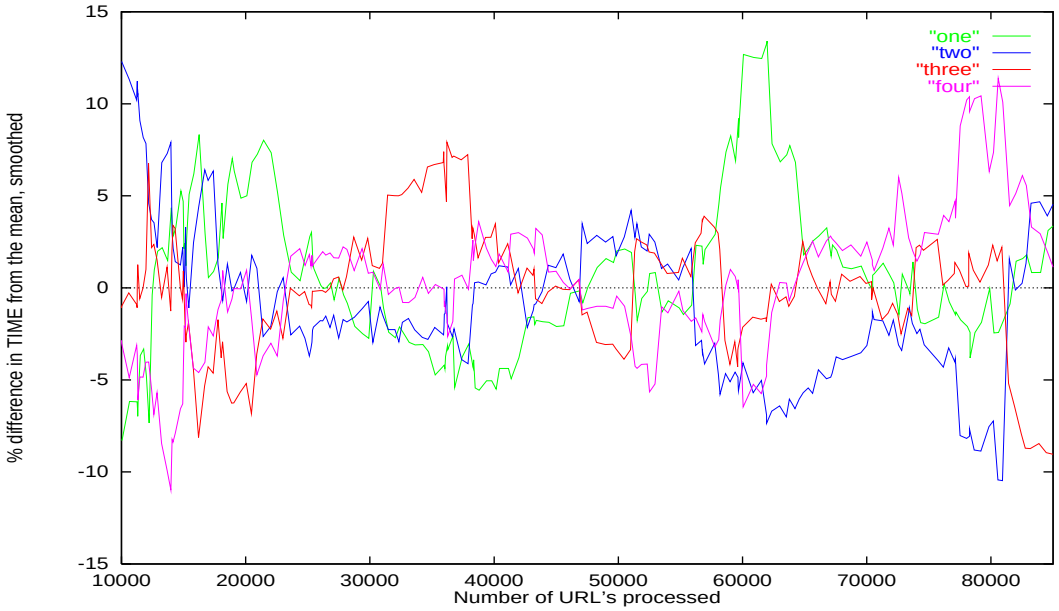

Figure 9.24: Experiment three, VTCS2, Hybrid Algorithms, Normalized, TIME.

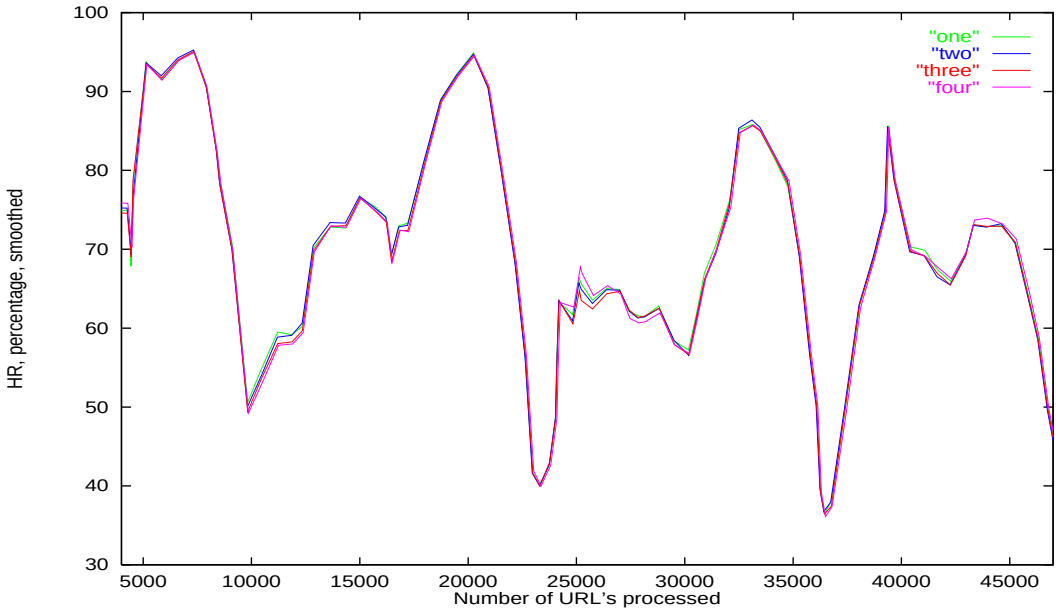

Figure 9.25: Experiment three, BULOG, Hybrid Algorithms, HR.

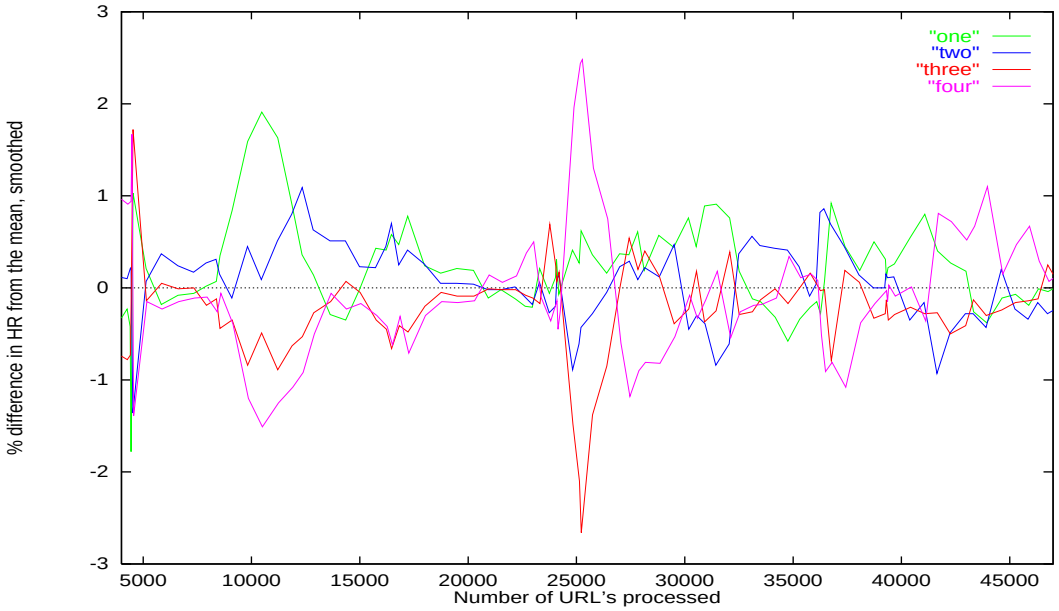

Figure 9.26: Experiment three, BULOG, Hybrid Algorithms, Normalized, HR.

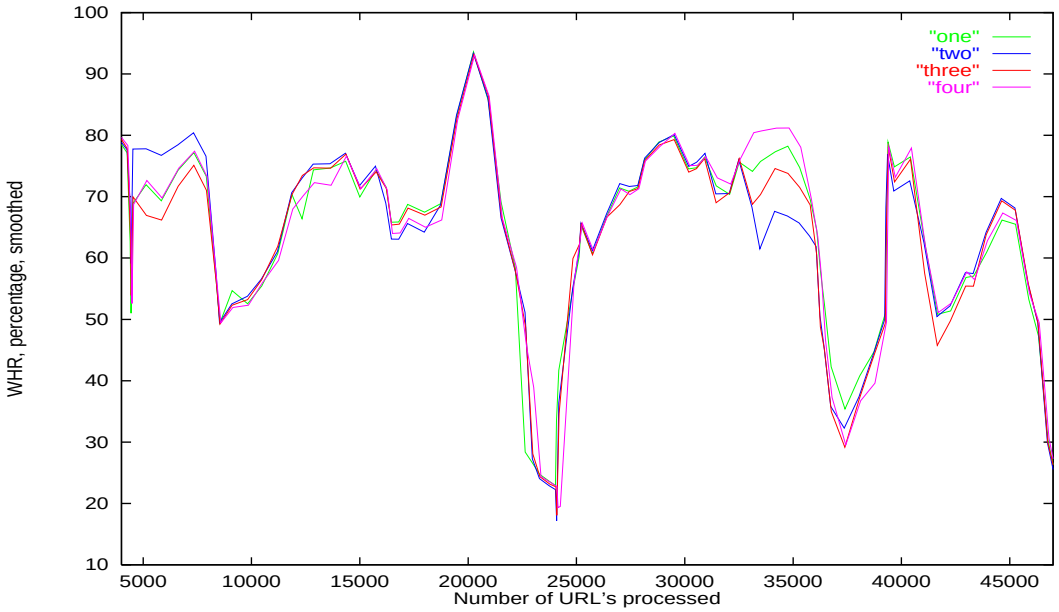

Figure 9.27: Experiment three, BULOG, Hybrid Algorithms, WHR.

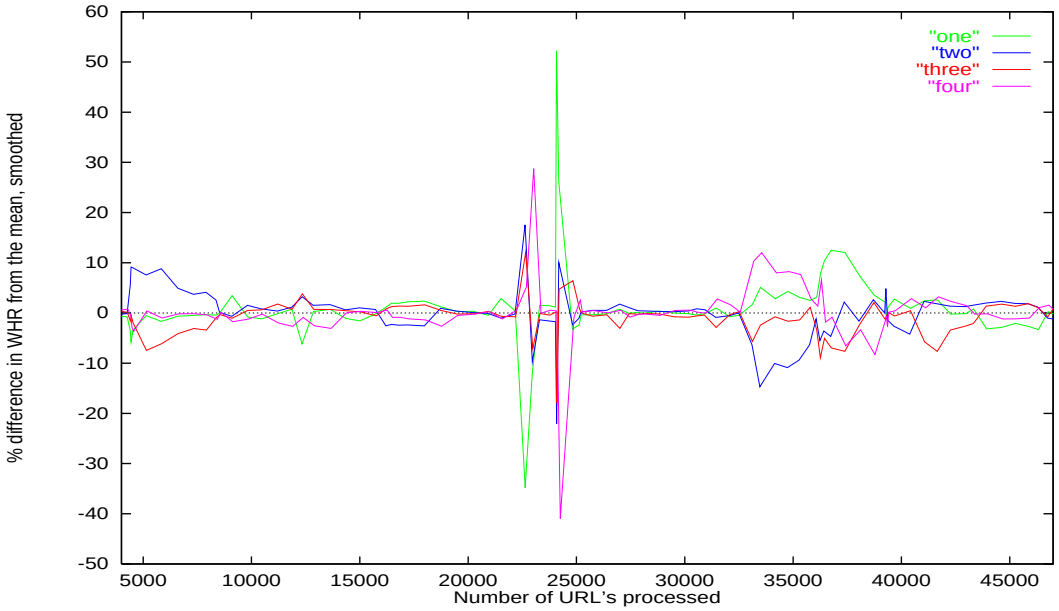

Figure 9.28: Experiment three, BULOG, Hybrid Algorithms, Normalized, WHR.

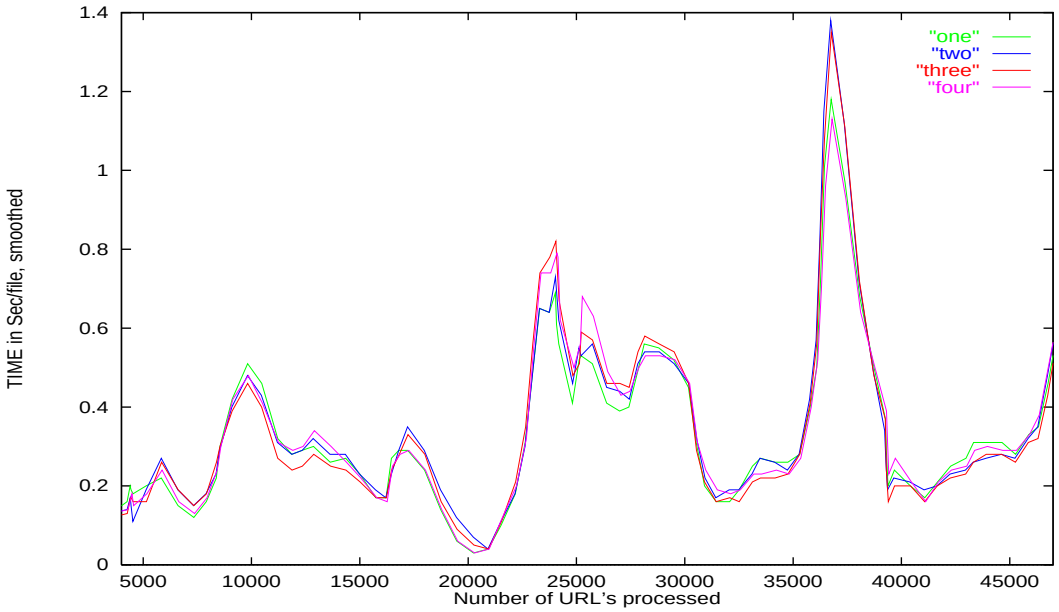

Figure 9.29: Experiment three, BULOG, Hybrid Algorithms, TIME.

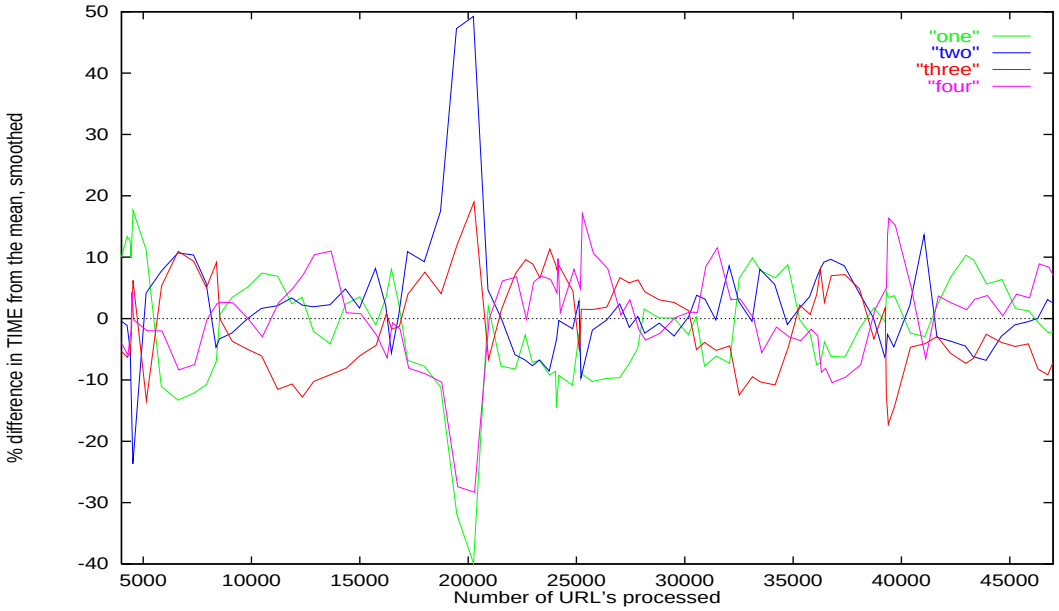

Figure 9.30: Experiment three, BULOG, Hybrid Algorithms, Normalized, TIME.

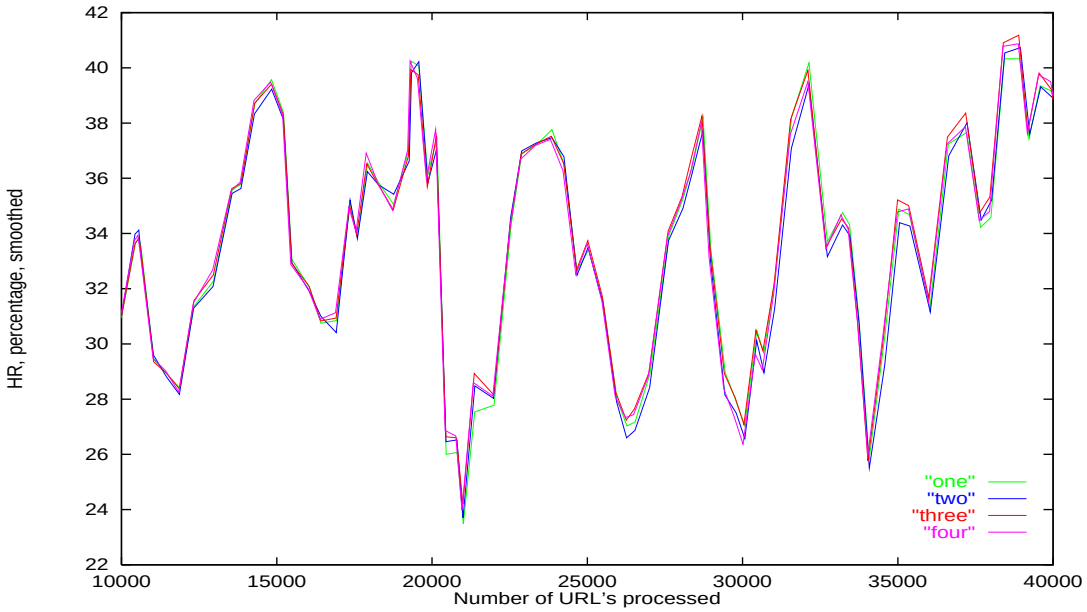

Figure 9.31: Experiment three, VTLIB, Hybrid Algorithms, HR.

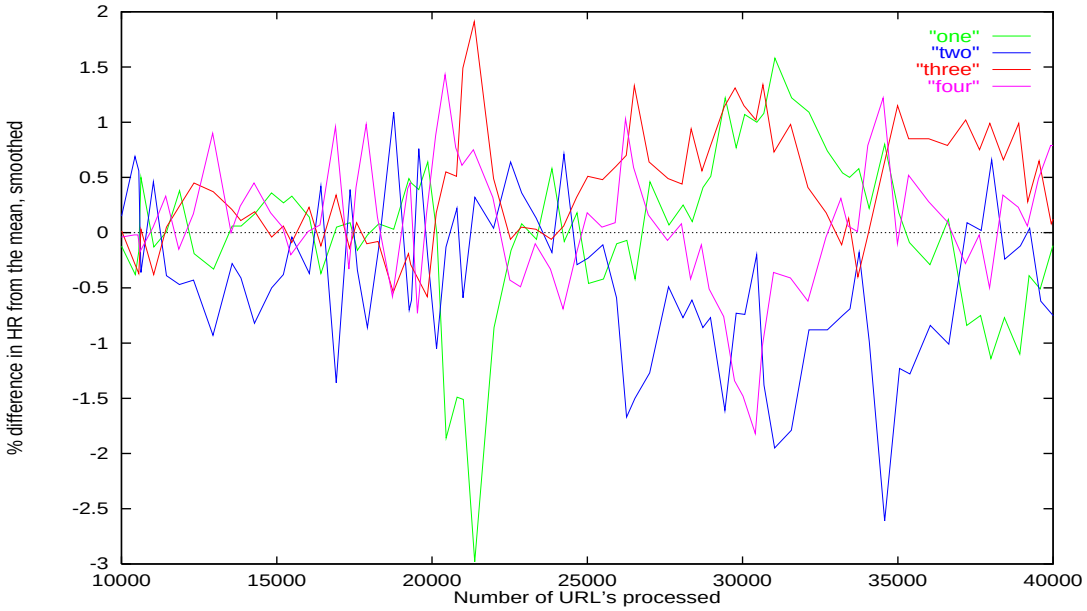

Figure 9.32: Experiment three, VTLIB, Hybrid Algorithms, Normalized, HR.

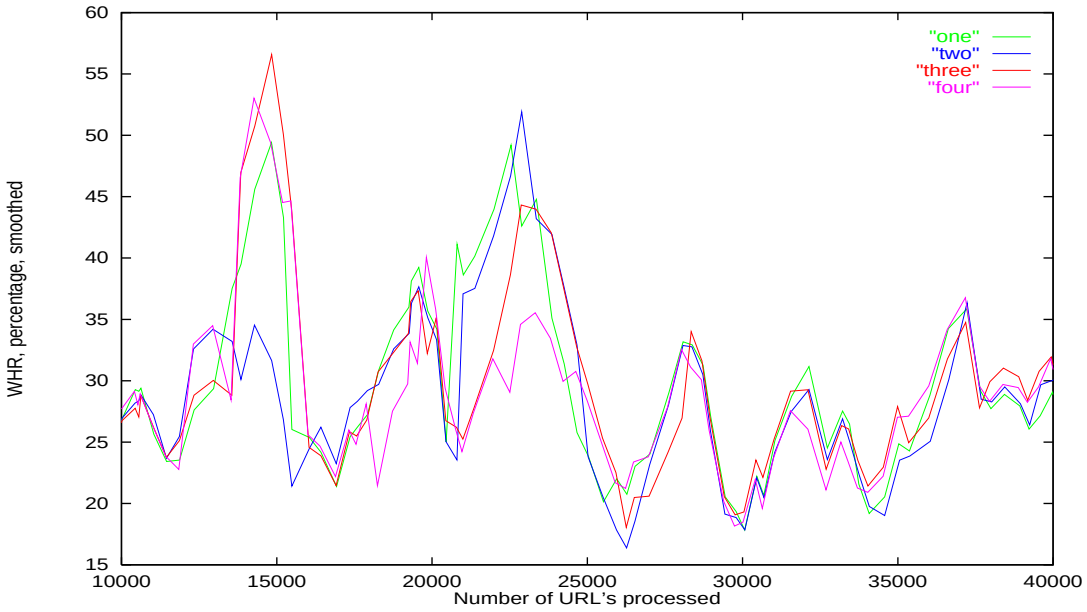

Figure 9.33: Experiment three, VTLIB, Hybrid Algorithms, WHR.

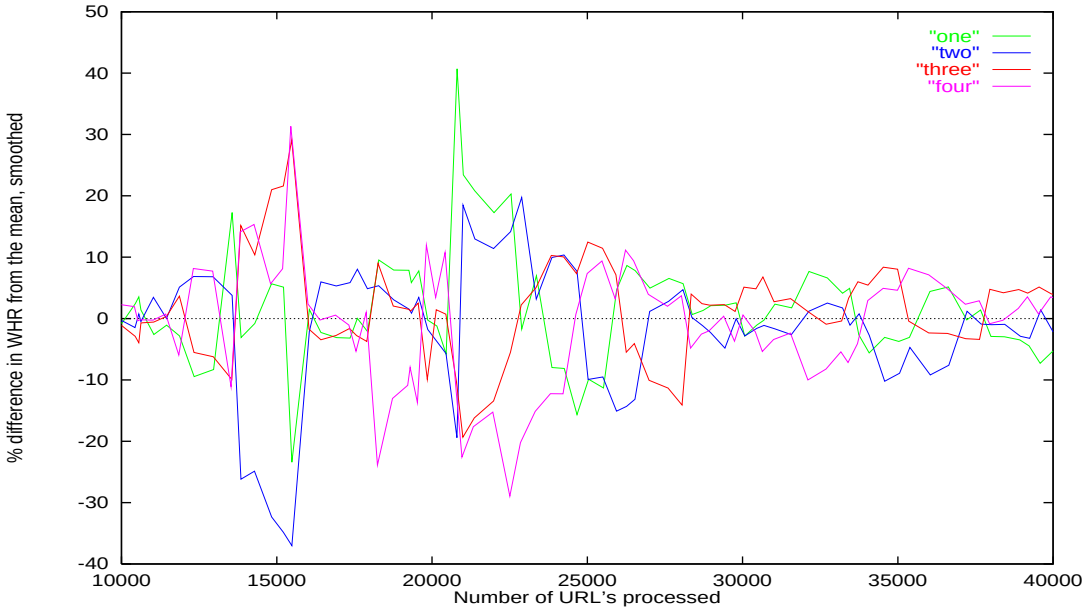

Figure 9.34: Experiment three, VTLIB, Hybrid Algorithms, Normalized, WHR.

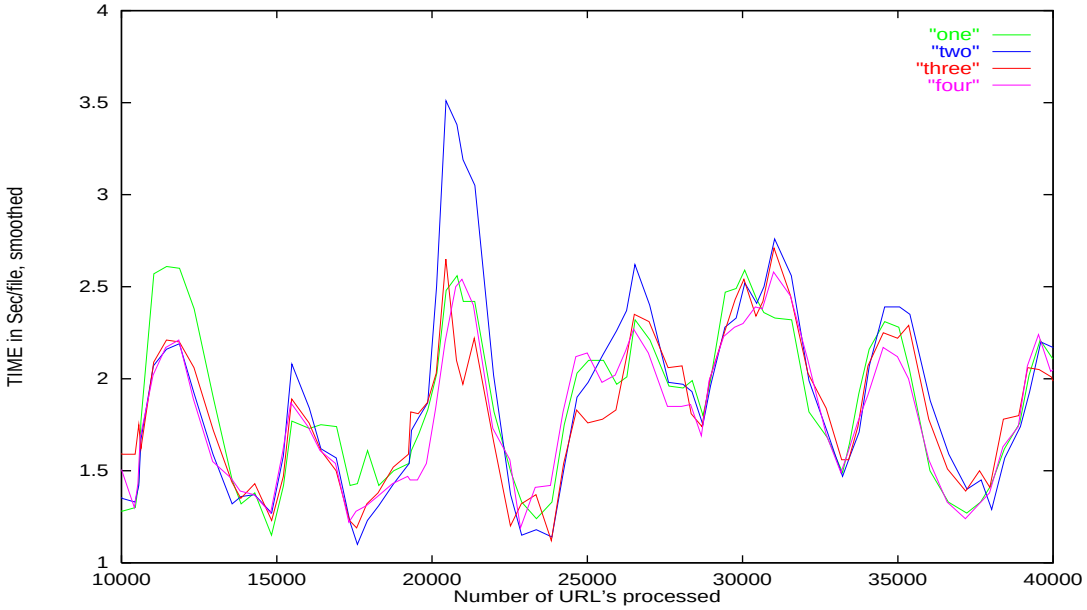

Figure 9.35: Experiment three, VTLIB, Hybrid Algorithms, TIME.

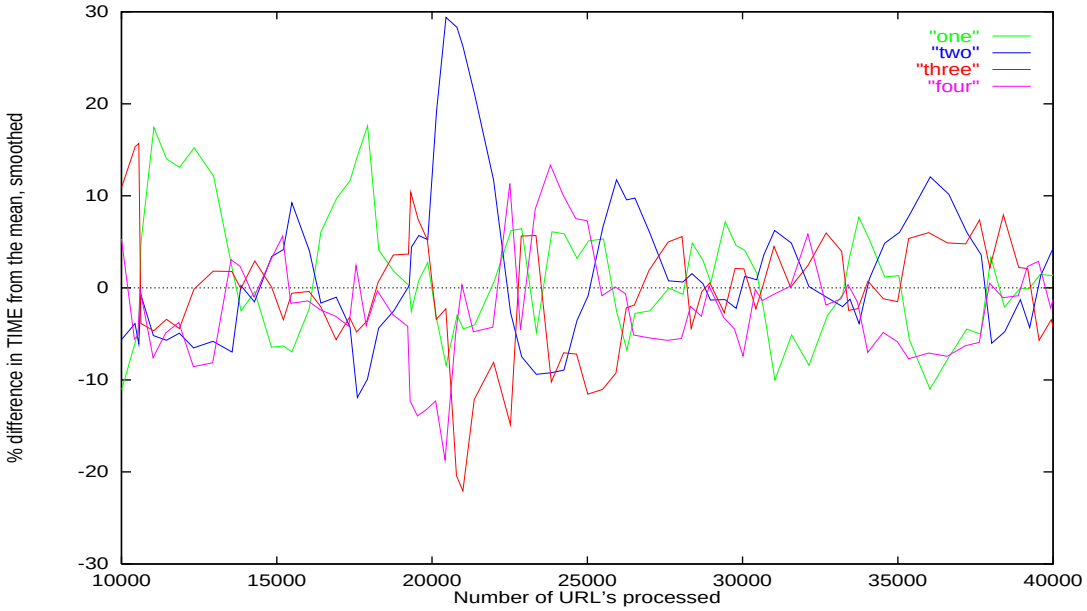

Figure 9.36: Experiment three, VTLIB, Hybrid Algorithms, Normalized, TIME.

Table 9.6: Confidence interval ranges at the 90% level for removal algorithms, accelerated VTCS2.

	HR % difference from the mean. Higher is better	Mean	WHR % difference from the mean. Higher is better	Mean	TIME % difference from the mean. Lower is better	Mean
LRU	(-2.66, -0.91)	-1.78	(6.86, 12.25)	9.55	(-0.42, 3.84)	1.71
LFU	(-5.35, -3.60)	-4.48	(3.22, 8.60)	5.91	(0.32, 4.57)	2.44
SIZE	(2.65, 4.40)	3.53	(-14.32, -8.93)	-11.62	(-4.10, 0.15)	-1.98
HYB	(1.86, 3.61)	2.73	(-6.53, -1.15)	-3.84	(-4.31, -0.05)	-2.18

Table 9.7: Confidence interval ranges at the 90% level for removal algorithms, accelerated BULOG.

	HR % difference from the mean. Higher is better	Mean	WHR % difference from the mean. Higher is better	Mean	TIME % difference from the mean. Lower is better	Mean
LRU	(-0.87, 0.03)	-0.42	(-0.24, 2.68)	1.22	(0.96, 9.42)	5.19
LFU	(-0.93, -0.04)	-0.49	(-1.95, 0.96)	-0.49	(-0.43, 8.04)	3.81
SIZE	(0.07, 0.97)	0.52	(-3.02, -0.11)	-1.56	(-10.31, -1.84)	-6.0
HYB	(-0.07, 0.83)	0.38	(-0.62, 2.29)	0.84	(-7.16, 1.31)	-2.93

9.4 Significance of the Parallel Log Tests

The data from the graphs of sections 9.2 and 9.3, although not smoothed, are used to determine which algorithms are significantly superior. Tables 9.6, 9.7 and 9.8 show the confidence interval ranges, of the removal algorithms, used to create Table 9.9. Table 9.9 presents the ordering of the significantly better removal algorithms.

Tables 9.10, 9.11 and 9.12 show the confidence interval ranges of the HYB constants, used to create Table 9.13. Table 9.13 presents the ordering of the significantly better constants in the HYB algorithm. The method used to construct these tables is discussed in section 8.4. Both of the previous two sections' results for LRU, LFU, SIZE, and HYB removal algorithm test, and the test using different constants with four proxies using the HYB algorithm are recorded in this table. The actual raw data is recorded in Appendices C.3 to C.8.

Table 9.8: Confidence interval ranges at the 90% level for removal algorithms, VTLIB.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
LRU	(-2.91, 0.32)	-1.29	(-1.14, 7.78)	3.32	(-1.85, 2.79)	0.47
LFU	(-4.17, -0.93)	-2.55	(-4.20, 4.72)	0.26	(-1.65, 2.98)	0.67
SIZE	(0.49, 3.72)	2.10	(-9.94, -1.01)	-5.47	(-2.59, 2.05)	-0.27
HYB	(0.13, 3.36)	1.74	(-2.57, 6.36)	1.89	(-3.18, 1.45)	-0.86

Table 9.9: Illustration of the significantly better removal algorithms in experiment three, using 90% confidence intervals.

Workload	Measure	Best	Medium	Worst
VTCS2	HR	SIZE/HYB	LRU	LFU
VTCS2	WHR	LRU/LFU	HYB	SIZE
VTCS2	TIME	HYB/SIZE/LRU		LRU/LFU
BULOG	HR	SIZE/HYB		HYB/LRU/LFU
BULOG	WHR	LRU/HYB/LFU/SIZE		
BULOG	TIME	SIZE/HYB		HYB/LFU/LRU
VTLIB	HR	SIZE/HYB	HYB/LRU	LRU/LFU
VTLIB	WHR	LRU/HYB/LFU/SIZE		
VTLIB	TIME	HYB/SIZE/LRU/LFU		

Table 9.10: Confidence interval ranges at the 90% level for HYB constants, accelerated VTCS2.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
one	(-0.77, -0.15)	-0.46	(-3.11, -0.05)	-1.58	(-1.03, 2.93)	0.95
two	(-0.39, 0.22)	-0.09	(-2.92, 0.13)	-1.39	(-4.06, -0.10)	-2.08
three	(0.27, 0.89)	0.58	(0.71, 3.76)	2.23	(-1.63, 2.33)	0.35
four	(-0.34, 0.28)	-0.03	(-0.78, 2.27)	0.74	(-1.20, 2.76)	0.78

Table 9.11: Confidence interval ranges at the 90% level for HYB constants, BULOG.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
one	(0.04, 0.38)	0.21	(-0.35, 2.55)	1.10	(-4.38, 0.89)	-1.74
two	(-0.06, 0.28)	0.11	(-1.91, 0.98)	-0.46	(-0.83, 4.44)	1.80
three	(-0.40, -0.06)	-0.23	(-2.36, 0.53)	-0.91	(-3.29, 1.99)	-0.65
four	(-0.26, 0.08)	-0.09	(-1.17, 1.73)	0.28	(-2.05, 3.23)	0.59

Table 9.12: Confidence interval ranges at the 90% level for HYB constants, VTLIB.

	HR		WHR		TIME	
	% difference from the mean. Higher is better	Mean	% difference from the mean. Higher is better	Mean	% difference from the mean. Lower is better	Mean
one	(-0.17, 0.48)	0.16	(-2.94, 6.92)	1.99	(-2.44, 5.01)	1.28
two	(-0.83, -0.18)	-0.50	(-7.71, 2.15)	-2.78	(-0.98, 6.47)	2.75
three	(0.02, 0.67)	0.34	(-2.79, 7.06)	2.14	(-5.37, 2.09)	-1.64
four	(-0.32, 0.33)	0.00	(-6.28, 3.58)	-1.35	(-6.12, 1.34)	-2.39

Table 9.13: Illustration of the significantly better constants in the HYB algorithms in experiment three, using 90% confidence intervals.

Workload	Measure	Best	Worst
VTCS2	HR	three/four	four/two/one
VTCS2	WHR	three/four	four/two/one
VTCS2	TIME	two/three/four/one	
BULOG	HR	one/two/four	four/three
BULOG	WHR	one/four/two/three	
BULOG	TIME	one/three/four/two	
VTLIB	HR	three/one/for	four/two
VTLIB	WHR	three/one/four/two	
VTLIB	TIME	four/three/one/two	

9.5 Conclusions of Experiment Three

Table 9.14 presents the ICS, cache size used, fraction of ICS used, IHR, IWHR, HR, and WHR for the three workloads. Once again the HR and WHR recorded are the best values from all the removal algorithms. The actual HR, WHR, and time per file is shown in Table 9.3. It should be noted that in all cases the proxy yields lower HR and WHR than the IHR and IWHR; the IHR and IWHR are the maximum achievable HR and WHR. However, the proxy performs comparatively well against the IHR and IWHR considering the small fraction of the ICS that is used for the cache size.

Table 9.14: Comparison of the IHR and IWHR with the best HR and WHR.

Work-load	ICS (MB)	Cache size (MB)	Cache Size as a % of ICS	% IHR	% IWHR	% HR	% WHR
VTCS2	451	50	11	46	34	36	24
BULOG	38	10	26	82	57	67	53
VTLIB	138	50	36	38	32	32	28

With respect to reduction of average latency the results of the removal algorithm test, Table 9.9 illustrates that the HYB algorithm is among the best performing algorithms in eight of nine cases. With respect to both TIME and HR, the HYB and SIZE algorithms are not significantly different in all six cases. With respect to WHR, LRU performs the best in every test, although in two of the tests it is not significantly better than the HYB algorithm.

From Table 9.13 the relative performance of the different constants for the HYB algorithm can be found. However, there are no algorithms that perform significantly better than any others. Algorithm three is generally slightly better, but not significantly better, for the VTCS2 and VTLIB workloads. Algorithm one is consistently slightly better, but again not significantly better, for the BULOG workload. Tables 9.15, 9.16 and 9.17 present the fraction of variation due to each of the three constants (W_B, W_N, and CONN) and the errors. The fraction of variation due to errors is larger than for any of the factors in all the experiments, leading to the conclusion that the factors are not very important. However, of

Table 9.15: Fraction of Variance Explained by Factors in VTCS2

Measure	W_B + W_N· CONN	W_N W_B· CONN	Connect pkt size (CONN) + $W_B \cdot W_N$	Error
HR	0.01	0.11	0.09	0.79
WHR	0.01	0.14	0.01	0.84
TIME	0.02	0.01	0.03	0.94

Table 9.16: Fraction of Variance Explained by Factors in BULOG

Measure	W_B + W_N· CONN	W_N W_B· CONN	Connect pkt size (CONN) + $W_B \cdot W_N$	Error
HR	0.00	0.25	0.03	0.72
WHR	0.00	0.02	0.08	0.90
TIME	0.07	0.00	0.02	0.91

Table 9.17: Fraction of Variance Explained by Factors in VTLIB

Measure	W_B + W_N· CONN	W_N W_B· CONN	Connect pkt size (CONN) + $W_B \cdot W_N$	Error
HR	0.28	0.13	0.03	0.56
WHR	0.13	0.00	0.00	0.86
TIME	0.00	0.20	0.01	0.79

the factors it would appear that, if any, the emphasis factor (W_N) is most important. The results indicate that the emphasis factor should be set to 1.1, emphasizing NREF rather than SIZE. The choice of values for the estimate of average file size (W_B), and the size considered a "connection" packet (CONN) are of relatively little importance, as the errors explain a much larger percentage of the variation.

Chapter 10

Conclusions

This chapter discusses the merits of the HYB algorithm and what can be done to further improve proxy caching for the World Wide Web.

10.1 The HYB Algorithm

The results of Chapter 7 clearly indicate that a removal algorithm using *only* latency as a factor is a very poor performer. The results of Chapters 8 and 9 indicate that the HYB algorithm, which uses a combination of document size, number of past references to the document, estimated connection bandwidth, and estimated connection time, is a very robust removal algorithm. The HYB algorithm demonstrates superior performance over the standard three removal algorithms (LRU, LFU, and SIZE) in most experiments for TIME and HR.

Of the three constants used in the HYB algorithm only factor W_N, the NREF/SIZE emphasis factor, seemed to show a small effect. The results generally conclude that setting W_N to a value of 1.1 is preferable. By setting the value above 1.0 this increases the emphasis of NREF over SIZE. Suggestions for improving the performance of the proxy cache even further are given in the following section.

10.2 Further Studies

This section discusses areas of the research that could be continued further. Possible suggestions to further improve the performance of the proxy are given.

10.2.1 Proxy Factors

The proxy, as discussed in section 4.3, removes the least valued eight documents out of a hashed group of 256. Although reducing the demand on the CPU, this does not lead to a set of truly optimal documents to remove. This value could be changed, to perhaps quadruple the CPU demand, by choosing four from 512 documents.

10.2.2 Psychological Effects of Delay Variance

The aim of the improvements were to improve the performance of the system as much as possible. There are, however, psychological effects of delay variance that might be interesting to study. A user that has an average access time of 3 seconds per file is more likely to be happy than a user with an average access time of say 2.9 seconds but suffers a huge variance in download times. Currently the removal algorithm smoothes the current estimated connection time and the CBW rate. This does not take into account the variance in delay that may be experienced from an origin server. Although this may result in improved average performance the variance may be greater than an algorithm that used this variance as a factor. It is, however, doubtful that this is a trivial problem with a trivial solution; every user will react differently and may not in fact be bothered by large variances. Certain experienced Web users may in fact benefit from reduced average response times with larger variance, because they will simply not bother with the documents that take too long. It should be noted that the experimental results are slightly biased by the fact that users do not wait for excessively long downloads which are often aborted. A proxy that takes latency into account will help these users the most, as fewer requests will be aborted.

10.2.3 Adaptive Algorithms

As discussed previously in sections 3.6 and 3.7 there may be transient traits within the data that might be worth exploiting. If a machine has a fixed limit on its disk space it might be worth using a greater proportion of the spare space in the daytime for caching

WWW traffic. At night there is less use of the WWW, but a greater need for storage of new mail messages that might arrive at night and not be read for hours; at this time the extra disk space might be required by the mail daemon. There are many aspects that adaptive algorithms could explore. In addition to the changing use of disk space there might also be periodic traits to the caching of different types of documents, such as a greater use of *.ps* files in the day versus a greater use of *.mpeg* at night. This would be because generally only businesses and universities can afford PostScript printers; many scientific papers are written and posted on the Web in PostScript. Consequentially during "office" hours many PostScript documents will be requested on the WWW. At night, when people are generally at home, a greater number of video files may be loaded. The reason for a greater usage of video, and possibly audio, at night is because generally the network is less loaded (lower latency to load documents) and home machines are more likely to contain sound cards than business and university machines.

REFERENCES

[1] Ghaleb Abdulla, Marc Abrams, and Edward A. Fox. Scaling the World Wide Web. <URL: http://ei.cs.vt.edu/~succeed/96ieeeAAF/>, April 1996. Unpublished paper (5 Dec. 1996).

[2] Marc Abrams, Charles Standridge, Ghaleb Abdulla, Stephen Williams, and Edward Fox. Caching Proxies: Limitations and Potentials. *4th Inter. World-Wide Web Conference*, pages 119–133, December 1995. <URL: http://www.w3.org/pub/WWW/-Journal/1/abrams.155/paper/155.html> Also in WWW Journal Vol. 1 (5 Dec. 1996).

[3] Marc Abrams, Charles Standridge, Stephen Williams, Ghaleb Abdulla, Edward Fox, Shashin Patel, and Randy Ribler. Multimedia Traffic Analysis Using Chitra95. *ACM Multimedia '95, San Francisco CA*, pages 267–276, November 1995.

[4] Martin F. Arlitt and Carey L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proc. SIGMETRICS*, Philadelphia, PA, April 1996. ACM. University of Saskatchewan.

[5] Jean-Chrysostome Bolot and Philipp Hoschka. Performance Engineering of the World Wide Web. *The World Wide Web Journal*, 1(3):185–195, Summer 1996. <URL: http://www.w3.org/pub/WWW/Journal/3/s3.bolot.html> (14 Dec. 1996).

[6] Anawat Chankhuthod, Peter Danzig, Chuck Neerdaels, Michael Schwartz, and Kurt Worrel. A Hierarchical Internet Object Cache. <URL: ftp://ftp.cs.colorado.edu-/pub/cs/techreports/schwartz/HarvestCache.ps.Z>. University of Southern California, University of Colorado, Boulder (5 Dec. 1996).

[7] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of WWW Client-based Traces. Technical Report BU-CS-95-010, Computer Science Department, Boston University, Computer Science Department, Boston University, 111 Cummington St, Boston, MA 02215, July 1995.

[8] Peter Danzig, Richard Hall, and Michael Schwartz. A Case for Caching File Objects Inside Internetworks. Technical Report CU-CS-642-93, U. of Southern California, U. of Colorado at Boulder, March 1993.

[9] Jennie File. NCSA Web Server Activity: Total Connections. <URL: http://www.ncsa.-uiuc.edu/SDG/Presentations/Stats/WebServer.html>, November 1995. Unpublished slides (5 Dec. 1996).

[10] Jennie File. NSFnet Backbone Byte Traffic. <URL: http://www.ncsa.uiuc.edu/SDG/-Presentations/Stats/NSFnetStats.html>, April 1995. Unpublished slides (5 Dec. 1996).

[11] HTTP Working Group. Hypertext Transfer Protocol — HTTP/1.1. <URL: http://-www.w3.org/pub/WWW/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-07.txt>, August 1996. (18 Dec. 1996).

[12] James Gwertzman and Margo Seltzer. World Wide Web Cache Consistency. *1996 Proceedings of the Usenix Technical Conference*, 1996. Microsoft Corporation, Harvard University.

[13] Darren Hardy, Michael Schwartz, and Duane Wessels. Harvest User's Manual. Technical Report CU-CS-743-94, University of Colorado at Boulder, Department of Computer Science, University of Colorado, Boulder, Colorado 80309-0430, September 1995. Version 1.3.

[14] Van Jacobson. Congestion Avoidance and Control. In *Proc. SIGCOMM*, pages 314–329, Stanford, CA, August 1988. ACM.

[15] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, 1991.

[16] Eric Dean Katz, Michell Butler, and Robert McGrath. A stable HTTP Server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(0169–7552(94)00068-9):155–164, 1994.

[17] Thomas Kwan, Robert McGrath, and Daniel Reed. NCSA's World Wide Web Server: Design and Performance. *IEEE Computer*, 1995(0018–9162):68–74, 1995.

[18] Thomas Kwan, Robert McGrath, and Daniel Reed. User Access Patterns to NCSA's World Wide Web Server. Technical Report UIUCDCS-R-95-1934, Dept. of Comp. Sci., Univ. of Illinois, February 1995.

[19] Ari Luotonen and Kevin Altis. World-Wide Web Proxies. *Computer Networks and ISDN Systems*, 27(2), 1994. <URL: http://www1.cern.ch/PapersWWW94/-luotonen.ps> (5 Dec. 1996).

[20] Radhika Malpani, Jacob Lorch, and David Berger. Making World Wide Web Caching Servers Cooperate. In *4th Inter. World-Wide Web Conference*, pages 107–117, Boston, MA, December 1995. <URL: http://www.w3.org/pub/WWW/Journal/1/lorch.059/-paper/059.html> (18 Dec. 1996).

[21] Jeffrey Mogul. The Case for Persistent Connection HTTP. *ACM*, SIGCOMM 95(0–89791–711–1/95/0008):299–313, 1995.

[22] Danial O'Callaghan. A Central Caching proxy Server for the WWW. *Asia Pacific WWW Conference 1995, Sydney Australia*, August 1995. University of Melbourne, Australia.

[23] James E. Pitkow and Margaret M. Recker. A Simple yet Robust Caching Algorithm Based on Dynamic Access Patterns. *The Second International WWW Conference '94*, 2:1039–46, October 1994.

[24] Neil Smith. What Can Archives Offer the World-wide Web, May 1994. <URL: http://www.hensa.ac.uk/www94/wwwcache.ps.Z> (5 Dec. 1996).

[25] Simon Spero. Next Generation Hypertext Transport Protocol. <URL: http://sunsite.unc.edu/ses/ng-notes.txt>, March 1995. Unpublished Paper (5 Dec. 1996).

[26] W. Richard Stevens. *TCP/IP Illustrated*, volume 3. Addison-Wesley, Reading, Massachusetts, 1996. Section 10.5, Retransmission Timeout Calculations.

[27] W3C. Propagation, Replication and Caching. <URL: http://www.w3.org/hypertext/WWW/Propagation/Activity.html>. Unpublished paper (5 Dec. 1996).

[28] W3C. Internet Survey Methodology and Web Demographics. Boston, MA, Workshop, January 1996.

[29] Duane Wessel. Intelligent caching for World Wide Web Objects. *INET 95 Proceedings*, 1995. University of Colorado, Boulder.

[30] Stephen Williams, February 1996. Personal Communications.

[31] Stephen Williams, Marc Abrams, Charles Standridge, Ghaleb Abdulla, and Edward A Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proc. SIGCOMM*, pages 293–305, August 1996. <URL: http://ei.cs.vt.edu/~succeed/96sigcomm/> (5 Dec. 1996).

Appendix A

Glossary

CBW: Connection Bandwidth, see section 5.3. Its variable *cbw* is used in equation (4.3).

CGI: Common Gateway Interface, see section 3.5.

CID: Child Id, see section 9.1. This is the identification number of the child process in WebJamma, its default range is $0 \ldots 39$.

CLAT: Connection Latency, see section 5.3. Its variable *clat* is used in equation (4.2).

CONN: Constant used in the HYB algorithm, see equation (4.2, and 4.3). It is used to define the threshold size used to determine whether the document will be used for either connection time (CLAT) calculations or connection bandwidth (CBW) calculations. An example size is 2KB: where documents smaller than 2KB are used only for connection time calculations, and documents larger than 2KB are used for bandwidth calculations.

DNS: Domain Name Service.

FTP: File Transfer Protocol.

HR: Hit Rate, see equation (1.1), and section 5.3.

HTTP: Hyper Text Transfer Protocol.

HYB: Hybrid removal algorithm, see section 4.3.5.

ICS: Infinite Cache Size, see section 7.3.

IHR: Infinite Hit Rate, see section 7.3.

IFH: Ignore First Hit, see section 4.2.

IP: Internet Protocol.

IWHR: Infinite Weighted Hit Rate, see section 7.3.

LFU: Least Frequently Used, see section 4.1.1.

LAT: Latency removal algorithm, see section 4.3.4.

LRU: Least Recently Used removal algorithm, see section 4.1.1.

NCSA: National Center for Supercomputing Applications

NREF: Number of REFerences, i.e., number of hits.

NSF: National Science Foundation.

PPV: Pay Per View, similar to the video on demand in hotels.

PrN: Proxy Number, see section 9.1. This is the identification number of the proxy, from 1 . . . 4. This is used when using WebJamma.

RTT: TCP's Round Trip Time smoothing estimation, see [26].

SIZE: Size removal algorithm, see section 4.1.2.

TCP: Transport Control Protocol.

TIME: Time to download, see section 5.3.

TTL: Time To Live - time before a file "expires", see section 3.3.

URL: Uniform Resource Locator.

W_B: Constant used in the HYB algorithm, see equation (4.6). It is used to set the emphasis of CLAT versus CBW.

WHR: Weighted Hit Rate, see equation (1.2), and section 5.3.

W_N: Constant used in the HYB algorithm, see equation (4.6). Increasing the value of the constant above one emphasizes NREF over SIZE.

WWW: World Wide Web.

Appendix B
Removal Algorithm Verification

This appendix reports on a simple test of the removal algorithms to verify that they remove the correct documents. This appendix discusses the results of three small tests using the four different removal algorithms. The test files are extremely short[1] so as to allow verification, by hand, that the removal algorithms work as required. In the following sections the defaults and software setup is described. Then a "by hand" run through of the test files is given. The first test verifies both the LRU and LFU algorithms, the second test verifies the SIZE algorithm, and finally the third verifies the LAT algorithm.

B.1 Software and Environment Settings

The cache size was set to 1MB, small enough to calculate by hand which documents would be removed by each removal algorithm. The upper and lower thresholds for the cache were set to 90% and 75% respectively. Thus once the size of the cache exceeds 900KB the cache will remove documents until the cache size is below 750KB. For example: a cache size of 800KB would not activate the removal algorithm.

The environment was such that the cache only operated on the documents in this verification process; no other users were accessing the cache. The cache was set to cache local documents; this made it much quicker to test the removal algorithms. A selection of documents was created; each file consisted of only the character "a" duplicated thousands of times. The naming schema for the test documents was simple, [a-e][1,2,3,5]00.html, for example the file "b200.html" was a file consisting solely of the character "a" duplicated

[1] Only about 20 documents were used in the trace; this allows verification by hand.

200,000 times. The number indicates the size of the file in KB. The log files have been reduced (a simple UNIX "cut -c85-"[2] command has been used) for ease of readability, only the "TCP_DONE" and "TCP_HIT" lines have been included, and only the characters after the 85th are shown.

The format of the log file is as follows. "TCP_MISS" is written to the file whenever a request is made; these have been removed as they are duplicated when the request is satisfied. The request can be satisfied in one of two ways. The first is by a "TCP_HIT" meaning that all four algorithms were currently caching the file. The second is with a "TCP_DONE" this means that either none or some of the removal algorithms were currently caching the file. The set of symbols {R,F,S,L} contained within the first square bracket indicate which removal algorithms were caching the file; they stand for LRU, LFU, SIZE, and LAT, respectively. The second value in the square brackets is the time in milliseconds that the download took. The third value is the size of the file in bytes.

B.2 LRU and LFU Verification

The trace file shown in Figure B.1 illustrates the trace used to verify the LRU and LFU removal algorithms. Table B.1 shows step by step how the documents are removed in the LRU removal algorithms. Table B.2 shows how the LFU removal algorithm operates.

B.3 SIZE Verification

The trace file shown in Figure B.2 illustrates the trace file used to verify the SIZE removal algorithm. Table B.3 shows step by step how the documents are removed.

[2]This command strips off the first 84 characters from each line in the file.

```
a200.html" TCP_DONE [    ] [1496] 200176
b200.html" TCP_DONE [    ] [1141] 200176
a200.html" TCP_HIT [RFSL] 200176
b200.html" TCP_HIT [RFSL] 200176
c200.html" TCP_DONE [    ] [1355] 200176
d200.html" TCP_DONE [    ] [1349] 200176
e200.html" TCP_DONE [    ] [1416] 200176
a200.html" TCP_DONE [ F  ] [1371] 200176
c200.html" TCP_DONE [R SL] [1318] 200176
d500.html" TCP_DONE [    ] [8110] 500176
d100.html" TCP_DONE [    ] [607] 100176
c200.html" TCP_DONE [R SL] [1412] 200176
a200.html" TCP_DONE [ F  ] [1341] 200176
d300.html" TCP_DONE [    ] [3579] 300176
c200.html" TCP_DONE [RFS ] [1303] 200176
a200.html" TCP_DONE [RFS ] [1452] 200176
d500.html" TCP_DONE [    ] [7606] 500176
a200.html" TCP_DONE [RFS ] [1404] 200176
c200.html" TCP_DONE [    ] [1516] 200176
```

Figure B.1: Trace file 1. Demonstrating LRU and LFU removal algorithms.

Table B.1: Step by step illustration of how the LRU removal algorithm removes documents.

File & Size	LRU hit?	Cache size	Cache size w/new file	Remove documents?	Documents cached & removal order
a200		0	200		a200
b200		200	400		a200, b200
a200	Y	400	400		a200, b200
b200	Y	400	400		a200, b200
c200		400	600		a200, b200, c200
d200		600	800		a200, b200, c200, d200
e200		800	1000	a200, b200	c200, d200, e200
a200		600	800		c200, d200, e200, a200
c200	Y	800	800		d200, e200, a200, c200
d500		800	1300	d200, e200, a200	c200, d500
d100		700	800		c200, d500, d100
c200	Y	800	1000	c200, d500	d100, c200
a200		300	500		d100, c200, a200
d300		500	800		d100, c200, a200, d300
c200	Y	800	800		d100, a200, d300, c200
a200	Y	800	800		d100, d300, c200, a200
d500		800	1300	d100, d300, c200	a200, d500
a200	Y	700	700		d500, a200
c200		700	900		d500, a200, c200

Table B.2: Step by step illustration of how the LFU removal algorithm removes documents.

File & Size	LFU hit?	Cache size	Cache size w/new file	NREF each file; blank - not cached a2	b2	c2	d1	d2	d3	d5	e2
a200		0	200	1							
b200		200	400	1	1						
a200	Y	400	400	2	1						
b200	Y	400	400	2	2						
c200		400	600	2	2	1					
d200		600	800	2	2	1		1			
e200		800	1000	2	2			1			1
a200	Y	600	600	3	2			1			
c200		600	800	3	2	1		1			
d500		800	1300	3						1	
d100		700	800	3			1			1	
c200		800	1000	3		1	1				
a200	Y	500	500	4		1	1				
d300		500	800	4		1	1		1		
c200	Y	800	800	4		2	1		1		
a200	Y	800	800	5		2	1		1		
d500		800	1300	5						1	
a200	Y	700	700	6						1	
c200		700	900	6		1				1	

```
a300.html" TCP_DONE [    ] [4115] 300176
a200.html" TCP_DONE [    ] [2174] 200176
a100.html" TCP_DONE [    ] [600] 100176
b200.html" TCP_DONE [    ] [961] 200176
c200.html" TCP_DONE [    ] [1354] 200176
a100.html" TCP_DONE [R S ] [640] 100176
b200.html" TCP_DONE [RFS ] [1937] 200176
a200.html" TCP_DONE [RFS ] [1535] 200176
c200.html" TCP_HIT [RFSL] 200176
a300.html" TCP_DONE [   L] [3426] 300176
d300.html" TCP_DONE [    ] [3414] 300176
a100.html" TCP_DONE [  S ] [699] 100176
a200.html" TCP_DONE [ F  ] [1451] 200176
b200.html" TCP_DONE [    ] [1626] 200176
c200.html" TCP_DONE [ FS ] [2135] 200176
a300.html" TCP_DONE [    ] [3777] 300176
d300.html" TCP_DONE [   L] [2401] 300176
```

Figure B.2: Trace file 2. Demonstrating the SIZE removal algorithm.

Table B.3: Step by step illustration of how the SIZE removal algorithm removes documents.

File & Size	SIZE hit?	Cache size	Cache size w/new file	Remove documents?	Documents cached & removal order
a300		0	300		a300
a200		300	500		a300, a200
a100		500	600		a300, a200, a100
b200		600	800		a300, a200, b200, a100
c200		800	1000	a300	a200, b200, c200, a100
a100	Y	700	700		a200, b200, c200, a100
b200	Y	700	700		a200, b200, c200, a100
a200	Y	700	700		a200, b200, c200, a100
c200	Y	700	700		a200, b200, c200, a100
a300		700	1000	a200, b200	a300, c200, a100
d300		600	900		a300, d300, c200, a100
a100	Y	900	900		a300, d300, c200, a100
a200		900	1100	a300, d300	a200, c300, a100
b200		500	700		a200, b200, c200, a100
c200	Y	700	700		a200, b200, c200, a100
a300		700	1000	a200, b200	a300, c200, a100
d300		600	900		a300, d300, c200, a100

B.4 LAT Verification

The trace file shown in Figure B.3 illustrates the trace file used to verify the LAT removal algorithm. This algorithm is impossible to demonstrate with a reproducible log, due to the variation in download time from different servers. For this algorithm to work effectively many different servers must be used for the cached documents. In this case only one server has been used, which changes the behavior of the removal algorithm. The LAT algorithm removes the documents with the shortest download time. When only one server has been accessed the *smallest* documents will be removed first. This is generally believed to be a *very* poor removal algorithm, as it is operating in an opposite manner to the SIZE algorithm, which removes the largest documents first. However, this test is only used to produce reproducible results and to verify that the algorithm works as expected. Table B.4 shows step by step how the documents are removed.

```
a300.html" TCP_DONE [    ] [3336] 300176
b300.html" TCP_DONE [    ] [3371] 300176
a100.html" TCP_DONE [    ] [548] 100176
a100.html" TCP_HIT [RFSL] 100176
a200.html" TCP_DONE [    ] [1093] 200176
b100.html" TCP_DONE [    ] [521] 100176
a300.html" TCP_DONE [   L] [3608] 300176
b300.html" TCP_DONE [ F L] [3199] 300176
a500.html" TCP_DONE [    ] [7629] 500176
a100.html" TCP_DONE [  S ] [723] 100176
a200.html" TCP_DONE [    ] [1255] 200176
b200.html" TCP_DONE [    ] [1439] 200176
c200.html" TCP_DONE [    ] [1200] 200176
d200.html" TCP_DONE [    ] [1265] 200176
a500.html" TCP_DONE [   L] [7707] 500176
```

Figure B.3: Trace file 3. Demonstrating the LAT removal algorithm.

Table B.4: Step by step illustration of how the LAT removal algorithm removes documents.

File & Size	LAT hit?	Cache size	Cache size w/ new file	Remove documents?	Documents cached & removal order
a300		0	300		a300
b300		300	600		a300, b300
a100		600	700		a100, a300, b300
a100	Y	700	700		a100, a300, b300
a200		700	900		a100, a200, a300, b300
b100		900	1000	a100, a200,	b100, a300, b300
a300	Y	700	700		b100, a300, b300
b300	Y	700	700		b100, a300, b300
a500		700	1200	b100, a300, b300	a500
a100		500	600		a100, a500
a200		600	800		a100, a200, a500
b200		800	1000	a100, a200,	b200, a500
c200		700	900		b200, c200, a500
d200		900	1100	b200, c200	d200, a500
a500	Y	700	700		d200, a500

Appendix C

Confidence Interval Data

Data from workloads VTCS2, VTCS3, and the accelerated workloads VTCS2, BULOG, and VTLIB are recorded. The values in the first four columns are the average percentage above or below the mean for each performance measure for each set of data. The fifth column is the actual mean value over each data set, either 24 hours or 5000 URLs for experiment two and three respectively. The mean for HR and WHR is the percentage achieved by the algorithms, the mean for the TIME is the mean time for each document, in seconds per document. The data sets are obtained using equation (5.4) for experiment two and equation (5.5) for experiment three. These data sets are then normalized, using equation (5.3), to create the first four columns.

C.1 Data from the VTCS2 Workload — Experiment Two

HR-R	HR-F	HR-S	HR-H	Mean
-0.61	-2.88	1.65	1.84	20.09
-7.18	-3.21	5.32	5.07	27.57
-1.27	-3.81	3.09	2.00	27.30
-1.65	-3.82	2.71	2.77	29.67
-4.44	-3.14	3.50	4.08	25.71
-6.19	-4.20	4.47	5.92	23.84
-4.11	-4.67	2.26	6.51	30.43
-4.71	-4.14	1.57	7.28	19.78
-8.50	-3.49	4.23	7.75	20.87
-1.09	3.58	-3.73	1.24	35.16
-12.80	6.24	-3.14	9.71	27.21

WHR-R	WHR-F	WHR-S	WHR-H	Mean
2.80	-1.05	-1.32	-0.44	21.61
-3.10	-0.31	1.18	2.23	30.88
4.71	2.65	-5.91	-1.45	14.35
1.90	0.63	-4.10	1.57	23.57
4.63	8.24	-15.35	2.48	14.64
5.09	7.38	-14.88	2.40	18.80
8.88	4.28	-15.24	2.08	25.28
5.23	8.60	-17.00	3.18	19.41
9.66	12.04	-22.21	0.51	20.05
11.28	16.72	-23.09	-4.91	26.08
-1.95	16.79	-18.49	3.65	17.95

TimeR	TimeF	TimeS	TimeH	Mean
-0.15	0.54	-0.12	-0.27	1.21
1.22	1.28	-1.21	-1.30	1.85
0.11	0.58	-0.14	-0.55	2.72
0.12	1.22	-0.45	-0.88	2.85
-0.04	-0.60	2.05	-1.41	2.41
0.79	0.39	0.11	-1.29	3.45
-1.91	2.87	0.83	-1.79	3.94
1.31	0.55	0.77	-2.63	2.56
-0.13	-0.89	2.69	-1.67	3.65
-0.35	-4.06	5.53	-1.12	3.68
2.49	-1.45	1.27	-2.31	4.97

C.2 Data from the VTCS3 Workload — Experiment Two

HR-R	HR-F	HR-S	HR-H	Mean
-1.34	-3.23	2.11	2.45	29.10
-4.19	-3.28	3.89	3.58	24.24
-8.06	-0.07	2.17	5.96	21.03
-8.42	-0.09	7.05	1.46	22.80
-7.75	-2.89	7.20	3.43	23.27
-2.42	0.61	-0.67	2.48	27.76
-3.51	-3.17	1.47	5.22	22.20
-1.05	-2.23	2.18	1.10	38.03
-8.87	-5.89	8.06	6.70	24.36
1.28	-1.08	-1.48	1.28	22.53
2.28	-3.20	0.46	0.46	19.69
-4.60	-2.19	2.64	4.15	14.23
-1.25	0.29	0.67	0.29	19.51
0.23	-0.27	-0.02	0.06	24.10
0.54	-0.80	0.02	0.24	25.43

WHR-R	WHR-F	WHR-S	WHR-H	Mean
1.42	0.99	-2.67	0.26	28.95
5.61	5.64	-10.14	-1.10	18.16
11.97	14.73	-23.70	-2.99	15.59
3.89	10.64	-9.30	-5.23	11.36
1.41	4.41	-7.30	1.48	17.71
5.11	9.11	-13.34	-0.88	21.32
3.69	3.62	-10.34	3.04	13.57
2.67	1.85	-5.49	0.98	41.27
-1.43	0.67	-3.33	4.10	22.35
5.90	2.71	-9.14	0.53	9.51
1.61	-0.47	-0.80	-0.34	7.58
-0.65	2.14	-3.78	2.29	3.27
-0.07	1.29	0.05	-1.27	32.04
0.89	0.67	-0.85	-0.71	21.62
0.46	-0.18	-0.38	0.10	20.05

TimeR	TimeF	TimeS	TimeH	Mean
3.18	3.65	-2.95	-3.88	2.53
-0.38	0.38	0.65	-0.65	4.08
2.21	-0.97	0.05	-1.29	3.42
1.63	-1.06	-0.03	-0.53	4.26
1.99	-1.25	0.69	-1.42	3.66
-0.36	-0.96	2.25	-0.94	1.80
0.17	0.18	0.31	-0.66	2.44
-0.32	2.02	0.04	-1.74	2.19
1.04	0.83	-0.02	-1.85	2.89
-0.61	0.00	0.97	-0.37	2.47
-0.09	0.08	-0.02	0.03	21.60
0.64	0.40	0.18	-1.22	5.36
0.11	0.04	-0.20	0.06	3.19
-0.07	0.01	0.07	-0.00	2.70
-0.25	0.28	0.13	-0.16	3.08

C.3 Data from the Accelerated VTCS2 Workload: LRU, LFU, SIZE, HYB — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
-0.11	-0.58	0.35	0.34	36.83
-6.13	-1.61	4.41	3.34	33.93
-2.20	-4.86	3.81	3.25	39.15
-1.50	-5.64	3.75	3.39	33.20
0.84	-4.94	2.37	1.73	41.02
1.31	-7.73	2.60	3.82	38.63
-1.03	-4.98	2.71	3.30	38.70
-3.79	-7.68	5.34	6.13	30.59
-1.44	-6.49	4.43	3.51	31.14
-1.01	-5.58	3.94	2.66	33.03
-5.77	-7.69	7.24	6.22	22.04
-1.23	-1.81	2.82	0.22	36.24
-1.64	-0.83	1.64	0.84	43.11
-1.27	-2.28	4.00	-0.46	35.30

WHR-R	WHR-F	WHR-S	WHR-H	Mean
16.54	8.57	-12.93	-12.18	28.49
-4.84	-0.56	4.19	1.21	36.32
2.89	1.14	-3.43	-0.60	37.89
2.16	-4.77	-0.03	2.64	6.43
6.25	5.95	-16.45	4.25	25.76
3.37	8.09	-11.10	-0.36	30.09
13.40	1.03	-10.90	-3.53	20.76
10.17	6.32	-18.92	2.43	18.17
11.77	10.60	-16.30	-6.07	24.91
13.03	4.60	-18.39	0.77	23.17
14.17	16.47	-11.85	-18.79	14.97
17.42	10.68	-19.73	-8.36	29.00
12.73	6.85	-8.32	-11.26	25.84
14.69	7.77	-18.54	-3.93	15.39

TimeR	TimeF	TimeS	TimeH	Mean
0.99	-6.03	2.98	2.06	2.81
6.19	-0.68	-3.81	-1.70	1.76
12.06	9.83	-10.56	-11.33	2.18
3.78	0.36	-1.59	-2.55	3.19
1.13	2.94	4.03	-8.10	1.96
-4.35	-4.54	7.23	1.66	2.22
2.62	2.07	-2.58	-2.11	2.20
-2.73	-3.53	-4.28	10.54	2.25
0.33	3.33	-0.89	-2.77	1.89
2.09	8.82	-1.08	-9.83	1.41
-1.38	2.12	-3.15	2.41	2.31
0.33	0.48	-4.24	3.43	1.17
2.29	3.69	0.13	-6.11	1.45
0.56	15.37	-9.86	-6.08	1.67

C.4 Data from the Accelerated VTCS2 Workload: Different HYB Constants — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
-0.16	-0.09	0.26	-0.00	39.93
-0.65	0.07	0.13	0.44	35.14
-0.45	0.11	-0.07	0.41	40.83
-1.12	0.05	0.81	0.26	29.03
0.10	0.00	-0.68	0.57	28.51
-0.25	-0.16	0.53	-0.12	41.74
-0.90	-0.22	0.41	0.72	41.66
-0.06	-0.53	0.57	0.02	32.74
1.38	0.37	0.78	-2.53	28.16
-1.46	-0.61	-0.15	2.22	38.09
-0.56	0.07	1.01	-0.53	22.66
-1.48	-0.61	2.20	-0.11	34.87
-0.37	0.60	0.76	-0.99	44.96
-0.46	-0.25	1.53	-0.82	35.80

WHR-R	WHR-F	WHR-S	WHR-H	Mean
-2.68	-0.20	5.44	-2.56	22.98
-11.32	3.92	2.85	4.55	38.79
-1.33	-1.29	-0.07	2.69	40.38
-3.10	-1.19	0.66	3.62	18.00
3.78	-5.33	0.09	1.46	26.96
-6.95	-2.49	11.18	-1.74	28.80
0.70	-6.71	2.61	3.40	21.27
1.32	-5.83	1.26	3.25	20.64
4.86	5.73	-4.73	-5.87	20.64
-4.57	0.83	0.71	3.03	27.27
-1.04	0.57	2.19	-1.73	13.53
-2.55	-0.88	5.92	-2.49	25.09
-1.47	-2.67	4.78	-0.64	26.22
2.23	-3.99	-1.63	3.39	21.02

TimeR	TimeF	TimeS	TimeH	Mean
2.30	0.38	-0.67	-2.01	3.27
9.38	1.04	-5.27	-5.14	2.22
-0.45	-1.73	-0.21	2.39	1.68
-0.79	-2.80	2.26	1.33	2.16
-6.52	-0.98	8.33	-0.83	1.87
-4.60	1.88	1.91	0.81	1.79
1.82	-1.04	-1.30	0.52	1.66
2.03	6.77	-6.13	-2.67	1.34
-3.53	-3.81	6.50	0.83	1.82
15.53	-6.56	-1.67	-7.30	1.29
1.44	-5.29	0.68	3.16	1.65
-0.54	0.82	-0.52	0.25	1.56
0.25	-7.02	2.18	4.60	1.49
-3.01	-10.73	-1.18	14.93	1.42

C.5 Data from the Accelerated BULOG Workload: LRU, LFU, SIZE, HYB — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
0.41	0.06	-0.24	-0.23	75.17
-0.32	-0.13	0.02	0.43	71.47
-0.16	-0.19	0.02	0.33	85.17
-0.28	0.11	0.65	-0.48	50.88
-1.01	0.39	0.13	0.48	60.97
0.13	-0.36	0.48	-0.25	72.90
-0.73	-2.77	1.77	1.72	54.39
-1.39	-1.00	1.32	1.06	68.24

WHR-R	WHR-F	WHR-S	WHR-H	Mean
2.13	-1.35	-1.06	0.28	62.84
-1.33	0.32	0.82	0.19	70.98
1.86	-1.67	-2.07	1.87	71.89
-0.59	-0.95	0.64	0.90	29.31
-2.24	0.21	3.28	-1.25	70.94
2.27	0.87	-6.38	3.24	71.27
7.71	-0.66	-7.17	0.12	48.14
-0.06	-0.70	-0.57	1.34	61.85

TimeR	TimeF	TimeS	TimeH	Mean
3.75	-6.36	0.22	2.39	0.24
11.09	5.99	-9.32	-7.75	0.25
-5.10	22.30	-15.06	-2.14	0.11
11.52	-5.67	0.64	-6.48	0.54
-1.26	-3.08	-5.65	9.99	0.57
-1.46	8.30	-3.95	-2.90	0.23
14.98	3.20	-6.20	-11.98	0.66
8.00	5.80	-9.26	-4.54	0.25

C.6 Data from Accelerated BULOG Workload: Different HYB Constants — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
0.46	0.16	-0.27	-0.35	75.02
0.06	0.63	-0.26	-0.44	72.35
0.35	0.16	-0.23	-0.28	83.69
0.06	-0.29	-0.47	0.70	56.66
0.41	0.35	-0.20	-0.56	61.18
0.06	-0.07	0.10	-0.08	77.06
0.22	0.17	-0.24	-0.15	60.44
0.08	-0.25	-0.25	0.42	70.70

WHR-R	WHR-F	WHR-S	WHR-H	Mean
-1.03	3.93	-1.99	-0.91	59.97
0.27	1.07	0.50	-1.84	74.68
1.38	-1.58	0.83	-0.63	69.74
0.70	-1.49	0.68	0.12	31.13
0.41	1.12	-2.19	0.66	74.62
2.35	-6.96	-1.48	6.09	73.58
6.11	-1.79	-3.11	-1.21	48.71
-1.40	2.00	-0.56	-0.04	61.50

TimeR	TimeF	TimeS	TimeH	Mean
0.65	1.56	-1.11	-1.10	0.29
0.07	1.79	-9.52	7.66	0.23
-5.34	9.65	2.66	-6.96	0.17
-7.71	-4.79	5.38	7.12	0.46
-3.96	-0.68	4.31	0.32	0.48
2.13	3.01	-4.66	-0.47	0.21
-2.53	3.54	2.35	-3.36	0.62
2.76	0.35	-4.61	1.50	0.26

C.7 Data from Accelerated VTLIB Workload: LRU, LFU, SIZE, HYB — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
0.33	-0.07	-0.27	0.01	34.29
-0.41	-0.14	0.30	0.26	33.15
-1.94	-2.30	2.83	1.41	30.53
-1.72	-6.21	3.86	4.07	32.17
-2.72	-4.05	3.79	2.97	34.23

WHR-R	WHR-F	WHR-S	WHR-H	Mean
5.74	6.67	-19.92	7.52	32.27
-1.64	-3.64	3.85	1.43	31.89
6.10	-3.77	-4.37	2.04	31.18
2.34	1.34	-4.01	0.33	25.28
4.07	0.70	-2.92	-1.84	30.52

TimeR	TimeF	TimeS	TimeH	Mean
-1.98	-2.34	3.22	1.10	1.00
1.60	-3.91	4.19	-1.89	1.32
1.93	3.00	-3.91	-1.03	0.85
0.79	2.00	-1.10	-1.69	1.12
-0.01	4.57	-3.75	-0.81	1.00

C.8 Data from Accelerated VTLIB Workload: Different HYB Constants — Experiment Three

HR-R	HR-F	HR-S	HR-H	Mean
0.20	-0.25	0.08	-0.03	33.62
-0.22	-0.39	0.03	0.59	34.67
-0.20	0.03	0.35	-0.18	33.08
0.70	-1.07	0.87	-0.50	31.29
0.30	-0.82	0.39	0.13	32.79

WHR-R	WHR-F	WHR-S	WHR-H	Mean
-2.52	-15.87	9.17	9.22	32.59
2.18	1.96	0.39	-4.54	31.15
5.42	3.75	2.05	-11.22	32.50
3.50	-0.60	-1.67	-1.23	23.43
1.38	-3.14	0.73	1.02	26.15

TimeR	TimeF	TimeS	TimeH	Mean
3.22	-1.46	-0.51	-1.26	1.76
1.72	8.77	-0.74	-9.75	1.66
3.26	1.63	-11.03	6.15	1.74
-0.34	1.40	1.59	-2.64	2.26
-1.45	3.41	2.49	-4.45	1.75

VITA

Roland Peter Wooster was born in Ely, England, to Shelagh (Kenney) and Geoffrey Wooster, on 30 March 1973. He graduated from the King's School Ely, England in June 1991 with A-levels in maths, further maths, physics and chemistry.

He started his undergraduate studies in computer science at the University of Lancaster, England in October 1991. He spent the second year of his undergraduate degree in a year abroad programme, at Purdue University, Indiana, USA. Subsequently he received a B.S. degree, with 1st class honours, in computer science in June 1995.

He commenced his graduate studies at Virginia Polytechnic Institute and State University, Virginia, USA, in August 1995. During his studies at Virginia Tech. he was elected to membership of Phi Kappa Phi honor society, Upsilon Pi Epsilon honor society for the computing sciences, the Graduate Program Committee and the Graduate Student Committee for the Department of Computer Science.

He will be graduating with a M.S. degree in computer science in December 1996. He will be working for Intel in Folsom, California starting in January 1997.

Roland Peter Wooster