DISTRIBUTED MULTIMEDIA PROXY CACHE REPLACEMENT ALGORITHMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Albert I. Reuther

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2000

ACKNOWLEDGMENTS

First I would like to extend a very special thank you to my advisor, Prof. David G. Meyer. His consistent guidance and encouragement has helped me bring this study to completion, and his high expectations of his work has helped me maintain my high expectations for this study.

Special thanks go to Prof. Mitchell Theys, Christopher Niessen, and Ku-Jei King for their critical analysis and wise counsel on several of my concepts and simulation techniques.

I would like to thank the other members of my thesis committee, Prof. Edward Coyle, Prof. Arif Ghafoor, and Prof. Ray Eberts for their guidance that helped keep my research progressing in the right direction.

I would also like to acknowledge and thank each of the sources for the logfiles used in this study. I would like to specifically thank John Dilley and Martin Arlitt from the Hewlett Packard Labs in Palo Alto, California for providing the HP World Cup 1998 Server Farm logfiles and Don Power, the Chief Technologist at OnCommand Corporation of San Jose, California for graciously providing the OnCommand system trace. The San Jose, Boulder1, and Urbana-Champaign logfiles were provided by the National Laboratory for Applied Network Research which is supported by the National Science Foundation on grants NCR-9616602 and NCR-9521745.

Finally, I would like to express my deepest thanks to Albert H. and Gesche Reuther, Christine and Charlie Chappell, and Kate Miller for their unwavering love, support, and encouragement.

"But one thing I do hold: Forgetting what is behind and straining toward what is ahead, I press on toward the goal to win the prize for which God has called me heavenward in Christ Jesus." Soli Deo Gloria!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Reuther, Albert I., Ph.D., Purdue University, August, 2000. Distributed Multimedia Proxy Cache Replacement Algorithms. Major Professor: David G. Meyer.

Implementation of caches in multimedia proxy servers can significantly improve the delivery of media by reducing media retrieval latency, reducing network bandwidth usage, increasing the number of clients that can be served simultaneously, and increasing the fault tolerance of the server system. An important component of proxy server caches is the cache object replacement algorithm, which determines which objects are removed from the cache when storage space is needed for placing new objects into the cache. These algorithms must effectively predict which currently cached objects are not expected to be accessed in the near future so they can be removed. However, these algorithms must also be fast; i.e. they cannot be too computationally complex, because these algorithms must execute often on caches that usually contain a large number of objects. Current implementations of proxy cache servers (Squid, Inktomi Traffic Server, Novell Internet Caching System, etc.) use the least-recently used cache replacement algorithm, but a number of other object statistics could be used as part of the cache replacement algorithm calculations.

This research presents the fully-associative, variable block size cache replacement algorithm problem as a unique model which illuminates a new method for looking at cache replacement algorithm issues. The study then explores the use of a variety of cache object statistics and how well these other statistics help predict the expected future demand for cached objects. Simulations are used as the method of comparison, and actual and stochastic data from WWW traces, commercial hotel video system traces, and educational multimedia system video traces are used as simulation input. Also, the effectiveness of admission policies and caching large videos is explored.

Not only will the findings from this study impact the World-Wide Web caching infrastructure and distributed multimedia system implementations, but the findings

can also influence the efficiency of distributed databases and other distributed information systems.

1. Introduction

1.1 The Problem

Computers and multimedia are being used to augment educational experiences in exciting ways. For many universities and corporations, one of the most inspiring applications is for just-in-time distance learning. By taking advantage of computer workstations on the learner's desk and a fast computer network, educational multimedia can be delivered directly to the computer desktop.

Local area networks (LANs), implemented with the latest Ethernet 100 Base-T technology, can deliver a modest number of multimedia streams simultaneously. However, current wide area network (WAN) technology cannot accommodate large amounts of educational multimedia per time unit because it requires a great deal of bandwidth to be delivered in a timely manner while maintaining a reasonable quality of service. This is especially evident in the case of multimedia videos. Quality of service (QoS) addresses the resolution and frame rate of the multimedia videos. To provide a picture of where the current technology is, the servers in the Purdue School of Electrical and Computer Engineering Digital Systems/Multimedia Learning Lab are delivering 720 by 480 pixel MPEG2 videos at 30 frames per second, which translates into a network bandwidth consumption of 2.5 to 3.0 megabytes/second per video.

The problem can be examined further using an example. In several years, Purdue University will probably have a central archive of multimedia modules that are served throughout campus to students at their workstations all across the campus.

For the sake of the example, say that this central archive is located in the main library and is implemented with a large number of file/media servers called a server farm. This server farm has access to all of the archived media objects which are stored on hard disks, optical storage devices, and/or tape storage devices. Each building on campus that has computer labs, including residence halls, will then have proxy servers. A proxy server is a server that stores a subset of the media on the main server farm. This media subset is stored on a portion of its hard disk array called the proxy server cache. The proxy servers are connected to the central server farm using WAN technology. All of the workstations within buildings are connected to the proxy servers using LAN technology [Tob95]. Generally, LAN technology is much faster at delivering media than WAN technology, but LAN technology is limited by the distance that it can deliver data. Also, the LAN is only shared by all of the workstations within a building, while the campus WAN is shared by all of the proxy servers on campus, and indirectly by all of the workstations on campus. So if ways can be found to decrease the bandwidth that is used on the WAN by shifting it to LAN traffic, WAN bandwidth will be saved for more user capacity and for other applications.

Say that a student is sitting at a workstation in the engineering library. When that student requests a media object, the engineering library proxy server acts like the central server farm, if it can, by delivering the requested media object from its proxy server cache if a copy of the requested object is available in the cache. If a copy of the requested object is not available in the proxy server cache, then the proxy server requests the media object from the central server farm at the main library. The central server farm then serves the student's request, and the engineering library server must decide whether to place a copy of the media object into its proxy server cache.

This determination of what media objects should be placed in the proxy server cache is the focus of this study. The engineering library server must compare the expected demand for the requested media object to the expected demand of the media objects that the server currently is holding in its proxy server cache. If the expected demand of the requested media object is less than the expected demand of all the other objects in the proxy server cache, then the engineering library server will just deliver the media object to the student's workstation and will forego copying the media object to its proxy server cache. However, if the engineering library server decides to store a copy of the requested media object in its proxy server cache and there is not enough space in the cache to accommodate the requested media object, the engineering library server must decide which media object to evict from its cache. Again this is done by comparing the expected demand for each of the media objects in the cache. The media object with the least expected demand is evicted. The next time that evicted media object is requested in from a engineering library workstation, it must be delivered from the central proxy server and the engineering library server must again decide whether to make a copy of that media object in its proxy server cache.

But this does not yet address how to determine the expected demand for each of the media objects. The expected demand is calculated by algorithms that are generally referred to as cache replacement algorithms. Currently the majority of research effort is being expended on cache block replacement algorithms to increase microprocessor efficiency, to increase database efficiency, and to increase World Wide Web document retrieval efficiency. The research in these areas provide a basis from which to start, but the data that is cached in these applications is far smaller in size than that of the media objects. The sizes of data blocks for microprocessors, databases, and WWW documents are anywhere from a several bytes to several megabytes, while the media objects are often many megabytes and even several gigabytes in size. So the bandwidth cost of making an error in predicting the expected demand of data is much less for the microprocessor, database, and WWW data than for media objects.

The cache replacement algorithms that are currently being researched are generally simple, easy-to-calculate algorithms. The algorithms use such statistics as the time since the last access, number of accesses of the cached data, and document size [Bes97]. In order to keep the calculations simple and fast, usually only one of the above statistics is used.

Because the potential cost of making an error in predicting the expected demand of a media object is much greater than for other applications, it is reasonable to spend more calculation time by using more sophisticated cache replacement policies to give us a greater ability to make better predictions. This study proposes combining several of the above-mentioned statistics to better predict the expected demand. The exploration in this study, though, is not limited to only multimedia systems. A significant portion of this research addresses the issues of Internet proxy caches, since there is still room for improving the cache replacement algorithms used in them.

Regardless of what document statistics are used, it can be expected that the cache replacement algorithms that perform the best are the ones that make replacement choices that best match the underlying client access patterns of given media delivery system.

1.2 Significance of Problem

As alluded to, the bottleneck in delivering multimedia and Web content over networks is the network bandwidth, the amount of data that the network can transmit in a given amount of time, and throughput of the the media server(s) . By caching media objects in proxy server caches closer to the students' access points, a decrease in overall network bandwidth usage will be realized when a requested media object is in the proxy server cache. The data traffic of the heavily shared campus WAN and Internet is decreased whenever a media object can be delivered from the proxy server cache. Of course the traffic on the LAN is not decreased; the media object must still be delivered to the student's workstation. Also, server farm usage is decreased and the load of serving the media is shared between the central server farm and the proxy servers.

Decreasing the data traffic on the campus WAN and Internet and decreasing the central server farm load and origin servers has several implications. The unutilized bandwidth can be used to accommodate more users, and it could mean longer time between major network hardware upgrades. It means that the central server farm and origin servers will also be able to accommodate more clients while providing the clients with faster retrieval time, since the object is being served by the proxy server and not the central server farm and origin servers. Also, the entire delivery system will be more fault tolerant because media objects do not have to be delivered from the origin server, if it is out of service, since any cache proxy server with the object could essentially deliver the object.

Furthermore, this technology does not need to be limited to the virtual classroom. Similar usage patterns can be derived for commercial video-on-demand systems, which are on the horizon for many areas of the United States, Europe, and Asia. These systems allow customers to access movies and other video programs from the comfort of their living rooms and have the selected program delivered on request. By having proxy server caches in neighborhoods, customers can have quick and efficient access to the movies they want when they want to see them [AK96]. Predecessors to these systems are already being implemented by delivering video over the WWW. These systems are being implemented by companies like Apple, Akamai, and Novell with QuickTime TV, RealNetworks with Broadcast.com, and Microsoft [Gro99]. Two of these video systems have even released video proxy cache server software for delivering web videos. Novell with their Internet Caching System server software implements video caching for Apple's QuickTime TV system [Mat99], while RealNetworks has teamed up with Inktomi to deliver RealProxy caching server software [Ink99]. As these companies realize, these systems can benefit from localized proxy cache systems as well. And by implementing the most effective cache replacement policies in these localized proxy servers, the WAN bandwidth is kept lower and customer access times are greatly decreased because most video accesses will not have to be delivered from a more centralized server farm.

And as it has been alluded, the findings of this study should be applied to WWW document retrieval systems. Currently, WWW proxy servers such as NLANR's Squid, Network Appliance's NetCache, Inktomi's Traffic Server, the Microsoft Proxy Server, and others use the least-recently used replacement policy. An initiative has been proposed by HP Labs to include a few other replacement policies into the Squid source code, but an all out comparison has still not been conducted to determine clear performance champions for different client access patterns. The more complex replacement algorithms very well may improve the performance of these systems, especially as these types of systems involve larger sets of data.

1.3 Research Contributions

This research makes several contributions. First, it presents the fully-associative, variable block size cache replacement algorithm problem as a unique model which illuminates a new method of looking at its issues. This research explores the performance of the cache replacement algorithms using a wide variety of traces and situations that are encountered in distributed multimedia systems including web server traces and multimedia system traces. Also, the research is the first public study to actually use video server event traces from an educational multimedia testbed system (the Video Jockey systems) and from a commercial hotel video server system (the OnCommand OCX [OnC99b] system). This study analyzes the statistics of the workload traces, and justifies the need to determine which proxy caching domain in which the proxy caching server will be used by showing the differences in several significant statistics for the different proxy caching domains.

Most previous studies have only compared a few cache replacement algorithms at a time. This research compares most of the cache replacement algorithms from previous studies using a variety of input workload access traces from various regions of the delivery network. Another contribution of this study is the introduction of four new cache replacement policies which are based on observations from other studies and on various workload access traces. The goal of this research determines which of these replacement policies perform best for different types of systems in different proxy caching domains, thereby determining which cache object statistics are most important for determining the future access patterns for various typical system traces. Finally, it explores the effectiveness of using a cache admission policy in determining whether a document should be cached when it is first accessed and explores whether video server traces should be cached in proxy servers or should merely be buffered.

2. Statement of Problem

2.1 Caches in Distributed Multimedia Systems

Since their introduction in the 1960s [Wil65], data caches have made data hierarchies an effective way of improving system performance. The caches are placed between the data storage devices and data consumers. In computer memory systems, this data hierarchy paradigm is implemented with the central processing unit as the data consumer while the main memory is considered the main data storage devices. One or more levels of cache are situated between storage devices and the consumers. The caches strategically store a subset of the information from a higher level with the intent of faster delivery of the data to the consumer while saving interconnect traffic bandwidth between the main data storage devices and the consumer.

Cache memory principles have been used by many more applications than just computer memory systems. They have been implemented in database transaction buffers, operating systems virtual memory direct paging algorithms, computer systems' I/O and disk buffers, and distributed multimedia systems. Distributed multimedia systems include distributed data dissemination hierarchies on the World-Wide Web as well as smaller, networked video-on-demand multimedia systems. The analogy between computer memory systems and server cache systems is illustrated in Figure 2.1. In this thesis, the focus is on these distributed multimedia systems.

Within these distributed multimedia systems, caches can be utilized in four domains: main server proxy caches, network proxy caches, firewall/proxy server caches, and client caches. These four domains are illustrated and enumerated in Figure 2.2.

1. Main server caches are a part of the main server farm that houses the central data storage, and they duplicate a portion of the main server's data to decrease the main server's latency and increase the effective bandwidth that the main

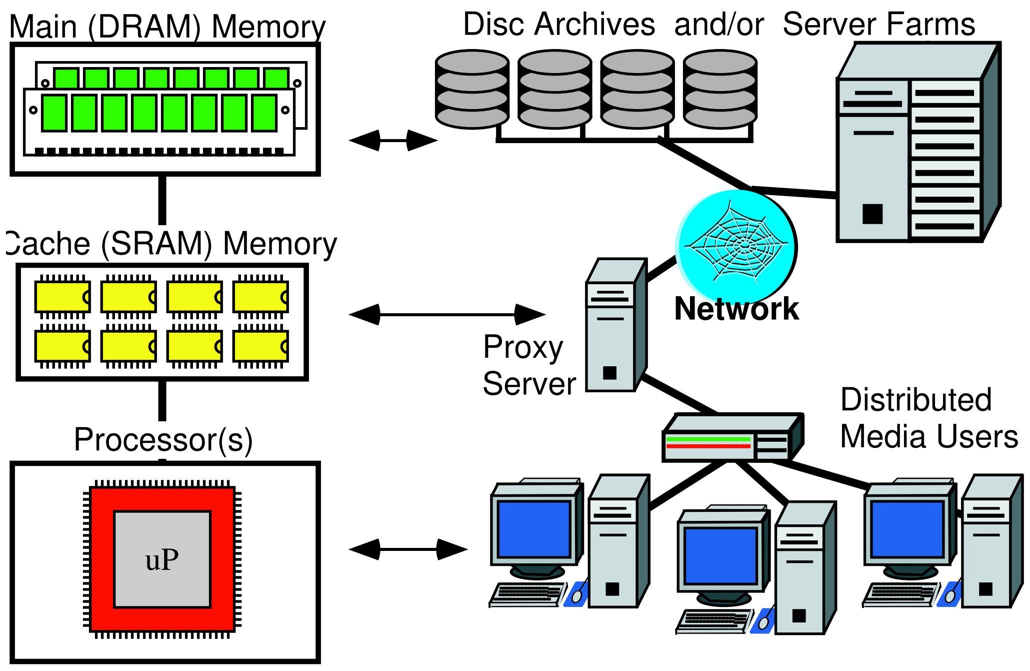


Fig. 2.1. Comparison of a Computer Memory Hierarchy to a Hierarchical Distributed Server Cache System

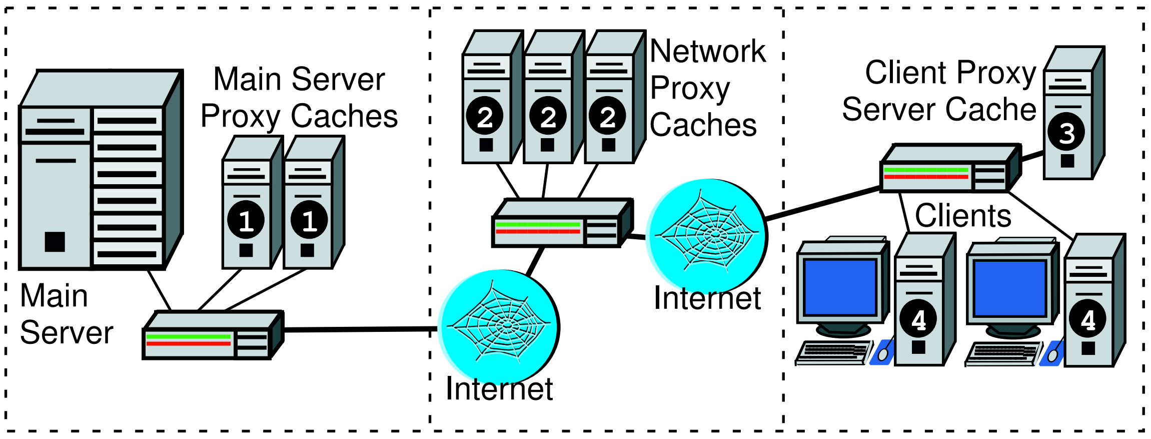


Fig. 2.2. Domains of Distributed Multimedia System Proxy Caches

server can deliver. However, this configuration does not decrease the network bandwidth usage from the main server to the client. The main server caches generally proxy a small working set of documents as compared to the World Wide Web as a whole.

2. Network proxy server caches are placed somewhere on the network between the main server and the client proxy servers, and they are often placed at the interface between the network backbone and the tributary networks. They cache the media requests for all of the clients of its tributary networks thereby saving the latency and network bandwidth from retrieving data from the main servers when cache hits occur. The working set of documents these network proxy servers can cache is the entire World Wide Web, an enormous set!

3. Client proxy server caches are usually placed within the internet service provider's systems, for residential and small business service, or among the firewall servers in corporations and institutions. They cache the media requests for all of the clients of its client networks thereby saving the latency and network bandwidth from retrieving data from the main servers when cache hits occur. The working set of documents these network proxy servers can cache is potentially the entire World Wide Web. But this set is usually more limited since clients with similar jobs, interests, and other such demographics tend to access similar web documents.

4. Finally, client caches are implemented on the users' computers, and they also can save latency and network bandwidth on cache hits. However, when caching large audio and video multimedia files, the client caches are not especially effective because users are not likely to consume the same video or audio file more than once in a short time frame.

2.2 The Path to Hierarchical Distributed Multimedia Systems

Before we proceed with exploring caches in distributed multimedia systems, it is interesting to follow the path that has led to these distributed multimedia systems as a whole. In the early 1990's as computer systems were becoming powerful enough to begin implementing multimedia distribution systems, it became apparent to a number of research teams that multimedia files could not be handled by servers as regular files were handled [GC92, LS93, Has93, DT94]. With normal file servers, the latency of the desired file can be sacrificed, to some extent, as long as the file is exchanged error-free between the file server and the user – accuracy is more important than latency. With multimedia files, these priorities are reversed. Since audio and video frames must be delivered in the proper sequence, each frame must be delivered by a deadline, which must be met to ensure the smooth delivery of the media. If the deadline is missed for a frame or set of frames, it is best for the delivery of the frame(s) to be canceled and dropped, because they are no longer of any use. The frames that follow the dropped frames should then be delivered. The system can then recover from the errors caused by the dropped frames, but the delivery of the multimedia stream as a whole is not delayed. In the aforementioned references, these principles were applied to single servers and server farms with no sense of hierarchy in the system.

Several more studies examined server storage techniques that would better ensure the timely delivery of multimedia files by these hierarchy-less server systems. [RV93, LS93, TP93, TPBG93, DSK94, KCS94, CGM97, SC98] looked at various disk drive array issues that improve the delivery of multimedia files by multimedia servers. Another set of studies examined the use DRAM memory buffers that temporarily store portions of multimedia streams that are enroute from the server's disk drive arrays to the multimedia clients via the server's network interface [NY94, DDM$^+$95, KRT95, OBRS95]. These buffering systems allowed very limited sharing between multimedia clients. This sharing of a buffered multimedia file occurred when multiple clients start retrieving the same file within a short period of time. However, such DRAM buffering presents a significant problem when clients are allowed to pause, fast forward, and review (rewind) video files. The buffer must reset its contents to the new point in the video which may incur a relatively significant latency.

The tutorial article by Gemmell, Vin, Kandlur, Rangan, and Rowe [GVK$^+$95] provides an excellent background on these main server issues. More recently Jack Y.B. Lee discussed the issues involved in implementing parallel video servers as the main video server farm [Lee98].

Meanwhile, the World-Wide Web was developed as a distributed media network. Muntz and Honeyman proposed modifying the Andrews distributed network file system (calling it iAFS) to implement multi-level caches on the World-Wide Web [MH92]. They ran simulations of a hypothetical iAFS used in a network proxy cache and found that it would improve the performance of the main Web servers by "substantially" decreasing the peak request rates at the main file server. Danzig, Hall, and Schwartz researched the use of network proxy servers for FTP traffic on NSFnet [DHS93]. They found that the network proxy servers could indeed decrease the FTP network bandwidth. They, however, did express several concerns about cache consistency across the Web and were not sure that the complexity of maintaining consistency justified the gains found in network proxy caching. Glassman implemented a proxy caching relay server for HTTP and Gopher traffic at the Palo Alto Digital Equipment Corporation site [Gla94]. Glassman found that 30% to 50% of all requests could be serviced from the proxy cache. He did have some issues with determining how long Web documents remained valid (unchanged). The best overview paper on World-Wide Web proxy caches is from Luotonen and Altis [LA94]; it overviews many issues about proxy Web server caches including the effectiveness of proxy server caches (implemented as firewalls) over client caches, the implementation of proxy server caches as both server and client, the estimation of time-to-live of Web objects, and using push caching related Web objects. Though they did not share any experimentation results, they strongly suggested that Web proxy server caches would show noticeable speedups in retrieval time.

Two open source hierarchical Internet object caches have emerged in the past few years. Both Harvest [CDN$^+$96] and Squid [Wes96] allow multiple-level hierarchies across the World-Wide Web. Both have been shown to improve Web object retrieval times. The success of these two open source caches has prompted a number of companies to develop and release their own Internet object caches. These companies include Akamai, Inktomi, Microsoft, Netscape, NetWare, Network Appliances, CacheFlow, InfoMedia, and others.

A good article that discusses the issues of implementing a World-Wide Web server farm is [KMR95]. The article details how the NCSA's server at the University of Illinois at Champaign-Urbana is implemented using a main server farm of Hewlett-Packard workstations running the Andrews distributed network file system [Sat90]. There are two other distributed network file systems that can be used for main server farms: Coda [SKK$^+$90] and the Continuous Media File System, CMFS [AOG92].

During the mid-1990's, two concept articles were published by Prof. P. Venkat Rangan and his research team at the University of California at San Diego that took the hierarchical distribution network to distribute audio and video content multimedia.

In the first of these two articles, Ramanathan and Rangan proposed Personal Service Agents (PSA's) that implement intelligent caching strategies to strategically store multimedia content in the hierarchical network [RR94]. These PSA's take into account the cost of storage at a network proxy cache server versus the cost of retransmitting the content on the network. In the article, two significant assumptions are made. The first assumption is that of full program caching. That is, no longer are only portions of a multimedia program being buffered, but rather the entire program is cached in a network proxy server cache. This implies that a greater number of clients could share the same program from the network proxy server cache, but it also means that much more storage capacity is required. The second assumption is that the client multimedia program requests are known ahead of time; they are not on-demand requests. This allows optimal or near optimal placement and scheduling of programs in the network. With the current technology, the assumption of full program caching is realistic for most multimedia hierarchical networks. But the assumption of knowing client request *a priori* is unrealistic, because it demands too much planning on the part of the clients. The concepts of [RR94] were patented in

[Ran96].

In the second of these concept articles, Papadimitriou, Ramanathan, and Rangan further illuminate the theoretical implementation of the PSA's and how the PSA's schedule multimedia programs as the programs are cached across the hierarchical network [PRR94]. The article also assumes full program caching and *a priori* knowledge of client requests. The content of this article is also expanded in a patent [PR97].

Around the same time, Tobagi proposed and began implementing a hierarchical network multimedia system for multimedia education on a university campus. [NJT93] discusses the issues involved in implementing computer networks on campus for the use of delivering educational multimedia. Then in [Tob95], Tobagi gave an overview of the entire campus system for realizing a multimedia education system for an entire university campus, including the main servers and proxy servers, the client stations, and the multimedia database.

On the commercial side, the success of Web document caches has driven several companies to develop Web-based video and media proxy caches. RealNetworks has worked with Inktomi to release a Inktomi Traffic Server based cache software that caches entire RealVideo and RealAudio files [Ink99]. Also, RealNetworks has an agreement with Akamai Technologies for streaming video and audio content [Aka99b]. Microsoft joined forces with InfoLibria to collaborate on the InfoLibria MediaMall software cache. MediaMall caches Microsoft VideoPlayer videos [Inf99]. And finally, Apple has partnered with Akamai and Netware to cache QuickTime video and audio documents and streams within the Internet network [Aka99a].

Now that we have a better understanding of what has been explored in the area of distributed multimedia systems, we can further explore the role of cache replacement algorithms for the systems.

2.3 Cache Replacement Algorithms

In any caching system, the cache replacement algorithm is critical in implementing an effective cache. As was previously explained, the cache temporarily stores a subset of the working set of objects of the entire system. Since it only caches a subset of the whole, the implication is that the cache will become full when its capacity is completely used. But the cache cannot just stop accepting new objects; it must determine what objects to remove from the cache to make room for the new ones. This is where the cache replacement algorithm is used.

The cache replacement algorithm should predict the expected demand of cachable objects by approximating the future client access patterns to be most effective. This prediction must be done based on cache object statistics from past accesses. Usually the new object is not included in this prediction process, since it is assumed to be more valuable than the objects that are currently in the cache. However, if the prediction includes the newly accessed object, the cache is said to implement an admission policy.

Three fairly simple cache replacement algorithms, random, first-in first-out (FIFO), and least recently used (LRU), have been studied and compared extensively. More recently, other object statistics have been considered for making the replacement decision such as the frequency of use (least frequently used - LFU) [RD90, PR94], the size of the cached object [WAS$^+$96], the network and server bandwidth needed to deliver the object [TVDS98], and the type of object that is being cached [TVDS98, AW97]. Many of these object statistics have been combined in replacement decisions, though there are many others that have not been explored. Usually though, these more complex replacement algorithms have only been compared to LRU, FIFO, and random. Also, many of the studies have used few and limited workload traces to evaluate the performance of these replacement algorithms. Hence this research explores taking advantage of multiple cache object statistics by comparing a large number of cache replacement algorithms using a variety of workload traces from various diverse sources. Furthermore, by analyzing the existing replacement algorithms and by statistically analyzing the workload traces used in this study, several more replacement algorithms are proposed and compared to the existing ones. It is expected that by using strategic information of the statistics of the cache objects, more effective decisions can be made in replacing cache objects. The progression of developments in cache replacement algorithms are discussed in Chapter 3.

Fig. 2.3. Knapsack-carrying Thief with Proxy Cache Server

2.4 The 0-1 Knapsack Problem and Cache Replacement Algorithms

The distributed multimedia system caches are fully associative caches since an incoming object can displace any of the objects currently being stored in the cache when there is not enough storage space for the incoming object. (In caches that are not fully associative, an incoming object can only displace a subset of the currently cached objects.) Since these caches are fully associative and the cached objects generally have variable sizes, we can think of the caches as instances of 0-1 knapsack problems as proposed by [TVDS98]. As it is presented in [CLR90], the 0-1 knapsack problem places a thief with a knapsack in a store that has a set of objects. The objects each have a associated worth and weight. The thief wants to get away with the most valuable set of objects but can only carry a certain weight. In its purest form, with the object worth and weights being non-negative, real numbers, the 0-1 knapsack problem is NP-complete [GJ79]. Since [CLR90] states that the worth and weight of the objects and the knapsack carrying capacity of the thief are both expressed as integers, we shall assume that the worth of the objects can actually be expressed as non-negative real numbers.

Mapping the 0-1 knapsack problem to the caching problem, the thief's knapsack carrying capacity is equivalent to the storage capacity of the disk cache of the server. The worth of the store's objects is calculated by the cache replacement algorithm. To help further this analogy, we shall call the media object worth its cache replacement score (CRS). The concept of the cache replacement score was first introduced by Timos K. Sellis in [Sel88] as an intelligent caching technique for relational database systems. It was first introduced to distributed multimedia caching in world-wide web servers in [TVDS98], and there it was called the "goodness value". The cache replacement score was explored without knowledge of the [Sel88] article in [Reu96], where the concept was used to compare several cache replacement algorithms on stochastic traces of educational video-on-demand systems.

In the server caching problem, the thief's knapsack dilemma must be solved each time the cache does not have enough capacity to store the media object that was most recently accessed. At this point, the CRS of each of the cached objects is calculated. The cache makes room for the new object by removing the lowest scoring cached object. If there still is not enough storage space in the cache, it removes the next lowest scoring cached object, and so on until the most recently accessed object can be accommodated in the cache. So essentially, each cache object's CRS is calculated, the objects are sorted, and then removed in ascending order until the new object can be accommodated. This sorting of objects is similar to that used in[WAS$^+$96]. This would be equivalent to the worth of the objects of the thief and the store changing every night. Each night the thief returns to the store and picks an object off of the store's shelves that is assumed to be a higher value than some of the objects in his knapsack. (If the wolf actually calculates the worth of the new object, then he is employing an admission policy to his knapsack.) So he discards the objects with the lowest worth until he can accommodate this new object. [CLR90] discusses this greedy approach to solving the 0-1 knapsack problem. This greedy approach, however, is not optimal.

A dynamic programming algorithm as described in [CLR90] solves the 0-1 knap-

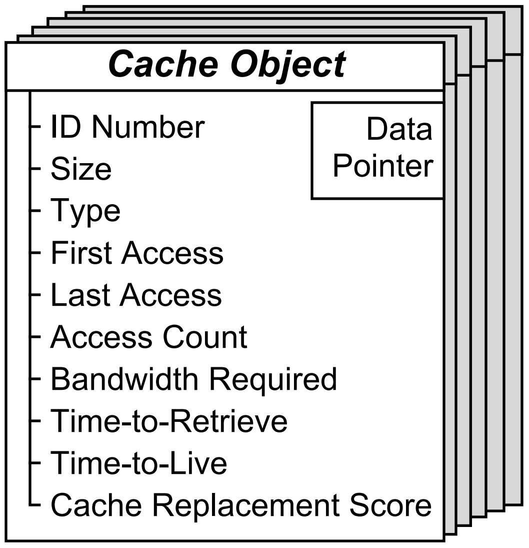


Fig. 2.4. A Cache Object Data Structure and Associated Data Members

sack problem optimally, but in terms of the caching problem, it requires knowledge of future accesses of media. Basically, the dynamic programming algorithm selects the objects that are currently in the cache as the set of objects from the entire set of system media objects that will be needed to satisfy the requests that are coming up in the future. But, unless the cache is knowledgeable of future accesses by implementing user reservation of objects, these object statistics are not available. These are media-on-demand systems in which users shouldn't have to preorder their media wants. Therefore, we shall explore the greedy approach which can be implemented by using only past and present statistics of the objects.

For most existing cache block replacement algorithms, the concept of a cache replacement score can be used. The CRS data member is associated with each object in a cache (see Figure 2.4), and its value is used as the criteria to determine which objects will remain in the cache and which objects will be removed. It is should be kept as a floating point number, and the cache object with the lowest score gets replaced. The cache score algorithm is a method that is associated with an entire cache or a portion of a cache.

2.5 Calculating the Cache Replacement Score

Using the cache replacement score paradigm for determining cache object eviction offers some advantages. We can develop simple CRS algorithms for simple replacement algorithms and combine those simple components to create more complex cache replacement score equations. This allows greater flexibility for determining the CRS. By including more statistics of the cache object in the determination of cache removal, these more complex cache replacement algorithms are expected to better predict the expected demand for objects in the cache. And by better predicting the expected demand of objects in the cache, the cache miss rate and cache hit latency will decrease, while also decreasing the network bandwidth usage. Also, different cache score equations can be used for different file types, different types of objects, or differently sized objects, which could further reduce the cache miss rate.

The three most common cache replacement algorithms are random (RND), least recently used (LRU), and first-in, first-out (FIFO) [Smi82]. The RND, LRU, and FIFO replacement algorithms can be defined in terms of this CRS concept. With the RND algorithm, the score for each object in the cache is determined by uniform random number generator. For the sake of generality, we can let the uniform random number be in the range $[0, 1]$, and the CRS, s, for cache object i, becomes

$$s_i^{RND} = rand(\cdot). \tag{2.1}$$

Each cache object, i, has a data member l_i that records when the object was last accessed, and another data member e_i that records when the object was most recently entered in the cache. Both of these can be recorded as clock time in seconds or as relative cache access counts when the object was accessed or entered in the cache; it generally does not matter which is used since both are non-decreasing. With t being the current time (or current cache access count), the LRU CRS for object i is calculated as

$$s_i^{LRU} = 1/(t - l_i), \tag{2.2}$$

and the FIFO CRS is calculated as

$$s_i^{FIFO} = 1/(t - e_i). \tag{2.3}$$

The differences in time are in the denominator because the lowest scoring objects are replaced first so larger time intervals need to have lower scores.

Many other object statistics can be used in the CRS calculation including the object type (text, graphics, audio, video, etc.), the object's size (or the logarithm of the object's size) [WAS+96], the time-to-retrieve (TTR) request [BH96], the time-to-live (TTL) estimates [BH96], and the frequency of use (LFU) [RD90, PR94].

Since the lowest scoring cached object is the one that will be removed from the cache when more storage space is needed, we need to write the CRS equations to reflect this order of priorities. For different object types, we can assign different priorities to the different types, i.e., for text, $s = 1$, for graphics, $s = 2$, etc. The statistic of object size, z, is usually stored as it's size in bytes, kilobytes, or megabytes. With z, we have the freedom to determine whether we want to make smaller or larger files to be removed first. This is achieved by

$$s_i^{size} = z_i, \tag{2.4}$$

with which small files are removed first, or

$$s_i^{size} = 1/z_i, \tag{2.5}$$

with which large files are removed first. These equations can also be written using the logarithm (usually base 2) of the size:

$$s_i^{size} = \log_2(z_i), \tag{2.6}$$

with which small files are removed first, or

$$s_i^{size} = 1/\log_2(z_i), \tag{2.7}$$

with which large files are removed first. These logarithms are not used on their own but are used in combination with other statistics in more complex cache replacement algorithms.

The cache servers should minimize the time required to load the objects, thereby implying that network traffic is minimized. Hence, the longer it takes to load an object into cache, the harder it should be to remove it from the cache. The time-to-retrieve, *ttr*, should be in the numerator of the score calculation:

$$s_i^{TTR} = ttr_i. \tag{2.8}$$

As with time-to-retrieve, time-to-live should be in the numerator:

$$s_i^{TTL} = ttl_i. \tag{2.9}$$

The time-to-live measure is an active research area in itself, and it tries to estimate how much longer a given object will be valid (i.e., will not be change at the origin server). So the longer it is expected to be valid, the higher its score. However, determining the time-to-retrieve and time-to-live are difficult to measure and gain accuracy. For instance, for time-to-retrieve values, should the value be used when the object was actually retrieved or the current value if the object were retrieved again? Using the value from when the object was originally accessed could be very inaccurate; if the network was very busy at retrieval time and then becomes far less busy, the original retrieval time is also very inaccurate and cannot be compared to objects that were retrieved when the network was less congested. Current retrieval times could be gained by conducting origin server ping requests, but that would be a great waste of network and server resources and would completely disregard the download times from other caches in the network. Also, ttl and ttr are not recorded in any proxy trace files. Therefore, the two statistics are not used in the simulation portion of this dissertation study. However, the simulator has been written to incorporate these statistics if they become available in trace files.

For the least frequently used (LFU) algorithm, a count of accesses, a_i, for object i is kept. Since we want to keep more popular objects in the cache, the score is calculated as:

$$s_i^{LFU} = a_i \tag{2.10}$$

If the equation were left at this, it would take a long time for a popular object to be removed from the cache, even after its popularity had long passed. Hence, this algorithm is usually augmented with an interval parameter, A_{max}. Every time A_{max} aggregate cache references have occurred, each object's a_i is halved [RD90, EH84].

2.6 More Complex Cache Replacement Scores

Several of the above cache replacement score algorithms produce good results, but several studies have found that combining these simple equations can decrease miss rates. [EH84] combines the FIFO and LFU equations to create the least reference density (LRD). By combining Equations 2.3 and 2.10, which calculates a density measure of how many accesses occur per time period since the object was most recently loaded into the cache. For object i, it is calculated by

$$s_i^{LRD} = \frac{a_i}{t - e_i}. \tag{2.11}$$

Several studies have taken LRU as the basis for their more complex cache replacement algorithms. LRU-MIN [ASA+95], when it needs to remove cached objects to make room for object j, takes all of the objects larger than object j in the cache and removes objects using LRU. If there are no more objects larger than object j, it then tries to remove objects that are larger than $1/2 \times size_j$, then larger than $1/4 \times size_j$, and so on. To translate this into a CRS equation, we can take advantage of base two logarithms. The LRU equation in Equation 2.2 will always be in the range $0 < s_i \leq 1$, and we can calculate the LRU-MIN CRS as

$$s_i^{LRU-MIN} = \frac{\lfloor \log_2 \frac{size_j}{size_i} \rfloor}{t - l_i} \quad j \neq i. \tag{2.12}$$

The SPACExAGE algorithm was shown to be effective in mass storage systems [Smi81, LRB82]. A modified version of the algorithm was found to outperform the LRU algorithm in [Red97]. In the CRS equation, SPACExAGE combines Equations 2.2 and 2.5 in one equation:

$$s_i^{SpacexAge} = \frac{1}{z_i \cdot (t - l_i)}. \tag{2.13}$$

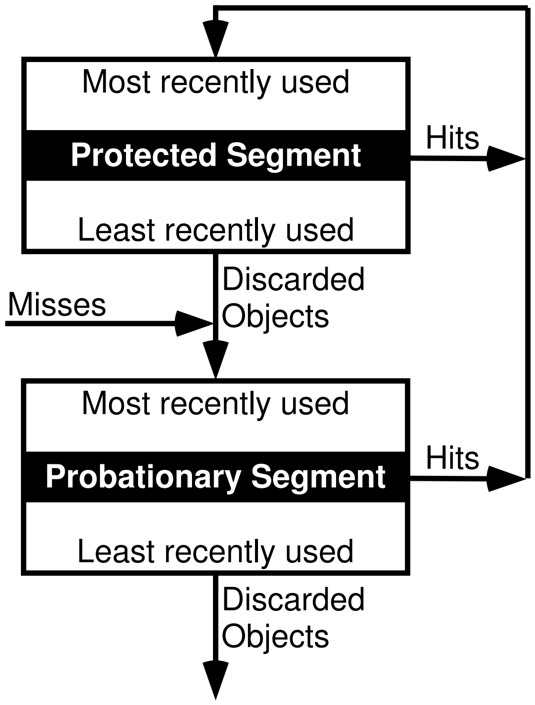


Fig. 2.5. Logical Flow of SLRU Cache Lines

The AVI-MRU algorithm [Red97] was shown to perform better than LRU by making audio and video files a lower priority than all other file types. It should be noted that this performance advantage was in *file miss rate*, not in *byte miss rate*. This algorithm can be formulated as:

$$s_i^{AVI-MRU} = \frac{1}{f(type_i) \cdot (t - l_i)}, \tag{2.14}$$

where $f(type_{a/v}) > f(type_{other})$.

As these cache replacement scoring algorithms increase in complexity, more memory must be used to store more object statistics. The first of these algorithms was presented in caching issues of hard disk drives [KLW94]. The Segmented LRU (SLRU) algorithm was developed because it was observed that in disk accesses, if a disk block was accessed twice, it would tend to be accessed many more times before being removed from the cache. So it divides the cache into two segments: a probationary segment and a protected segment, both of which are managed with the LRU algorithm, as it is depicted in Figure 2.5. When an object is first accessed, it is placed in the the probationary segment. If the object is subsequently accessed again while it is still in the cache, it is placed in the protected segment. One parameter must be set in this algorithm, and that is in what percentages the cache is split into the two segments. Karedla, Love, and Wherry found that the best performance came in the range of 60-80% of the cache being the protected segment. For the sake of generality, 70% was used in all of the simulations. So each time a new cache object is moved from the probationary segment to the protected segment, a maintenance check must be made on whether the protected segment is larger than 70%. If the protected segment has grown too large, the least recently accessed items are moved to the probationary segment, until the protected segment is once again less than 70% of the total cache size. To implement this algorithm in the cache replacement score paradigm, we must introduce a new variable, p, where $p_i = 0$ when object i occupies the probationary segment, while $p_i = 1$ when it is in the protected segment. So the CRS equation can be written as:

$$s_i^{SLRU} = p_i + 1/(t - l_i). \tag{2.15}$$

In [BCF+99], the Perfect-LFU is introduced. Perfect-LRU keeps track of the number of times that an object is accessed over the lifetime of the object, not only during the time that the object is in the cache. So even when a previously cached object has been evicted from the cache, its Perfect-LRU access count is maintained in the cache's metadata records. The perfect access count can be defined as A_i for object i, and the Perfect-LRU CRS equation becomes:

$$s_i^{PerfectLFU} = A_i. \tag{2.16}$$

This algorithm could be prohibitive in memory space allocation if the working set of objects is exceptionally large.

The next four algorithms use an inflation variable to store an addable scoring value for each cache object. This inflation variable is denoted as L, and there is only one instance of this value for the entire cache. For each of the algorithms, L is initialized to zero, and it is updated every time that a cache object is evicted. When this update occurs, L is set to the CRS of the evicted object.

The first of these algorithms that employ the inflation variable is called least-frequently used with dynamic aging (LFU-DA), introduced in [DAP99] and [ACD+99]. In LFU-DA, the inflation variable L is called the cache age factor, which is added to the cache access count when a cache object is entered into the cache or when it is reaccessed.

$$s_i^{LFU-DA} = a_i + L \tag{2.17}$$

This cache age factor was implemented to avoid setting a parameter as in the LFU-Aging algorithm [AFJ99b].

The other three algorithms that use L are in the GreedyDual-Size (GD-Size) algorithm family that was introduced in [CI97]. The GD-Size(1) and GD-Size(packets) algorithms were developed in [CI97] and were also studied in [DAP99] and [ACD+99], while the GD-Size(Hits) (sometimes called GDSF-Hits) is introduced in [DAP99] and [ACD+99]. The basic structure of each of these CRS equations is the same:

$$s_i^{GD-Size}(x_i) = \frac{x_i}{z_i} + L, \tag{2.18}$$

and the difference is generally the object statistics that are input with x_i. For GD-Size(1), $x_i = 1$ and the equation becomes:

$$s_i^{GD-Size(1)}(1) = \frac{1}{z_i} + L, \tag{2.19}$$

which is formulated to minimize misses. GD-Size(packets) is intended to minimize the number of network packets that are sent, so it sets $x_i = 2 + z_i/536$ to packet sizes:

$$s_i^{GD-Size(packets)}(2 + z_i/536) = \frac{2 + z_i/536}{z_i} + L. \tag{2.20}$$

The number of TCP/IP packets that an object occupies is two header packets plus $z_i/536$ body packets. A TCP/IP packet can carry a maximum payload of 536 bytes.

Finally, GD-Size(Hits) takes into account the popularity of the cached object by setting $x_i = a_i$. The GD-Size(Hits) CRS equation then becomes:

$$s_i^{GD-Size(Hits)}(a_i) = \frac{a_i}{z_i} + L. \tag{2.21}$$

On the highly complex end of cache replacement algorithms, Lorenzetti, Rizzo, and Vicisano developed a stochastic-based replacement algorithm that was inspired by trends that they observed in Web workload trace analyses from the University of Pisa [LRV97]. The algorithm is called Lowest Relative Value (LRV), and is more succinctly described in [CI97]. The LRV algorithm uses a combination of time since last access, number of accesses, and object size. It keeps track of the conditional probability of an $(a + 1)$th access given that an object has been accessed a times. For a given object i, P_i is this conditional probability and is calculated with the ratio A_{a+1}/A_a, where A_a is the number of objects requested at least a times in the trace. For $a_i = 1$, $P_1(z_i)$ is determined by the percentage of single access objects of the same relative size. That is, the size bins are $\lfloor log_2(size) \rfloor$ with a bin range of 2^{10} to 2^{20} (one kilobyte to one megabyte). Any objects that are smaller than one kilobyte are placed in the smallest bin, while objects larger than one megabyte are placed in the largest bin. With this background, two equations need to be defined, the first of which gets plugged into the subsequent CRS equation. $D(\cdot)$ is defined as:

$$D(t - l_i) = 0.035 \log\left((t - l_i) + 1\right) + 0.45 \left(1 - e^{\frac{-(t-l_i)}{2e6}}\right). \tag{2.22}$$

The coefficients for both additive terms of $D(t - l_i)$ are parameters that can be tuned to the workload that a given proxy cache is experiencing. However, for this study, these baseline coefficients were used. Now the CRS equation can be written as:

$$s_i^{LRV} = \begin{cases} P_1(z_i)\frac{1-D(t-l_i)}{z_i} & \text{if } i = 1 \\ P_{a_i}(z_i)\frac{1-D(t-l_i)}{z_i} & \text{otherwise} \end{cases} \tag{2.23}$$

2.7 Contributed Cache Replacement Score Algorithms

As part of this dissertation research, a number of replacement algorithms and system traces have been explored as will be discussed in the next several chapters.

From these analyses, several more cache replacement score algorithms have been developed. The first of these algorithms pairs up the LRU (Equation 2.2) and LFU (Equation 2.10) algorithms. The idea behind this combination is to calculate a frequency-biased least-recently used policy, thereby including an element of popularity in the calculation. So an object's score has a basis in the number of times it is accessed while it is in the cache, but the score also decays by the LRU portion of the equation while it is not accessed. For object i, it is calculated by

$$s_i^{LFLRU} = \frac{a_i}{t - l_i}. \tag{2.24}$$

Essentially, it produces a popularity-weighted LRU algorithm.

The second of these algorithms was inspired by the Least Relative Value (LRV) research [LRV97]. In the previous section, the LRV algorithm used a combination of time since last access, number of accesses, and object size but was shown to be quite complex. These three object statistics were observed to be significant in their web workload trace analyses from the University of Pisa. But by combining these in a multiplicative/reciprocal equation, each of these statistics would be represented in:

$$s_i^{SizeLFLRU} = \frac{a_i}{\log_2(z_i) \cdot (t - l_i)}. \tag{2.25}$$

This CRS equation combines the aforementioned LFLRU equation (2.24) with the small-favoring logarithmic size-based equation (2.7). In the same spirit of LFLRU, this algorithm calculates a popularity- and size-weighted LRU algorithm.

Using SizeLFLRU and LRD as inspiration, the third algorithm combines the LRD equation (2.24) with the small-favoring logarithmic size-based equation (2.7). This CRS equation is then written as:

$$s_i^{SizeLRD} = \frac{a_i}{\log_2(z_i) \cdot (t - e_i)}. \tag{2.26}$$

This algorithm formulation becomes a size-weighted access density for each object.

The last of these algorithms is a variation the LRU in Equation 2.2 which favors certain data types. In the Web workload traces that were used in this study and are described in Chapter 5, it was observed that image and html files composed the bulk of all objects requested. Generally over 90% of the accesses were images (GIFs, JPEGs, etc.) and html, and most of the cache hits were also of those file types. So the Priority LRU (PLRU) uses an additive variable, τ, which is set to $\tau_i = 0.001$ if object i is of html type and $\tau_i = 0.002$ if object i is of image type. Otherwise, $\tau_i = 0$. By setting τ_i to these values give the html and image types asymptotic advantages over objects of other types, that is, the other objects will be thrown out before the html and image types. Then the CRS equation becomes:

$$s_i^{PLRU} = \tau_i + \frac{1}{t - l_i}. \tag{2.27}$$

2.8 Idealistic Replacement Algorithms for Comparison

Before finishing this section, we need to define two more cache replacement algorithms. Both of these algorithms are idealistic algorithms that are used in determining how close to optimal cache replacement algorithms performed for a given data set. The first algorithm takes into account the caches size (the weight that the thief can carry), and it shows how low the miss rate and byte hit rate of a cache could be if foreknowledge of accesses could be used. The use of t as the current time is continued, and f_i is introduced as the time of the next future access of object i. Then Belady's optimal algorithm CRS becomes

$$s_i^{OPT} = \frac{1}{f_i - t}. \tag{2.28}$$

Thus, it removes the object from the cache that will be accessed as far into the future (if ever again) as possible. It must be noted that Belady intended this algorithm to be used only for caches in which objects are all of the same size. However, this algorithm provides an adequate, though not absolute, lower bound. Two papers discuss true variable size lower bound algorithms with the VMIN algorithm in [PF76] and the GOPT algorithm in [DS78]. But these algorithms would be very difficult, if not impossible, to implement in this cache replacement scoring framework.

The other idealistic algorithm assumes that the thief has infinite strength (or perhaps just herculean strength), and it assumes that the cache size is infinitely large and can accommodate all requests as hits once an object has been loaded into the cache. Therefore, it only records the misses due to compulsory misses which are the misses that all cache replacement algorithms must incur. There is actually no CRS equation for this idealistic algorithm, because no objects are ever removed from the cache.

2.9 The Questions

Having discussed this new paradigm for considering the cache replacement problem, we can discuss the questions that this research is addressing.

2.9.1 The Main Question

The first set of questions that this research asks are: By including more statistics of the cache object in the determination of cache removal, do the more complex cache replacement algorithms better predict the expected demand for objects in the cache? And which object statistics are involved in the algorithms that perform better, i.e., have a lower cache miss rate and/or cache byte hit rate? Is the additional computational complexity of using more cache object statistics worth the decrease in cache miss rates and byte miss rates?

2.9.2 The Second Question

The second question builds on the main question and asks: does an admission policy improve cache replacement policy performance? And if so, which algorithms benefit from using an admission policy? This asks whether it is more beneficial to include the currently requested object in the eviction process if there is no room to accommodate the currently requested object. Most cache replacement algorithms in use today do not use an admission policy and assume that the currently requested object is more valuable than some other objects already in the cache.

It must be noted up front that admission policies are effective only when a replacement policy is not dependent on time only. For instance, LRU will not be affected by an admission policy, because the object that was just requested is the most recently used object and will not be removed from the cache. But SLRU performs differently with an admission policy since it bases its decision on both time and access count. So a popular, older cache object may stay in the cache while a large, more recently accessed object may be denied admission before it was loaded into cache.

2.9.3 The Third Question

The third question pertains only to the video caching systems. As mentioned before, this study assumes that entire media objects are being cached. Other studies have used the proxy servers to buffer the media object stream on its way to the client. This buffering only stores a fraction of the object in the cache and is effective for serving only a few clients from the locally buffered partial object (such as in the Fellini project [OBRS95] and others).

So the third question asks: is it most efficient to cache whole media object files in the proxy server or would it be more effective to just buffer portions of the file as they are used?

To answer this question of efficiency, we must analyze how the media objects are individually accessed. If a media object is usually accessed in small portions, and the likelihood of the entire (or a large amount of the) media object be accessed in one session is low, then it would be rather inefficient to cache the whole media object in proxy servers. This is because it will take many small, piecewise accesses over the WAN to equal the bandwidth usage of moving a copy of the entire media object into a proxy server.

On the other hand, if a media object is usually accessed in such a way that most if not all of the object is viewed in each session, it is most efficient to cache the entire media object in the proxy server.

Further light can be brought to this issue by looking at an example. Say there is a 20 megabyte file that we want to serve on our multimedia network. In the small piecewise access pattern scenario, say that an average of one twentieth of the file (one megabyte) is accessed in a session. Hence it would take 20 of these average accesses to equal the bandwidth that is used to download the media object into a proxy server. However, in the whole-object access pattern scenario, each session will consume 20 megabytes of WAN bandwidth. So by making a copy of the media object in the proxy server, it will save large amounts of WAN bandwidth for further accesses.

In an analysis study of usage patterns of educational multimedia objects, [RM97] found that most education multimedia is accessed in a smaller piecewise manner. But these objects are usually accessed a large number of times within small time periods (temporal locality) at different places within the media object. This implies that the whole-object caching scheme can be useful, but its usefulness is not guaranteed. Other scenarios with a higher likelihood of whole-object accesses would guarantee a higher degree of usefulness. For instance, a pay-per-view movie-on-demand system would have a higher likelihood of accessing the entire video object within one session, because the viewers have paid to watch the entire movie and want their money's worth by watching it all in one session.

Before presenting the methods of experimentation in answering these questions, we need to explore what related works have contributed to answering these questions.

3. Related Work

In this related work chapter, the development of caches will be explored. Specifically, the focus will be on how cache replacement algorithms have developed in complexity as caches have been used in various applications. The discussion starts with caches being used in computer hardware memory hierarchies. It then moves to virtual memory page allocation/replacement algorithms, file migration allocation/replacement algorithms, disk system replacement algorithms, and database page access and replacement algorithms. Finally the discussion turns to distributed multimedia caches including World-Wide Web proxy caches and video-on-demand proxy caches.

3.1 Computer Hardware Memory Caches

The first uses for data (and instruction) caches were in computer hardware [Wil65]. These caches were implemented to speed up the realized memory accesses of the central processing unit; there was little need to be concerned with the bandwidth usage of the interconnecting buses in these systems, since they were not being shared with other memories, processors, or peripherals. Hardware caches are designed to minimize access latency and must be implemented entirely in hardware, so the caches have uniform cache block sizes and simple cache management logic. This simple management logic includes the cache replacement algorithm.

A part of the cache management logic is the implementation of associativity; i.e. the number of cache slots to which the block of a given memory location can be mapped. Hardware cache memories use restrictive associativities to keep complexity manageable thereby making it possible to implement the caches entirely in hardware. Early caches were direct mapped, meaning that the block of given memory location can only map into one location. For direct mapped caches, the cache replacement algorithm is trivial; the block of the newly accessed memory location replaces the block that was residing in the cache.

More recent implementations have overcome the complexity problems of developing two-way and four-way set associativity to reduce cache misses caused by cache thrashing. A few fully associated hardware caches have been implemented including Jouppi's victim cache [Jou90], which caches the 64 most recently evicted blocks from the main cache [CHK$^+$96]. With the cache set associativity schemes that are not direct mapped, the memory management system has a choice of which block should be removed from the cache. Since the cache replacement algorithm must be implemented entirely in hardware, the complexity of the cache replacement algorithm must be rather low. The three commonly used cache replacement algorithms are random (usually pseudo-random), first-in first-out (FIFO), and least-recently used (LRU) [Smi82]. Cache replacement algorithms that exhibit higher complexity than these three generally cannot be implemented efficiently in hardware, and are then partially implemented in software. Most added complexity has been used to maintain data coherence in multiprocessor, superscalar, and speculative execution environments [Ste90].

3.2 Operating System Virtual Memory Direct Paging Algorithms

A few years after cache memories were implemented in computers, operating systems began implementing virtual memory direct paging algorithms. Virtual memory allows an operating system to act like it has more RAM memory than the amount of physical RAM memory that it has. Furthermore, virtual memory direct paging allows an operating system to only have a portion of a process's instructions and data occupy the computer's physical memory. But with only a part of each process in the physical memory, different memory pages need to be swapped into RAM memory when the pages' contents are needed, while other pages need to be swapped out when they are no longer needed. Page replacement policies are used to determine which page will be removed to accommodate another incoming page.

When the contents of a virtual memory page is not found in physical memory, it is called a page fault, and the faulted page must be copied from the virtual memory disk partition to a page location within the physical memory. Though page faults occur infrequently, they are very costly to the performance of the processor. Also, there are many direct pages in the physical computer memory. For example, if each page is four kilobytes, and the physical memory of the computer is 64 megabytes, then there is capacity for 16k pages to fit into physical memory. For these two reasons, the page replacement algorithm should not be too complex.

But the page replacement algorithm should also be as accurate as possible. Silberschatz and Galvin [SG98] explain that the FIFO algorithm is not especially accurate, and they go on to explain that Belady's OPT algorithm would be the best algorithm to use for direct page replacement. [SG98] then argues that LRU approximates OPT best for direct page replacement, but LRU is too complex to compute for thousands of pages for each page fault. Therefore, they present several LRU approximation algorithms, including the additional-reference-bits algorithm, the second-chance algorithm, and the enhanced second-chance algorithm. Since each of these algorithms approximate the LRU algorithm, we can expect that generally they will not perform better than LRU. But for the demand paging application, the increase in page-faults is not significant enough to offset the decrease in page-fault handling time that is realized by implementing the less complex LRU approximation algorithms. Because of the considerations for the number of pages in physical memory and maintaining computational simplicity in the page replacement algorithm, only FIFO, LRU, and approximation algorithms based on FIFO and LRU are used for virtual memory direct paging algorithms.

3.3 File Migration Algorithms

Similar to the virtual memory direct paging application of replacement algorithms is file migration. In the earlier days of computing, user processes were stored on large magnetic tapes. Only the processes that were expected to be run in a given day were temporarily stored on the computer's hard disk drives. When the user needed a process file (software code and data) that was being stored on the magnetic tape, the process would be migrated from the tape onto the computer hard disk drive. So file migration deals with higher levels of the memory hierarchy.

The file migration application of replacement algorithms have several differences from the hardware and direct paging cache algorithms discussed before. The file migration process files are of variable sizes; the hardware caches have a fixed block size, and the direct paging pages also are of a fixed size. Furthermore, the [Smi81] and [LRB82] studies assume that the cache itself has a variable size which may not be that realistic. Every computer system has a fixed (though sometimes huge) disk drive capacity. Also, the hardware and direct paging algorithms run on-demand. The file migration algorithms that were studied were run once a day in the middle of the night. At that time, the algorithm removed the process files that it predicted would not to be used in the next day [Smi81, LRB82]. Since the computer disk drive capacity was fixed, then every night when the algorithm ran, only a fraction of the drive capacity was freed for process files that were new accesses. If more drive capacity was needed than was freed the night before, these algorithms ran into trouble. Smith, though, does acknowledge that the file migration algorithm could have been modified to accommodate the fixed capacity issue for on-demand use [Smi81].

The study of Smith [Smi81] introduces a stochastically optimal (Stochopt) algorithm which uses the entire reference history for each of the process files in the system. It records whether each process was accessed in a day, and then it predicts whether the process will be accessed in the next day. Smith compares this Stochopt algorithm to VMIN [PF76], GOPT [DS78], Working Set (WS) [Den68], Space-Time Working Set (STWS), and several stochastic expected-(mean)-time-to-next-reference algorithms. Smith found that his Stochopt algorithm outperformed all of the other non-optimal algorithms and performed nearly as well as the optimal VMIN and GOPT algorithms. Smith does admits, however, that converting his Stochopt algorithm for on-demand usage would make for a rather computationally complex algorithm.

The Lawrie, Randal, and Barton study [LRB82] substantiates some of the same comparisons that Smith did in [Smi81]. They compared STWS, GOPT, Size, LRU, and several Size and LRU combinations. They found that STWS performed best among the non-optimal algorithms. Note that they did not include Stochopt in their study. [LRB82] also explored clustered prefetching in which not only the requested process files are migrated, but also other similar processes. They found this approach to be rather ineffective to marginally effective.

Because of faster networks, greater use of file servers, and dramatically increasing hard disk drive capacities, file migration is no longer an issue on most of today's computer systems.

3.4 Disk System Caching Algorithms

Caches are used in I/O subsystems of computers to alleviate the bottlenecks caused by the mechanical latencies of disk and tape drives. One paper written by Karedla, Love, and Wherry in 1994 [KLW94], examined the performance of cache replacement policies used in these I/O subsystem caches. They investigated the caches used specifically in disk drive caches. The paper introduced a frequency-based variation of LRU which they called segmented LRU (SLRU). The SLRU algorithm is described in Section 2.6 including Equation 2.15. The three replacement algorithms were simulated with four different customer workload traces comprised of an inventory control workload, a batch processing workload, a scientific time sharing workload, and an airline reservation workload. The study found that SLRU algorithm performed the best on all of the workloads for a wide variety of cache sizes.

3.5 Relational Database Buffer Management Algorithms

Relational database buffer management refers to the temporary storage of a subset of a relational database records in the computer's main memory. Database records are usually only processed from the computer's main memory. Two fundamental differences set relational database buffer management apart from other cache management applications. First, the buffer size for a given database process is usually variable, though the size of the overall memory is fixed. Second, database management algorithms generally have more control over the order in which database records are accessed and processed. This processing order control means more optimization is possible when buffering those records from the disk drives into main memory for processing.

A study by Effelsberg and Haerder compared several conventional buffer replacement algorithms on database reference strings [EH84]. They compared Random, FIFO, LFU, LRU, Second Chance, Generalized Clock, and LRD. They found LRU and Second Chance to perform satisfactorily, though LRU with locked records sometimes performed worse than their Random algorithm. They also found LRD and Generalized Clock performed well.

Most studies of relational database buffer management algorithm use allocation principles to determine which records should be held in main memory. For instance, Denning's Working Set algorithm is a cache space allocation algorithm which works much like LRU [Den68, CD85], while Sacco and Schkolnick's Hot Set algorithm analyzes the loops of record accesses and implements an allocation algorithm similar to LFU [SS82]. Other more complex algorithms try to further exploit on optimize on the consistent database record access patterns. But since humans are much more random in selecting media objects, these algorithms are not of much use for multimedia proxy caches.

3.6 World-Wide Web Proxy Caches

World-Wide Web proxy caches are a fairly recent addition to the list of applications that require cache replacement algorithms. As it was explained in Section 2.1, these proxy caches can be situated throughout the Internet. Using standardized protocols, the caches can communicate with each other to retrieve the requested objects from the closest source, whether that is the origin server or another cache that is currently storing the object [MLB95]. However, the concept of of WWW proxy caches came from simple beginnings.

In reading about these studies, take special note of how few cache replacement algorithms are compared in each study and how few input data traces are used in each study. Also note the types of traces that were used in each study. Chapter 5 presents a classification basis of Web proxy cache input workloads, and each of the input workloads that are mentioned in this section are named with their domain classification. Simply put, each classification has its own temporal locality, spatial locality, size distribution, and access pattern characteristics.

3.6.1 The Early Papers

In the first paper that proposed hierarchical proxy caches on the World-Wide Web, the cache replacement algorithm that was implemented for the simulations was LRU [MH92], but no comparisons were made to other possible cache replacement algorithms. In another study that examined FTP traffic on NFSnet, Danzig, Hall, and Schwartz compared the LRU and the LFU policies on simulated network proxy servers. They found that the LRU and LFU policies performed similarly on FTP traffic, because if a file was downloaded more than once, it was usually within a few hours of the first download [DHS93].

Pitkow and Recker introduced the first cache replacement algorithm specifically for a World-Wide Web proxy cache. Their algorithm is based on user access pattern analysis of Georgia Tech WWW access traces and psychological research on human memory retrieval. It implements a least recently used strategy that is only run once a day [PR94]. The algorithm uses a seven day access record window to predict whether the cached object will be used the next day; ties were broken by removing the larger cache object first. No simulations were run as a part of this study, but the authors conjectured that this algorithm would be beneficial for use on large proxy cache servers. But since the algorithm only ran once a day instead of running on-demand, a significant number of cache objects may be removed from the cache which may be requested before the object actually needed to be removed from the cache as an on-demand system would.

Abrams, Standridge, Abdulla, Williams, and Fox explored how cache replacement algorithms performed when the size of the object was taken into account. They introduced two algorithms, LRU-Min and LRU-THOLD [ASA$^+$95]. As explained in Section 2.6, to make space for an incoming cache object, LRU-Min tries to remove all items larger than the incoming object in LRU order, then tries to remove items larger than 1/2 the incoming object's size, then 1/4, etc. The LRU-THOLD algorithm doesn't cache any items that are larger than a certain threshold; they found that it was quite difficult to tune this threshold parameter. Their simulations included HTTP, Gopher, FTP, and WAIS documents from three different types of educational workload traces on the Virginia Tech campus (educational client proxy traces), and LRU was used for comparison. With miss rate as the comparison measure, they found that LRU never performed best, and that LRU-Min and LRU-THOLD split being the best in the simulations. They concluded that LRU-Min was the best policy among the three (and it required no parameter adjustments), while LRU-THOLD was best when used on small cache sizes.

The majority of an article by Bolot and Hoschka involves analyzing and predicting World-Wide Web user access patterns using seasonal ARIMA (autoregressive integrated moving average) trends [BH96]. Near the end of the article, though, they use those analysis results to propose a proxy cache replacement algorithm with calculated weightings based on four cache object statistics: time since last reference, size of object, time to retrieve, and time-to-live. In simulations using an INRIA Web server trace data (origin server proxy trace), they compared this algorithm to LRU and showed that it performed slightly better on miss rate and much better on a weighted miss rate that measured perceived retrieval time.

In Williams, Abrams, Standridge, Abdulla, and Fox's article from 1996, they looked at cache replacement algorithms as a problem in sorting the cache objects by calculated keys to determine the order of removal [WAS$^+$96]. They defined three factors, the primary sort algorithm, the secondary sort algorithm, and the workload (data set), and they measured both the hit rate and the weighted hit rate. Six algorithms were considered for sorting: SIZE, $\log_2(\mathit{SIZE})$, ETIME (FIFO), ATIME (LRU), DAY(ATIME) (Pitkow and Recker's daily LRU), and NREF (LFU). These simulations used the same three different types of educational workload traces from the Virginia Tech campus (educational client proxy traces) as [ASA$^+$95], which included HTTP, Gopher, FTP, and WAIS documents. The replacement algorithms were only run on-demand because they saw no advantage in running the algorithms periodically as was done in Pitkow and Recker's study [PR94]. Instead of using an optimal algorithm against which to compare, they established maximum hit rate by using an infinitely sized cache; that is, they compared each algorithm to an ideal cache that only incurred compulsory misses. They found that when hit rate was the measure, SIZE and $\log_2(SIZE)$ as primary sort algorithms always did best. This is because SIZE and $\log_2(SIZE)$ gave preference to smaller files, and the cache could hold many more small files than large ones thereby raising the hit rate. When using the weighted hit rate as the measure, there was no clear best performing algorithm, but ATIME (LRU) performed consistently well, while SIZE was consistently the worst performer. Finally, the study found that the secondary sort algorithm had little impact on the performance of the cache. This is probably because they used the secondary sorting algorithm to arbitrate between ties from the primary sorting algorithm instead of giving it more significant influence on the primary sorting. By using multiple simple cache replacement score formulas to produce a more complex primary cache replacement algorithm, the algorithms of Section 2.6 and 2.7 use more cache object data than just the primary sort criteria leading to a more informed decision.

3.6.2 Beyond the Basics

Reddy, in his 1997 study, also found that it was advantageous to take the size of the cached object into account [Red97]. Reddy introduced two modified most-recently-used (MRU) algorithms (AVI-MRU and Size-MRU), introduced the SPACExAGE from file migration to the WWW proxy caching research area, and compared them to LRU. AVI-MRU gives audio and video files a lower priority than all other files so the audio and video files are removed before any other file types are even considered. The Size-MRU operates by having files larger than a 32 kilobytes threshold managed by MRU policy, while those less than 32 kilobytes are handled with LRU. The motivation behind the Size-MRU algorithm is that large files are not supposed to displace too many smaller files. Reddy references two file migration papers [Smi81, LRB82] as the inspiration for his SPACExAGE algorithm. His SPACExAGE algorithm implements four LRU-managed chains which hold objects that are under four kilobytes, between four kilobytes and 32 kilobytes, between 32 kilobytes and 64 kilobytes, and over 64 kilobytes. When a cache object needs to be removed, the $space \times age$ (size $\cdot$ LRU) product is calculated and the object with the largest product is removed. Reddy's study used NCSA main server traces [KMR95] (origin server proxy trace), and he measured the request response times and relative miss rates. He did not, however, measure byte miss rate. Reddy found that AVI-MRU performed slightly better than LRU and that SIZE-MRU was very sensitive to the threshold value. Generally, he found that SPACExAGE outperformed the other algorithms, especially on video files. He also found that the advantage of SPACExAGE over LRU decreased as the cache capacity was increased.

Breslau, Cao, Fan, Phillips, and Shenker published a study that extensively analyzed six different traces which included three client proxy traces (a Digital Equipment Corp. web proxy trace from 17,000 workstations, a Questnet Australian regional ISP trace, and a FuNet Finnish regional ISP trace), two educational client proxy traces (a user trace from the Computer Science department at the University di Pisa and a Home IP service trace from UC Berkeley), and one network proxy trace (a one-day trace from National Lab for Applied Networking Research – NLANR) [BCF$^+$99]. Unfortunately, the proxy server from which the NLANR trace was taken was not shared in the paper. (Please refer to Section 5.4.5 for an explanation of the NLANR proxy caching facilities.) The majority of the paper discusses various characteristics of the access traces, but the last section is most applicable to this chapter. In that last section, a comparison of Perfect-LFU (Equation 2.16), In-Cache-LFU (Equation 2.10), LRU (Equation 2.2), and GD-Size (Equation 2.19) is made using the six aforementioned traces as input. They found that In-Cache-LFU performed worst on all of the input traces and was a poor choice for cache replacement algorithms. GD-Size performed best in terms of hit rates for small caches, while Perfect-LFU generally performed best in terms of byte hit rate. They conceded that Perfect-LFU needed to use more memory space to maintain request counts for all of the objects in a trace, and it didn't take document size into account. They also explained that LRU usually performs best when temporal locality effects are strong. However, they concluded that since Perfect-LFU generally outperformed LRU, these traces may not have as much of a temporal locality effect as they had expected.

Lorenzetti, Rizzo, and Vicisano set out to design an algorithm based on statistical parameters and lifetimes of cached objects with the intent to beat LRU [LRV97]. They analyzed a trace from their own department, the Department of Information Engineering at the University of Pisa. The algorithm that they developed is described in Section 2.6. They developed Equation 2.22 as an interaccess distribution approximation, but a shortfall of this equation is the two coefficient parameters that could be tuned to the workload characteristics of the proxy server. They went on to observe that the number of previous accesses is a good indicator of the probability of further accesses, and that smaller files are accessed more often which they attributed to slower modem connections. Furthermore, they found that the document type didn't influence access requests much since the vast majority of accesses were text or images. These observations led them to formulate Equation 2.23 which uses LRU, access count, and document size to embody the cache replacement algorithm. They compared LRV with FIFO, LRU, Size, and Random using the University of Pisa trace (educational client trace) as input. They found that LRV consistently outperformed the other algorithms when byte hit rate was the measure, and that LRV generally outperformed the others when hit rate was the measure. With hit rate as measure, Size outperformed LRV on larger cache sizes. These findings prompted them to conclude that LRV proved to be particularly useful in small caches. This algorithm, though it may be effective, may be difficult to implement (especially in small caches) since it does need to collect more statistics than the other cache replacement algorithms in this dissertation, and requires a great deal more computation including a logarithm and exponential calculation for each cache object.

As was discussed in Section 2.6, three algorithms based on the GreedyDual-Size

(GD-Size) algorithm family were introduced in [CI97, DAP99, ACD$^+$99]. The GD-Size(1) and GD-Size(packets) algorithms were developed in [CI97] and were also studied in [DAP99, ACD$^+$99], while the GD-Size(Hits) (also called GDSF-Hits) was introduced in [DAP99, ACD$^+$99]. The GreedyDual-Size algorithm family was inspired by the GreedyDual algorithm [You94] which handled uniform-size variable-cost cache replacement [CI97]. All three algorithms are presented in Section 2.6. Cao and Irani developed the GD-Size(1) algorithm to minimize the cache miss rate while they developed GD-Size(packets) to minimize network traffic (byte miss rate). Cao and Irani proved that GreedyDual-Size is online optimal stating that it is "k-competitive, where k is the ratio of the size of the cache to the size of the smallest document". They then compared these two algorithms to LRU, Size, Hybrid, and LRV using three client proxy traces from Digital Equipment Corporation (a client proxy trace), Virginia Tech, and Boston University (two educational client proxy traces). (Hybrid was introduced in [WA97] which is described later.) They found that GD-Size(1) achieved the best hit rate across all simulations, but it didn't perform as well when byte hit rate was the measure. GD-Size(packets) usually achieved the highest byte hit rate and second best hit rate overall. The study also attempted to explore three other metrics, latency reduction, hop reduction, and weighted-hop reduction, and they tried to tailor GD-Size algorithms for those metrics. This type of information usually isn't included in the trace files, so they attempted to estimate download latency and network hops from the traces. The study found that GD-Size(1) was the best algorithm to reduce average latency, and that algorithms that took network costs into account performed no better than algorithms that ignored network costs.

Arlitt, Dilley, and others at Hewlett-Packard Labs introduced two algorithms, GreedyDual-Size with Frequency (GD-Size(Hits)) (Equation 2.21) and Least Frequently Used with Dynamic Aging (LFU-DA) (Equation 2.17), which were refinements on the above GD-Size(1) and GD-Size(packets) algorithms. These algorithms were introduced in [DAP99, ACD$^+$99]. The HP Labs Technical Report [DAP99] implemented LRU, LFU-DA, and GD-Size(Hits) into a Squid cache and used SpecWEB as the input workload. They found that LRU outperformed GD-Size(Hits) when there were many large documents in the trace workload. But LFU-DA outperformed LRU in byte hit rate for all cases. More significantly, they found that CPU demand on the Squid proxy cache machine actually decreased even though the cache replacement algorithms were more complex. They attributed this decrease in CPU demand to the reduced miss rates that were effected by the more complex algorithms. This can easily be justified when one figures that the usual bottleneck of the proxy cache server is the I/O subsystem. Since the CPU is usually not the bottleneck, it is worthwhile to invest some extra cycles in the replacement algorithm to eliminate some more disk and/or network bandwidth usage. The [ACD$^+$99] study compared LRU, LFU-Aging (Equation 2.10), GD-Size(1), GD-Size(packets), GD-Size(Hits), and LFU-DA. The input workload trace was a cable modem ISP proxy trace (large client proxy server) from the San Francisco Bay area [AFJ99a]. They found that the GD-Size(Hits) algorithm performed best in terms of both hit rate and byte hit rate. LFU-DA always performed better than LFU-Aging, and LFU-DA can be coded to perform fewer calculations than LFU-Aging. Furthermore, in byte hit rate, LFU-DA was second only to GD-Size(Hits), but LFU-DA didn't perform as well in hit rate. Since this study was with a cable modem ISP trace instead of a plain telephone modem ISP trace, these results are significant for future consumer client ISP proxy server implementations.

In Tewari, Vin, Dan, and Sitaram's study [TVDS98], the concern was with developing a caching algorithm that could guarantee the performance of the server to the clients. They considered the server's bandwidth and cache storage capacity as the two factors for a two-constraint 0-1 knapsack problem. They further split the two-constraint knapsack problem into two one-constraint problems by having separate "goodness" calculations depending on whether the server was currently space-constrained or bandwidth-constrained. If the server was space-constrained, the "goodness" factor was calculated as a mean time-to-reference (averaged inter-reference time) with the highest scoring object being removed. On the other hand, if the server was bandwidth-constrained, the "goodness" factor was calculated as the bandwidth consumption of the object, and again the highest scoring object is removed. In their simulations, they used traces from NCSA [KMR95] (origin server proxy trace) and NLANR [AW96] servers (network server proxy traces) as well as stochastic Zipf access distributions [Zip49]. They ran simulations of just continuous media (CM) objects and combined CM and non-CM objects and found that their algorithm performed best for both hit rate and byte hit rate. These results should be tempered because their algorithms best addressed the issues that their measures were evaluating.

3.6.3 Other Related Web Caching Papers

Three papers argued that using the estimated page load delay would provide the best replacement statistic. [WA97] introduced two algorithms: the Latency Estimation Algorithm (LAT), which attempts to estimate the object access latency (ttr) on a source server basis of each cached object and replace the object with the shortest download time, and Hybrid, which considers object access latency, number of references, and document size in the replacement decision. They modified a Harvest cache and used workload traces from Virginia Tech and Boston University (educational client proxy traces). They were able to run online and replay modes on the Harvest cache, and they used live network latency values. They found that LAT performed worse than both LRU and Size for average download time, hit rate, and weighted hit rate, while Hybrid usually performed best for average download time and hit rate. In [SSV97] and [SSV99], Scheuermann, Shim, and Vingralek introduce two other algorithms, LNC-R-W3 and LNC-R-W3-U. Least Normalized Cost Replacement for the World Wide Web (LNC-R-W3) takes the average rate of object reference, object size, and delay-to-fetch (ttr) into account for the replacement decision. In a comparison study with the LRU and LRU-Min algorithms using a Northwestern University client proxy cache trace, they found that LNC-R-W3 consistently outperformed LRU and LRU-Min [SSV97]. Least Normalized Cost Replacement for the World Wide Web with Updates (LNC-R-W3-U) includes distributed cache consistency into the replacement decision by including time-to-live with time-to-retrieve. They showed that LNC-R-W3-U performed better than LNC-R-W3, LRU and LRU-Min for delay savings ratio, and performed comparably to LNC-R-W3 for hit rate. Also they showed that integrating time-to-live with other object statistics improved cache staleness over LRU and LRU-Min. With all three of these studies, determining the time-to-retrieve and time-to-live were difficult to measure and gain accuracy. Also, ttl and ttr are not recorded in any proxy trace files. Therefore, the two statistics are not used in the simulation portion of this dissertation study. However, the simulator has been written to incorporate these statistics if they become available in trace files.

Two papers address the effectiveness of using an admission policy in proxy caches and helped prompt the inclusion of studying admission policy usage in this dissertation. Aggarwal, Wolf, and Yu developed a algorithm which is very similar to SPACExAGE (Equation 2.13) called Pyramid Selection Scheme (PSS) which includes an efficient object metadata implementation. They include an admission policy in their algorithm so that no requested object that is to be placed in the cache is displacing another object in the cache that is more likely to be accessed in the cache. They compared their PSS algorithm to SPACExAGE (which they call Size-adjusted LRU), and in all of their simulations, PSS performed better than Size-adjusted LRU. Unfortunately, they did not discuss the impact that their admission policy had on the performance of their PSS algorithm. In [KSW98], Kurcewicz, Sylwestrzak, and Wierzbicki argue that caching every object that clients on a network request unnecessarily tax the disk subsystem which is the main bottleneck of busy proxy caches. Their algorithm determines whether an origin server has been accessed by more than one unique client; their rationale is that if an origin server is "shared", its objects are more likely to be accessed again. Their algorithm has a parameter T that sets the time in minutes in which two clients must access an origin server's objects to be considered a "shared" server. Using two educational client traces from Warsaw MAN and University of Cracow, they found that this admission filter lowered disk activity, but it also lowered the hit rate. They conceded that it was a tradeoff that may not be effective for everyone.

Several other papers deserve mentioning though they are not as important to this dissertation as the aforementioned studies. Markatos found that using main memory (along with disk caches) improves the performance in origin servers for web documents [Mar96]. Cao, Zhang, and Beach proposed the Active Cache system which provides proxy caches the ability to cache dynamic content [CZB98]. The Active Cache temporarily stores a “cache applet” along with the data object. The first two papers that address cache consistency across distributed cache proxy servers are [MLB95] and [GS96b]. Finally, Gwertzman and Seltzer published two papers on geographical push-caching which is an intelligent method for prefetch objects that are related to requested objects [GS95, GS96a].

Now that we have a background on how cache replacement algorithm research has developed in a number of disciplines, we can define the simulation model and data for this study.

4. The Simulation Environment

To answer the questions of this study, computer simulation has been used to compare different cache replacement policies.

The simulator is split into two parts, the data generators and the cache simulator. This allows the simulator input to come from a variety of sources, and it allows different simulation runs with different cache scenarios to use the same input data. In presenting the cache model, the cache simulation model and the cache simulator functionality will be explained first. Then the construction of the simulation runs will be explained, as well as how the results will be presented. Finally in Chapter 5, the set of simulation scenarios that were used in the simulations are explained.

4.1 Simulation Model

The architecture of server system model is hierarchical in nature. Level (i) servers can receive media object from its corresponding $(i-1)$ level servers or from any other (i) level server.

For this simulation, we are modeling only one system which is situated somewhere in the network. Since the simulator assumes the role of this single autonomous cache proxy server, there are no assumptions concerning from what source requested objects are received. One can safely assume that the cache proxy server receive media objects from the central server or from other network proxy servers by using a greedy-forwarding strategy [LYW91] in which the media objects are forwarded from the most convenient (least costly) source. Thus, the simulator is able to mimic the cache behavior of any of the proxy server configurations shown in Figure 2.2. By assuming this hierarchical structure for the network, it allows us to assume a deterministic transmission cost linearly related to the size of the media object. Each server is able to make decisions independently; there is no central scheduling system that makes decisions for the entire network. And the system needs no *a priori* knowledge as the Rangan patented systems [Ran96, PR97] do, because it is making greedy decisions based on the information it has from the past accesses and the current scenario. Within the multimedia distribution system there are M media objects, which are identified as m_1, m_2, $\ldots$, m_M. Each of these objects, m_i, has a object size, ms_i, associated with it. The object size is a positive integer. For simulation flexibility, the object size does not have units associated with it; for the multimedia objects, the object size is the length of the video in minutes (rounded up) or the size of the object in kilobytes, while the object size is the number of bytes in the web proxy server simulations. To make sure that the cache is the part of the system that is being tested, the networks are expected to be able to accommodate the bandwidth necessary to deliver the media objects without degradation in quality of service (using streaming technology for the video objects).

With this network scenario, the network proxy cache server is described as such. The cache has a maximum size denoted $Cmax$ as well as a cache replacement algorithm associated with it. It also maintains two variables to keep track of its usage: the number of size units currently occupying the cache, Cs, and the number of objects currently occupying space in the cache, Cn. Both Cs and Cn are non-negative integers, and $Cmax$ and Cs do not have units associated with them. Each media object that is currently occupying the cache has data associated with it. For each of the media objects in the cache, Cm_i, $1 \leq i \leq Cn$, the data structure contains the object number, Cmn_i; the object size, Cms_i; and the objects current score, Cmc_i. On the network proxy server, the storage maintenance time requirements for disk defragmentation and other tasks are disregarded. The system should be able to do such maintenance during idle or low-demand times.

Now that the system model has been defined, the simulation parameters can be defined. For each simulation, the total number of iterations, N, is defined as the total number of media objects that are requested. Furthermore, a maximum object size, $Msmax$, is defined as the maximum size of any of the M objects, $ms_i \leq Msmax$, for

$1 \leq i \leq M$. Also, a minimum object size, *Msmin*, is defined as the minimum size of any of the M objects, $ms_i \geq Msmin$, for $1 \leq i \leq M$. These variables come in handy when defining the data sets used in the simulations, which are presented in Section 4.3 and Chapter 5.

4.2 Simulator Functionality

The simulation software includes the configuration loading module, the simulation file loading module, the printing module, and the simulation core including the cache replacement algorithms. The configuration loading module loads in the configuration file for the given simulation scenario. The simulation file loading module loads the data file for the simulation into the simulator. The printing module formats and prints the output files that are used to generate tables and graphs of the simulation results. Finally, the simulation core actually implements the cache replacement algorithm portion of the proxy cache server which is depicted as a flowchart in Figure 4.1.

Each iteration of the simulator involves loading the next requested object into the cache. If the summation of the sizes of all of the objects in the cache exceeds the size of the cache ($Cs > Cmax$), then objects must be evicted from the cache. The cache replacement scores are then calculated for each of the objects currently in the cache. The lowest scoring objects are evicted until the summation of the sizes of all of the objects in the cache no longer exceeds the size of the cache ($Cs \leq Cmax$). If the summation of the sizes of all of the objects in the cache does not exceed the size of the cache ($Cs \leq Cmax$), then the next object is loaded into the cache. For this simulator, only the metadata is loaded and maintained in the software; the actual object data is not loaded into the system.

When an admission policy is not being used, the newly requested object is not included in the calculation of the cache object scores nor is the newly requested object eligible for eviction for having the lowest cache replacement score. Also, when a video object is still being used by a client, the video object is not eligible for eviction. This stems from the assumption that once a video is being streamed from the proxy cache server, it will continue to be served from the proxy cache server until the client is

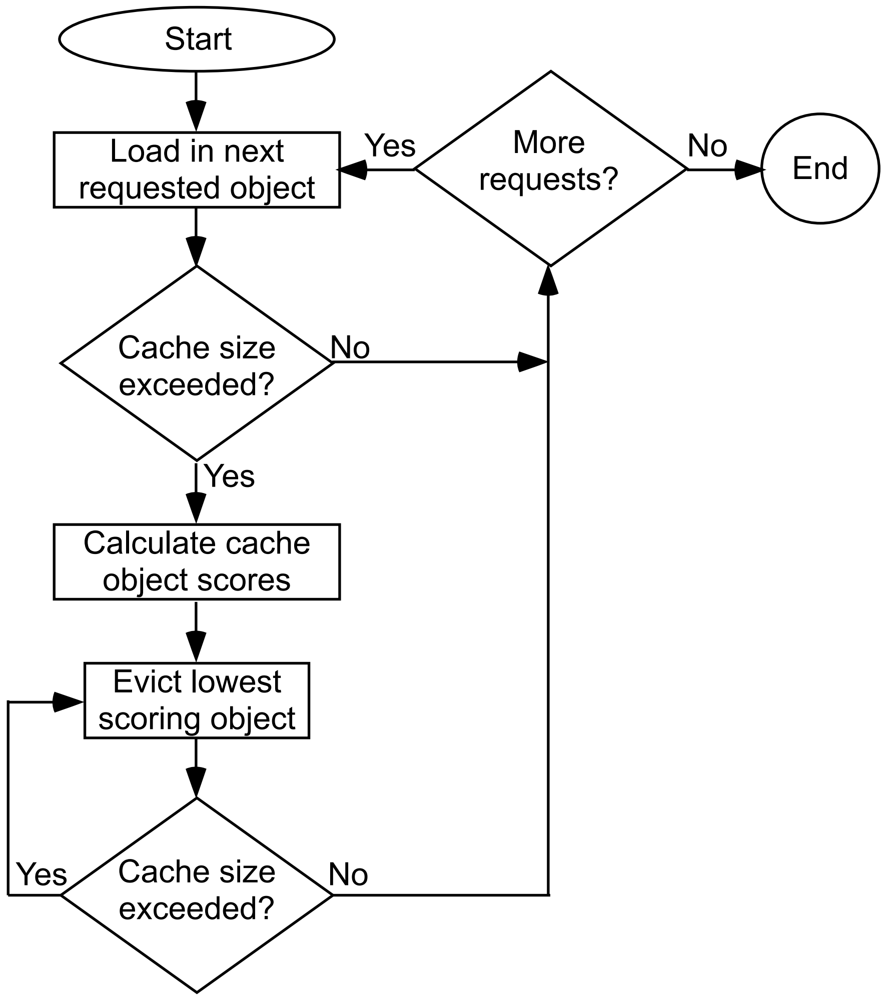

Start
Load in next requested object
Yes
More requests?
No
End
Cache size exceeded?
No
Yes
Calculate cache object scores
Evict lowest scoring object
Yes
Cache size exceeded?
No

Fig. 4.1. Basic Simulator Functionality Flowchart.

done viewing the video object. On the other hand, when an admission policy is being used, the newly requested object is included in the calculation of the cache object scores and the newly requested object is eligible for eviction for having the lowest cache replacement score. And, the fact that a video object is still being used by a client is disregarded. In this case, it is assumed that when video streaming service is interrupted by evicting the video object from the cache proxy server, the service will be seamlessly resumed by the origin server. This may be somewhat unrealistic, but it was the only way that the simulation would work for the video proxy server workload traces. Therefore, the simulation results for the video proxy server traces with admission policies may not be completely accurate.

There are a number of assumptions that are made in implementing the simulator that need to be discussed. These assumptions are made to force the simulation to focus only on the cache replacement algorithms. First, it is assumed that the proxy server being simulated has plenty of memory and I/O bandwidth to serve all of the requested objects. Furthermore, it is assumed that the video proxy server being simulated has enough memory and I/O bandwidth to stream and server all of the videos that clients have requested. Also, all of the disk access latencies and system latencies are neglected in these simulations. All of the objects in the system are assumed to be read-only; if the sizes of two identically named objects are different, the newer one is assumed to be a modification of the older one and the two objects are treated as two unique objects. No warmup periods have been used for any of the simulation scenarios. Since there are quite a number of workload traces studied in this dissertation, it is very difficult to determine equivalent warmup periods that allow a fair comparison of cache performance for each of the workload traces. Finally, only on-demand cache replacement is implemented. Periodic cache replacement is not explored because simulating periodic cache replacement can only increase the miss rates of the cache replacement algorithms under simulation.

4.3 Presentation of Results

In order to present the results of the simulations, we must first define how the simulation scenarios will be constructed.

4.3.1 The Measures

Two measures will be used to compare the cache replacement algorithms on the different simulation data traces. The first measure is miss rate (MR) which relates to cache miss latency. MR is defined as the fraction of requests that were not serviced by the cache. The second measure is byte miss rate (BMR) which relates to network bandwidth usage. BMR uses the actual sizes of the objects and conveys how many size units were not serviced by the cache divided by the total number of size units that were requested by the clients. (As explained in Section 4.1, object object size is expressed in size units which could be bytes, kilobytes, or minutes of video.) Both the MR and BMR will be reported as a fraction between zero and one or a miss percentage. These two measures are intended to answer the Main Question of Section 2.9.1.

To answer the Third Question of Section 2.9.3, another measure must be taken. Some average media object viewing percentages for each session could be set, i.e., one percentage value that is applied to all of the unit sizes of the media object requests. The sum of all of the fractional unit sizes would constitute the total network bandwidth usage if no caching were used. For example, if three objects, of unit sizes 80, 100, and 120, are accessed once each with an average media object viewing percentage of 25%, the sum would be $20 + 25 + 30 = 75$ size units.

These average media object viewing percentages could then also be compared for the caching and non-caching scenarios. Instead of expressing the weighted miss rates as percentages or fractions of one, raw size units need to be compared; that is to say, the actual network bandwidth usage values need to be compared. This will constitute the third measure. And since this third question only applies to the video delivery systems, it will only be used for those traces.

4.3.2 Simulation Parameters

The simulation parameters are system dependent because of differences in the workloads that they experience.

The maximum sizes of the cache, $Cmax$, will be determined as a percentage of the total unit size of the simulation data set for the video system workload traces. For the DVJ2 data files, the total unit size of all of the objects will be calculated. For the stochastic data files, the mean unit size of the data set is calculated as the product of the average object size and the number of objects in the data set. And for the OnCommand data files, all of the videos are assumed to be the same size so the cache sizes can be determined as how many videos should be able to occupy the cache simultaneously. Multiple cache sizes will be simulated for each data set and some preliminary percentages of the data set size are 5%, 10%, 15%, 20%, 25%, and

30%.

Finally, for the Web cache traces, one cache size set is defined. For all of the Web workload sets, the cache sizes are 1 megabytes, 4 megabytes, 16 megabytes, 64 megabytes, and 256 megabytes (1,048,576; 4,194,304; 16,777,216; 67,108,864; and 268,435,456 bytes). These sizes are smaller than current commercial implementations, but they are the right sizes for the sizes of the workload traces, and the results will scale up to larger cache sizes.

Furthermore, for each of the stochastic data set scenarios, 50 different data sets will be generated to adequately eliminate abnormalities in the randomly generated data. The object sizes will be uniformly distributed between 500 megabytes and 1.000 gigabytes, which models the object sizes of the DVJ2 system. Each data set will have 10,000 requests which is similar to the volume of requests the DVJ2 system had in the Fall 1999 semester. The resulting measures will reflect the cumulative behavior over the 50 different data sets.

5. Simulation Data Sources

This chapter will describe the sources and statistics of the data used for the simulations in this study. This chapter begins by describing the simulation file format and discussing several significant related studies in the area of Web workload trace analysis. The statistical tables and graphs for the trace workloads are then described. A number of the statistics for each of the trace workloads are brought together in tables to discuss the justification of delineating the three cache proxy server domains from Section 2.1 as well as a different domain of video cache proxy servers. Finally, a representative trace workload of each of the simulation data sources is described, and the statistics are given and discussed.

5.1 The Simulator Data Files and Related Analysis Studies

There are several different sources for the simulation data used in this dissertation, but the file format for each simulation data set is the same regardless of the source. Translation scripts were written in Perl to translate the source data files into a common simulator data format.

The common simulator data files have a specific format in which data is stored. Each of the files contains a header that indicates N, $Msmax$, $Msmin$, and M. Then each of the object sizes, the ms_i's, is listed along with the file types and unique integer identifier numbers. Finally, all of the N media object selections are listed by using the unique integer identifier numbers to save file space. Each of these lines is an access request, and each line stores the unique integer identifier number along with an access time code, the duration of the access, the time-to-retrieve, and the time to live of the object. Each of the time-related numbers have one-second accuracy. These simulation data files are generated by translating system logfiles into this file format or by producing data from stochastic usage models.

As these source data files were being translated by the Perl scripts, a number of statistics were determined for each workload trace. These statistics can help in hypothesizing which cache replacement algorithms will perform best (and worst) for the given system's workload trace. Most of the techniques for analyzing the statistics that were gathered from these logfiles are from papers by John Dilley, Martin Arlitt, and other researchers at the Internet Systems and Application Laboratory in the Hewlett-Packard Laboratories in Palo Alto, California. Their first paper on the subject [DFJR96] discusses measurement tools and modeling techniques for evaluating web servers. Arlitt and Williamson's papers analyzed six different web server access logs and developed the ten workload characteristics common to Internet Web servers [AW96, AW97]. Arlitt and Jin conducted an extensive analysis on the 1998 World Cup Web Site and reported the results in [AJ99]. Some of the data from the World Cup study is used in this study and is described more extensively in Section 5.4.6. Finally, Arlitt, Friedrich, and Jin published an HP-Labs Technical Report on an analysis of a cable modem Internet service provider's proxy cache server logfiles [AFJ99a]. That study provided some forward-looking insight into how access patterns change when clients have greater bandwidth for Internet access.

Several other papers are also worth mentioning. Almeida, Bestavros, Crovella, and de Oliveira analyzed logfiles from four different web servers in academia and industry, and they presented a model for both the temporal and spatial locality of accesses to these servers [ABCdO96]. The other three papers have already been mentioned with respect to their contributions of new cache replacement algorithms. Lorenzetti, Rizzo, and Vicisano analyzed a proxy cache logfile from the University of Pisa, Italy as a basis for their LRV cache replacement algorithm [LRV97]. Cao and Irani analyzed several proxy server logfiles and concluded that more than access times needed to be taken into account for proxy cache replacement algorithms to be effective [CI97]. This was part of their justification of their GreedyDual-Size algorithms. Finally, Breslau, Cao, Phillips, and Shenker analyzed the Zipf-like distribution characteristics of several proxy cache logfiles [BCF$^+$99].

5.2 Explanation of the Gathered Trace Statistics

For each of the workload traces, up to nine tables are generated as well as one graph. Each of the items below describe one of the tables while the last item describes the concentration of reference graphs. Please note that the term "bytes" is used generically for the size of objects; among the video files, these "bytes" are either kilobytes or seconds of video.

Summary of Access Log Characteristics This table imparts basic data about the workload trace including the start and end date stamps, the total number of requests, the average number of requests per minute, the total number of bytes transferred, and the average number of bytes transferred per minute. The requests per minute and average number of bytes transferred per minute give a rough estimate of how busy the server was on average over the time period that the workload was collected.

Summary of Other Characteristics The characteristics covered in this table are the maximum item size, minimum item size, and several measures that help determine the uniqueness of the items that are requested. The unique requests number is the number of unique items that were requested throughout the workload trace, while the unique workload size is the sum of the sizes of all of the unique items. The distinct requests/total requests is the percentage of the total requests of objects for the first time while the distinct bytes/total bytes is the percentage of bytes that were transferred due to a first-time request. These are essentially measures of the compulsory miss rate of the workload trace, without any regard for whether the object will fit into the cache. Ideally, a cache should only be caching objects that will be accessed again, so the second requests/distinct requests provides the percentage of the distinct objects that were accessed a second time while the second bytes/distinct bytes provides the percentage of the distinct bytes of object requests that were accessed a second time. Usually, objects that have been accessed twice will be accessed more times [LRV97]. Finally, the distinct files accessed only once and distinct bytes accessed only once are percentages of how many of the objects are accessed only once. That is, how many objects in the workload set are not accessed again after they are accessed the first time. These one-time-only objects don't need to be kept in the cache since they won't be accessed again; it is the challenge of the cache replacement algorithms to evict these objects from the cache or to not even let them into the cache by using an admission policy. But identifying such objects is much easier to explain than do. Overall, these statistics provide some rough estimates about how well a proxy cache server can perform optimally under the given workload.

Breakdown by File Size This table provides a breakdown of all of the accesses according to the size of the objects. The rows are broken down into powers of two. The smallest size category of 4 kilobytes and smaller was chosen because it involves sending only a few network packets. The largest size category of 128 kilobytes or larger was chosen because most large graphic, audio, and video files will fall into this category. There are two columns. The unique files column breaks down the percentages of each file size for all of the first accesses or compulsory misses of the cache. Then the multirequest column breaks down the percentages of each file size for all requests after that initial compulsory miss. That is, it addresses all of the reaccesses of objects. This table is intended to help determine whether file sizes should be included in the cache replacement algorithms; if there is a very high percentage of reaccesses that are under 4 kilobytes, then using size in the cache replacement algorithm may be beneficial.

Object Reaccess Information by File Size This table breaks down the accesses into first accesses, second accesses, three+ (three and greater) accesses, and total accesses for files in the aforementioned size categories. Generally, objects that have been accessed twice will be accessed more times, and this table tries to further address this observation from [LRV97]. This breakdown table shows which size category has the most second and three+ accesses as compared to their first accesses. The percentage value next to the first accesses number is the percent of the total accesses that the first accesses constitute. This is carried on by the percentage next to the second accesses being the percentage of second accesses over first accesses, and the percentage next to the three+ accesses being the percentage of three+ accesses over second accesses. The intent of these percentages is to show a breakdown of cascading accesses for certain categories (usually smaller) of file sizes. In all of the workload traces, it should be expected that smaller files would have higher reaccess rates than larger files. But, some workload traces should have higher reaccess rates for small files than others.

Breakdown by File Type This table displays the number of requests and number of bytes transferred broken down by the MIME file types [Mim00]. The MIME file types define a type and a subtype, but this breakdown only takes the type into account. The most common MIME types are: html, text, image (including gif and jpeg subtypes), application, video, and dynamic.

Multirequest Breakdown by File Type This table displays the number of requests and number of bytes transferred for items that were accessed more than once broken down by the MIME file types [Mim00]. The MIME file types define a type and a subtype, but this breakdown only takes the type into account. The most common MIME types are: html, text, image (including gif and jpeg subtypes), application, video, and dynamic. With the aforementioned Breakdown by File Type table and this table, the intention is to determine whether the file type would be a good criteria for use in the cache replacement algorithms for the given workload.

Object Reaccess Information by File Type This table breaks down the accesses into first accesses, second accesses, three+ (three and greater) accesses, and total accesses for files in the aforementioned file types. As it was also mentioned before, objects that have been accessed twice will be accessed more times, and this table also further addresses this observation from [LRV97]. This breakdown table shows which file types have the most second and three+ accesses as compared to their first accesses. The percentage value next to the first accesses number is the percent of the total accesses that the first accesses constitute. The percentage next to the second accesses being the percentage of second accesses over first accesses, and the percentage next to the three+ accesses being the percentage of three+ accesses over second accesses. The intent with these percentages is to show a breakdown of cascading accesses.

Unique File Size Information by File Type This table displays statistics on file sizes for each of the unique items in the workload traces broken down by MIME file type [Mim00] which were described above in the Breakdown by File Type table description. For each file type, the number of unique item files, mean item size, median item size, maximum item size, and sum of all unique item sizes (totalsize) is displayed. The idea behind this table is to provide more information on the breakdowns of file types and file sizes.

Stack Depth Analysis The stack depth analysis measures the temporal locality of the workload trace. It utilizes a standard least-recently used stack depth analysis. When a new item is requested, all the other items on the stack are pushed down one position, and the requested item's integer identifier is inserted on top of the stack. When an item is referenced again, its stack depth is recorded and the item's identifier is moved to the top of the stack with the other item identifiers being moved down to accommodate. Once the entire logfile has been analyzed, the list of rereferenced stack depths is analyzed. The stack depth analysis includes the mean stack depth, median stack depth, standard deviation of the stack depth, maximum stack depth, normalized mean stack depth, and normalized median stack depth. The two normalized statistics are taken by dividing the mean stack depth and median stack depth by the total number of re-references. Logfiles with a high degree of temporal locality have a relatively small normalized mean and median stack depths, while logfiles with a low degree of temporal locality have relatively large normalized mean and median stack depths [AJ99, AFJ99a]. Hence the intention of this analysis table is to determine the effectiveness of the LRU algorithm and other time-based algorithms as part of the cache replacement algorithm.

Concentration of References Graph The Concentration of References graph has three plot lines: requests, content data transferred, and storage space. The graph shows the distribution of popularity of the items that are requested in the logfile trace. All of the unique items in the workload set are sorted in decreasing order by the number of times they are requested. The item order is then translated to a percentage of total items which is plotted on the x-axis. Then the cumulative percentage of all client requests for each item is determined which becomes the requests plot line. The cumulative percentage of all content data transferred for each item is determined which becomes the content data transferred plot line. The requests and content data transferred plot lines show how popular the most-requested items are. If the plot lines increase rapidly near the y-axis, then the most requested items are very popular as compared to the other items. Finally, the storage space plot line is the cumulative percentage of the unique item sizes for the request-sorted items. The storage space plot line reflects what percentage of the total unique workload size a cache must be to store a certain percentage of the most popular items. When this plot line is shallow near the y-axis, it means that a relatively small cache could store the most popular items [AJ99, AFJ99a]. Essentially, this graph helps determine how much gain can be had for a cache proxy server for different size caches. For example, if a server was designed to cache the most popular 1% of the working set of objects, then it should be designed with the size of the first point of the storage space curve times the total unique workload size of workload trace.

5.3 The Domains for Proxy Cache Server Workload Traces

Some of the statistics described in Section 5.2 only convey characteristics about that particular trace, and cannot be compared to the statistics of other traces because they are not normalized by the number of requests, stack size, etc. However, other statistics from the same workload traces can be compared to each other, and they can be used to characterize different types of proxy cache server workload traces.

In Section 2.1, four domains in which distributed multimedia cache proxies are utilized were introduced. In this section, the three server domains and a video server domain are explored in terms of the statistical characteristics of their workload traces. In each of the tables of this section, the eight columns are statistics from eight different workload traces which represent those discussed in Section 5.4, 5.5, and 5.6. The worldcup label refers to the wc_week14 trace from the HP World Cup '98 origin server farm (further described in Section 5.4.6); the dsml label refers to the Purdue Digital Systems Multimedia Lab Web Server origin server trace (further described in Section 5.4.3); the uc label refers to the uc_day1 trace from the Urbana-Champaign NLANR network proxy cache server (further described in Section 5.4.5); the sj label refers to the sj_novwk5 trace from the San Jose NLANR network proxy cache server (further described in Section 5.4.5); the boeing label refers to the boeing.990301 trace from the sixth client proxy cache server from Boeing's firewall perimeter (further described in Section 5.4.4); the stack label refers to the trace from Purdue's Stack (ECN) client proxy cache server (further described in Section 5.4.2); the OnC label refers to a trace from an OnCommand movie on-demand video system (further described in Section 5.6); and the dvj2_99f label refers to the dvj2_99f trace from Purdue's Digital Video Jockey version 2 educational multimedia delivery system (further described in Section 5.5). It should be noted that the columns are ordered from origin server proxy traces to network proxy servers to client proxy servers to video-on-demand servers with two traces from each domain. This will help in the analysis of the statistics.

The first comparison table, Table 5.1, takes the request percentages from the Other Characteristics tables. As expected, the origin server traces have very low distinct request percentages and fairly high second request percentages. The video server traces have very similar percentages to the origin server traces, as should be expected, since they are similar to origin servers in functionality. The differences in these two trace sets comes in once-only accesses, of which the video server traces have very low percentages while the origin server traces have modest percentages. The two network proxy server traces have very high distinct access and lower once-only access percentages and correspondingly low second request percentages, as should be expected. Finally, the two client proxy traces have lower distinct request accesses and once-only access percentages than the network proxy server traces, and somewhat higher second request percentages. These statistics suggest that origin server proxy caches and video proxy cache servers will have rather low miss rates and byte miss rates overall. Conversely, network proxy caches will have quite high miss rates and byte miss rates. Meanwhile, client proxy cache servers will have rather high miss rates and byte miss rates, but not nearly as high as the network proxy cache servers.

Table 5.1
Comparison of Workload Composition Percentages

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
Distinct Requests/Total Requests (%)	1.61	4.88	81.09	77.13	53.82	54.51	2.23	1.10
Distinct Bytes/Total Bytes (%)	3.96	21.42	82.98	90.30	80.56	77.35	2.23	1.08
Second Requests/Distinct Requests (%)	73.79	42.63	6.79	6.96	15.54	16.93	100.0	95.58
Second Bytes/Distinct Bytes (%)	54.08	32.98	2.16	3.25	8.50	10.85	100.0	96.16
Distinct Files Accessed Only Once (%)	26.21	57.37	93.21	93.04	84.46	83.07	0.00	4.42
Distinct Bytes Accessed Only Once (%)	1.82	67.02	97.84	96.75	91.50	89.15	0.00	3.84

Table 5.2 is drawn from the first rows of the Object Reaccess Information by File Size tables. The percentages given are for files under 4 kilobytes, and the first row is the percentage of accesses of these files that were distinct (compulsory miss) accesses, while the second row is the percentage of distinct files that have been accessed a second time. For the two origin server traces, a very low percentage of all accesses of under 4 kilobyte files were first-time accesses, and a rather high percentage of these distinct files were accessed a second time. For the network proxy cache server traces, almost the exact opposite occurs: a very high percentage of all accesses are first-time

accesses while under 10% of the files are accessed a second time. Finally, for the client proxy traces, almost half of their accesses are first-time accesses, but around one in five of those first-time files are reaccessed. The statistics for the two video server traces are not available because all of the video objects on these systems are significantly more than 4 kilobytes in size. This table further suggests the miss rate and byte miss rate hypotheses presented in the previous paragraph.

Table 5.2
Comparison of Percentage of Files Under 4 kilobytes

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
First accesses (%)	0.9	1.7	80.5	71.8	46.0	45.7	n/a	n/a
Second accesses (%)	68.9	56.4	7.7	8.6	19.0	20.3	n/a	n/a

Tables 5.3 and 5.4 compare the percentage of requests and content data transferred for the three most requested file types: image, html, and application. In Table 5.3, the first three rows show the percentages of first-time requests of these file types while the last three rows show the percentages of subsequent (second time and more) requests. In this table we see that a very high percentage of accesses and reaccesses in the origin servers are for files of type "image" while around 10% to 20% are for type "html". There are many requests for type "application" in the dsml trace because many of the documents are Adobe PDF files which are classified as "application/pdf". In the network proxy cache traces, a very low percentage of all accesses are of type "application". Most first time accesses on the network proxy caches are of type "image" while the reaccess percentages of "image" and "html" types are more balanced. Finally, the client proxy traces have similar percentages to the network proxy traces, but they are not as unbalanced. That is, the access for "image" types versus "html" type are not as different, and the reaccess percentages are much closer to the first-time access percentages. This table suggests that using the "image" and "html" types as cache segment separators may decrease the miss rate of these proxy caches.

In Table 5.4, the first three rows show the percentages of the content data transferred by first-time request of these file types while the last three rows show the percentages of the content data transferred by subsequent (second time and more) requests. Compared to the requests, the percentages of content data transferred is not skewed as dramatically to the "image" file types. However, a large percentage of bytes transferred were for "application" types in the dsml, uc, sj, and boeing traces. This might suggest that using a file type separator for "application" type files may help decrease the byte miss rates on these types of traces.

Table 5.3
Comparison of Percentages of Requests for Image, Html, and Application File Types

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
First - image (%)	83.21	55.44	71.50	70.87	40.11	60.79	n/a	n/a
First - html (%)	12.28	14.02	22.05	23.39	35.77	34.29	n/a	n/a
First - app (%)	0.75	17.35	1.53	0.88	9.00	1.12	n/a	n/a
Subsequent - image (%)	83.86	57.62	41.34	61.58	43.83	69.46	n/a	n/a
Subsequent - html (%)	11.64	14.46	44.07	30.09	36.08	26.99	n/a	n/a
Subsequent - app (%)	0.75	14.51	2.93	1.49	5.69	1.09	n/a	n/a

Table 5.4
Comparison of Percentages of Content Data Transferred for Image, Html, and Application File Types

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
First - image (%)	41.92	20.79	36.47	49.68	4.77	47.96	n/a	n/a
First - html (%)	23.79	1.75	8.02	13.40	60.35	39.70	n/a	n/a
First - app (%)	1.59	63.37	31.12	17.84	30.73	3.45	n/a	n/a
Subsequent - image (%)	42.83	25.33	5.72	23.47	4.90	60.76	n/a	n/a
Subsequent - html (%)	23.39	2.15	11.16	23.65	28.66	31.09	n/a	n/a
Subsequent - app (%)	1.62	57.77	76.85	47.47	62.46	2.15	n/a	n/a

A comparison of the Stack Depth Analyses is presented in Table 5.5. Though the first four data rows of the table are interesting, these statistics are difficult to compare from one domain to another. The worldcup and dsml traces as well as the boeing and stack traces are from very different sized servers within the origin proxy server domain and client proxy server domain, respectively. However, the mean stack depths and median stack depths are similar. Also, the standard deviations of the worldcup and dsml traces are very similar. As would be expected, the uc and sj traces have very similar statistics as do the OnC and dvj2_99f traces. When the normalized mean and median depths are considered, the video server traces and origin server traces have the smallest numbers which means that their temporal localities are very high. For these two trace domains, LRU can be a very effective component of a cache replacement algorithm. The client proxy traces have less temporal locality, but they have more temporal locality than the network proxy cache traces. This should be expected when one considers the expected object sets and access patterns of the different domains as discussed in Section 2.1.

Table 5.5
Comparison of Stack Depth Analysis

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
mean stack depth	771.07	413.84	14466.71	13597.9	7817.23	2161.18	7.28	6.50
median stack depth	3051	3014	295	30	3	325	2	17
std. deviation	1529.25	1300.11	35153.37	32566.10	17124.71	5432.76	7.13	10.17
max. stack depth	14041	24800	251073	250569	132213	71075	31	98
norm. mean depth	0.0009	0.0080	0.0463	0.0413	0.0315	0.0164	0.0049	0.0006
norm. median depth	0.0035	0.0058	0.0009	0.0001	0.000015	0.0025	0.0014	0.0017

A comparison of the concentration of reference statistics in Tables 5.6, 5.7, and 5.8 yield some interesting results as well. Each of the three tables show concentration statistics for the most popular 1%, 3%, 5% and 10% objects in the workload traces. The Comparison of Concentration of Request References in Table 5.6 shows what percentage of all requests were requests for the most popular 1%, 3%, 5% and 10% objects in the workload traces. For instance, 55.4% of all requests in the worldcup trace were for the most popular 1% of the objects in the workload. The origin server traces have the highest concentration of references of their most popular objects, and they are followed with a large margin by the client proxy traces and network proxy traces. The video server traces have a fairly low popularity concentration of references.

In the Comparison of Concentration of Content Data Transferred of Table 5.7, the

Table 5.6
Comparison of Concentration of Request References

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
1% most referenced	55.4	68.3	13.7	16.9	31.1	25.2	20.1	3.5
3% most referenced	75.9	81.7	18.3	21.1	38.0	34.7	20.1	10.3
5% most referenced	82.4	87.0	21.5	25.2	42.0	39.8	34.4	14.8
10% most referenced	89.5	91.1	27.0	30.6	48.6	47.2	44.8	27.1

numbers are the percentages of the total content data transferred by the requests for the most popular 1%, 3%, 5% and 10% objects in the workload traces. As an example, 30.2% of all content data transferred of the worldcup workload trace were from the most popular 1% of the content data. Again we see the highest concentrations in the two origin server traces, with the network proxy traces and client proxy traces far behind. Also, the statistics of the video server traces are more similar to the client proxy and network proxy statistics.

Table 5.7
Comparison of Concentration of Content Data Transferred

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
1% most referenced bytes	30.2	20.3	16.0	6.6	7.2	6.1	20.1	4.6
3% most referenced bytes	49.1	31.8	17.0	7.5	12.8	12.3	20.1	12.2
5% most referenced bytes	67.1	45.6	18.1	9.4	15.2	16.0	34.4	16.6
10% most referenced bytes	80.9	65.9	22.0	16.5	19.6	23.7	44.8	28.8

In Table 5.8 which depicts the Comparison of Concentration of Storage Space Used, the numbers show the percentage of the total workload size that the most popular 1%, 3%, 5% and 10% objects in the workload traces used. So in the worldcup trace, the most popular 1% of the requests occupy only 0.2% of the size of the entire workload set. The origin server traces have the lowest storage space used for 1% and 3%, but they catch up to the client and network proxy traces when the 5% and 10% values are compared. This implies that the most popular objects on the origin servers are very small files, while the next tier of popular files in the origin server traces are somewhat larger. Since the video objects are much larger and are relatively the same size (within two orders of magnitude), the storage space concentrations of the video server traces show larger percentages.

Table 5.8
Comparison of Concentration of Storage Space Used

Workload Trace	worldcup	dsml	uc	sj	boeing	stack	OnC	dvj2_99f
1% most referenced bytes	0.2	0.1	0.6	0.3	0.4	0.2	3.0	1.2
3% most referenced bytes	1.3	0.5	1.1	0.8	1.3	1.2	3.0	3.2
5% most referenced bytes	3.6	1.7	1.7	1.5	2.1	2.2	6.1	4.9
10% most referenced bytes	7.0	6.5	6.0	7.5	4.3	6.1	9.1	10.3

5.4 Web Server Data Files

In order to better understand the different datafiles, more detailed information and statistics about the traces are necessary. The next several sections will provide these details about the workload traces that are used in this dissertation study including the web server data files, the DVJ2 data files, the OnCommand data files, and several stochastic model files.

5.4.1 Virginia Tech Proxy Servers

The Virginia Tech Proxy workload traces were collected for use in several studies conducted in the Virginia Tech Computer Science department. The traces were first introduced in a 1996 paper by Williams, Abrams, Standridge, Abdulla, and Fox [WAS$^+$96]. It is comprised of three different access traces collected from different scenarios. Each of these workload traces are described below.

BL Logfile Trace

The BL trace is a educational client proxy trace. It was recorded by a proxy machine over the course of 37 days in the Fall 1995 semester, and it recorded the Web accesses of the entire Computer Science department in that time period. There were 53,742 requests totaling 645 megabytes of data.

Table 5.10 shows that 57% of all requests were first-time requests (compulsory misses), while nearly 77% of all of the objects were accessed only once and they were generally smaller files. Also, around 20% of the objects accessed once were accessed a second time. Most of the files accessed and reaccessed were of type "image" and "html" though nearly 10% were of the type "other" as shown in Tables 5.13 and 5.14. As for the percentages of bytes transferred, types "image" and "html" are among the largest percentages, but types "application", "audio" and "video" are strong contingents as well. The Virginia Tech Computer Science department clients took advantage of their high bandwidth connections. The normalized stack depth statistics in Table 5.17 are fairly typical of client proxy workloads, and they imply that there is high temporal locality in the reaccesses. Finally, the Concentration of References graph in Figure 5.1 shows that the most popular 1% of all objects was accessed 13.8% of the time and they comprised only 9.3% of all bytes transferred. These are not as concentrated as other traces, but a good replacement algorithm can still perform well on this trace.

Table 5.9
Summary of Access Log Characteristics for BL Trace

Trace start time:	Sun Sep 17 00:24:57 1995
Trace end time:	Wed Oct 25 17:43:19 1995
Total Requests:	53,742
Avg Requests/Minute:	0.96
Total Bytes Transferred:	674,846,593
Avg Bytes Transferred/Minute:	12,103.06

Table 5.10
Summary of Other Characteristics for BL Trace

Maximum Size Item (bytes):	7,949,704
Minimum Size Item (bytes):	8
Unique Requests:	30,692
Unique Workload Size (bytes):	453,956,838
Distinct Requests/Total Requests (%):	57.11
Distinct Bytes/Total Bytes (%):	67.27
Second Requests/Distinct Requests (%):	23.14
Second Bytes/Distinct Bytes (%):	17.41
Distinct Files Accessed Only Once (%):	76.86
Distinct Bytes Accessed Only Once (%):	82.59

Table 5.11
Breakdown By File Size for BL Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq$ 4kb	59.91	66.20
4kb $< s \leq$ 8kb	16.35	14.38
8kb $< s \leq$ 16kb	10.62	8.86
16kb $< s \leq$ 32kb	6.51	5.60
32kb $< s \leq$ 64kb	3.65	2.39
64kb $< s \leq$ 128kb	1.75	1.61
128kb $< s$	1.21	0.98

Table 5.12
Object Reaccess Information by File Size for BL Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	18,388 (54.7 %)	4,469 (24.3 %)	10,789 (241.4 %)	33,646
4kb $< s \leq$ 8kb	5,017 (60.2 %)	1,102 (22.0 %)	2,212 (200.7 %)	8,331
8kb $< s \leq$ 16kb	3,260 (61.5 %)	712 (21.8 %)	1,330 (186.8 %)	5,302
16kb $< s \leq$ 32kb	1,997 (60.8 %)	432 (21.6 %)	858 (198.6 %)	3,287
32kb $< s \leq$ 64kb	1,120 (67.1 %)	217 (19.4 %)	333 (153.5 %)	1,670
64kb $< s \leq$ 128kb	538 (59.3 %)	108 (20.1 %)	262 (242.6 %)	908
128kb $< s$	372 (62.2 %)	63 (16.9 %)	163 (258.7 %)	598

Table 5.13
Breakdown By File Type for BL Trace

File Type	% of Requests	% of Bytes Transferred
application	1.69	11.59
audio	0.25	17.99
compressed	0.07	1.46
dynamic	0.59	0.27
html	32.43	14.39
image	54.15	46.22
java	0.71	0.23
other	9.57	3.71
text	0.49	0.56
video	0.05	3.57

Table 5.14
Multirequest Breakdown By File Type for BL Trace

File Type	% of Requests	% of Bytes Transferred
application	3.04	22.93
audio	0.13	7.03
compressed	0.03	0.37
dynamic	0.27	0.11
html	30.66	15.56
image	52.48	46.99
java	0.52	0.26
other	12.46	5.95
text	0.42	0.44
video	0.01	0.36

Table 5.15
Object Reaccess Information by File Type for BL Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	208 (22.9 %)	107 (51.4 %)	594 (555.1 %)	909
audio	104 (78.2 %)	14 (13.5 %)	15 (107.1 %)	133
compressed	33 (84.6 %)	5 (15.2 %)	1 (20.0 %)	39
dynamic	257 (80.6 %)	42 (16.3 %)	20 (47.6 %)	319
html	10,363 (59.5 %)	2,264 (21.8 %)	4,803 (212.1 %)	17,430
image	17,006 (58.4 %)	3,972 (23.4 %)	8,124 (204.5 %)	29,102
java	260 (68.6 %)	55 (21.2 %)	64 (116.4 %)	379
other	2,273 (44.2 %)	597 (26.3 %)	2,274 (380.9 %)	5,144
text	165 (63.0 %)	45 (27.3 %)	52 (115.6 %)	262
video	23 (92.0 %)	2 (8.7 %)	0 (0.0 %)	25

Table 5.16
Unique File Size Information by File Type for BL Trace

file type	number	mean (bytes)	median (bytes)	maximum (bytes)	totalsize (bytes)
all	30,692	14,791	303	7,949,704	453,956,838
application	208	132,563	320,537	1,859,386	27,573,121
audio	104	1,017,686	25,630	7,949,704	105,839,351
compressed	33	273,297	388,264	1,447,894	9,018,791
dynamic	257	6,117	3,493	78,804	1,572,133
html	10,363	6,057	2,977	1,830,557	62,766,593
image	17,006	12,239	3,055	1,808,969	208,144,919
java	260	3,786	3,394	21,834	984,293
other	2,273	5,245	3,033	646,844	11,922,598
text	165	17,096	25,765	342,889	2,820,888
video	23	1,013,659	283,206	2,770,091	23,314,151

Table 5.17
Stack Depth Analysis for BL Trace

mean stack depth:	2,773.87
median stack depth:	3
standard deviation:	4,342.17
maximum stack depth:	29,867
normalized mean depth:	0.051,614
normalized median depth:	0.000,056

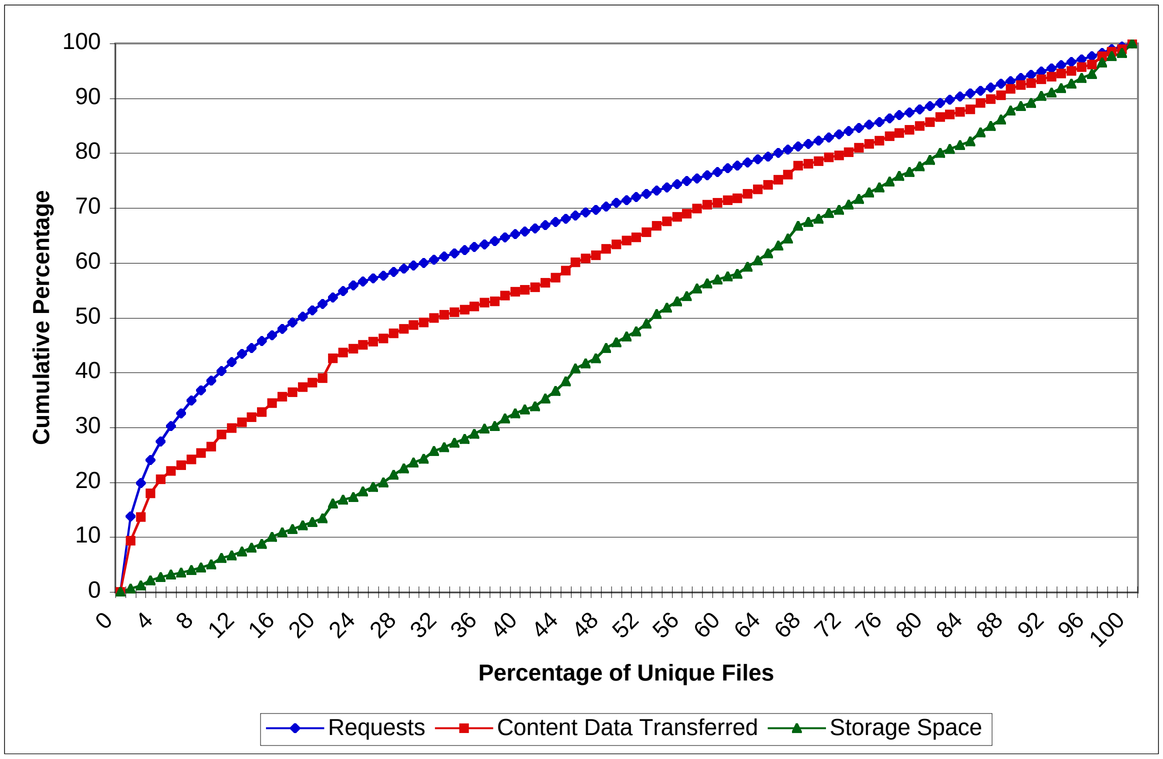


Fig. 5.1. Concentration of References for BL Trace

BR Logfile Trace

The BR workload trace is an origin server proxy workload trace. It is a record of all of the external access requests of the web servers in the `cs.vt.edu` domain. (That is no access requests are recorded from machines within the `cs.vt.edu` domain.) The trace is from 38 days in the Fall 1995 semester, and it involves 179,600 accesses that transferred 10.1 gigabytes of web objects. The first time requests (compulsory misses) were very low at 5.94% of all the requests, while the percentage of unique files accessed only once was only 41.54% as shown in Table 5.19. In the same table, one can see that nearly 60% of all objects were accessed at least twice, and over 82% of all the unique content was accessed at least twice.

In Tables 5.22 and 5.23, it can be seen that the overwhelming majority of the object requests and requests for objects that have been requested before in the trace were for types "image" and "html". However, the overwhelming majority of bytes transferred was for files of type "audio". By looking at the logfile, one finds that these audio files are basic audio (`.au`) files and midi audio (`.mid`) files. Some of these basic audio files are several megabytes in size which explains why the basic audio file requests comprise only a small percentage of total requests while constituting a large percentage of bytes transferred. The very small normalized stack depth statistics in Table 5.26 are common for origin server proxy workload traces. The Concentration of References graph in Figure 5.2 shows that the most popular 1% objects constitute 42.8% of all of the requests and 37.5% of all bytes transferred.

Table 5.18
Summary of Access Log Characteristics for BR Trace

Trace start time:	Sun Sep 17 00:19:27 1995
Trace end time:	Wed Oct 25 17:49:49 1995
Total Requests:	179,600
Avg Requests/Minute:	3.22
Total Bytes Transferred:	10,070,967,283
Avg Bytes Transferred/Minute:	180,579.18

Table 5.19
Summary of Other Characteristics for BR Trace

Maximum Size Item (bytes):	10,032,887
Minimum Size Item (bytes):	18
Unique Requests:	10,660
Unique Workload Size (bytes):	217,525,043
Distinct Requests/Total Requests (%):	5.94
Distinct Bytes/Total Bytes (%):	2.16
Second Requests/Distinct Requests (%):	58.46
Second Bytes/Distinct Bytes (%):	82.44
Distinct Files Accessed Only Once (%):	41.54
Distinct Bytes Accessed Only Once (%):	17.56

Table 5.20
Breakdown By File Size for BR Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq$ 4kb	69.52	66.91
4kb $< s \leq$ 8kb	12.05	12.10
8kb $< s \leq$ 16kb	7.44	7.25
16kb $< s \leq$ 32kb	5.06	7.00
32kb $< s \leq$ 64kb	3.02	3.17
64kb $< s \leq$ 128kb	1.49	0.79
128kb $< s$	1.43	2.78

Table 5.21
Object Reaccess Information by File Size for BR Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	7,411 (6.2 %)	4,211 (56.8 %)	108,824 (2584.3 %)	120,446
4kb $< s \leq$ 8kb	1,284 (5.9 %)	777 (60.5 %)	19,659 (2530.1 %)	21,720
8kb $< s \leq$ 16kb	793 (6.1 %)	481 (60.7 %)	11,771 (2447.2 %)	13,045
16kb $< s \leq$ 32kb	539 (4.4 %)	336 (62.3 %)	11,483 (3417.6 %)	12,358
32kb $< s \leq$ 64kb	322 (5.7 %)	213 (66.1 %)	5,149 (2417.4 %)	5,684
64kb $< s \leq$ 128kb	159 (10.6 %)	101 (63.5 %)	1,235 (1222.8 %)	1,495
128kb $< s$	152 (3.1 %)	113 (74.3 %)	4,587 (4059.3 %)	4,852

Table 5.22
Breakdown By File Type for BR Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	0.49	0.54
audio	3.18	87.79
compressed	0.02	0.01
dynamic	0.72	0.07
html	20.96	2.95
image	67.48	8.09
java	0.00	0.00
other	6.44	0.37
text	0.70	0.13
video	0.01	0.04

Table 5.23
Multirequest Breakdown By File Type for BR Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	0.38	0.40
audio	3.34	88.45
compressed	0.01	0.01
dynamic	0.69	0.05
html	19.40	2.80
image	69.36	7.85
java	0.00	0.00
other	6.31	0.34
text	0.50	0.09
video	0.00	0.03

Table 5.24
Object Reaccess Information by File Type for BR Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	236 (26.8 %)	153 (64.8 %)	493 (322.2 %)	882
audio	64 (1.1 %)	59 (92.2 %)	5,591 (9476.3 %)	5,714
compressed	10 (34.5 %)	6 (60.0 %)	13 (216.7 %)	29
dynamic	135 (10.4 %)	62 (45.9 %)	1,100 (1774.2 %)	1,297
html	4,864 (12.9 %)	2,613 (53.7 %)	30,162 (1154.3 %)	37,639
image	4,017 (3.3 %)	2,641 (65.7 %)	114,539 (4337.0 %)	121,197
java	3 (75.0 %)	1 (33.3 %)	0 (0.0 %)	4
other	911 (7.9 %)	566 (62.1 %)	10,087 (1782.2 %)	11,564
text	416 (32.9 %)	127 (30.5 %)	721 (567.7 %)	1,264
video	4 (40.0 %)	4 (100.0 %)	2 (50.0 %)	10

Table 5.25
Unique File Size Information by File Type for BR Trace

file type	number	mean (bytes)	median (bytes)	maximum (bytes)	totalsize (bytes)
all	10,660	20,406	319	10,032,887	217,525,043
application	236	65,929	367,686	793,812	15,559,287
audio	64	1,975,407	2,115,765	10,032,887	126,426,073
compressed	10	45,808	20,480	102,400	458,080
dynamic	135	18,314	29,565	209,173	2,472,430
html	4,864	4,455	303	132,815	21,667,455
image	4,017	10,254	3,484	590,657	41,189,064
java	3	2,346	494	6,334	7,037
other	911	4,628	238	133,916	4,216,349
text	416	9,273	3,549	163,943	3,857,467
video	4	417,950	499,812	499,812	1,671,801

Table 5.26
Stack Depth Analysis for BR Trace

mean stack depth:	544.8
median stack depth:	3
standard deviation:	976.46
maximum stack depth:	10,392
normalized mean depth:	0.003,033
normalized median depth:	0.000,017

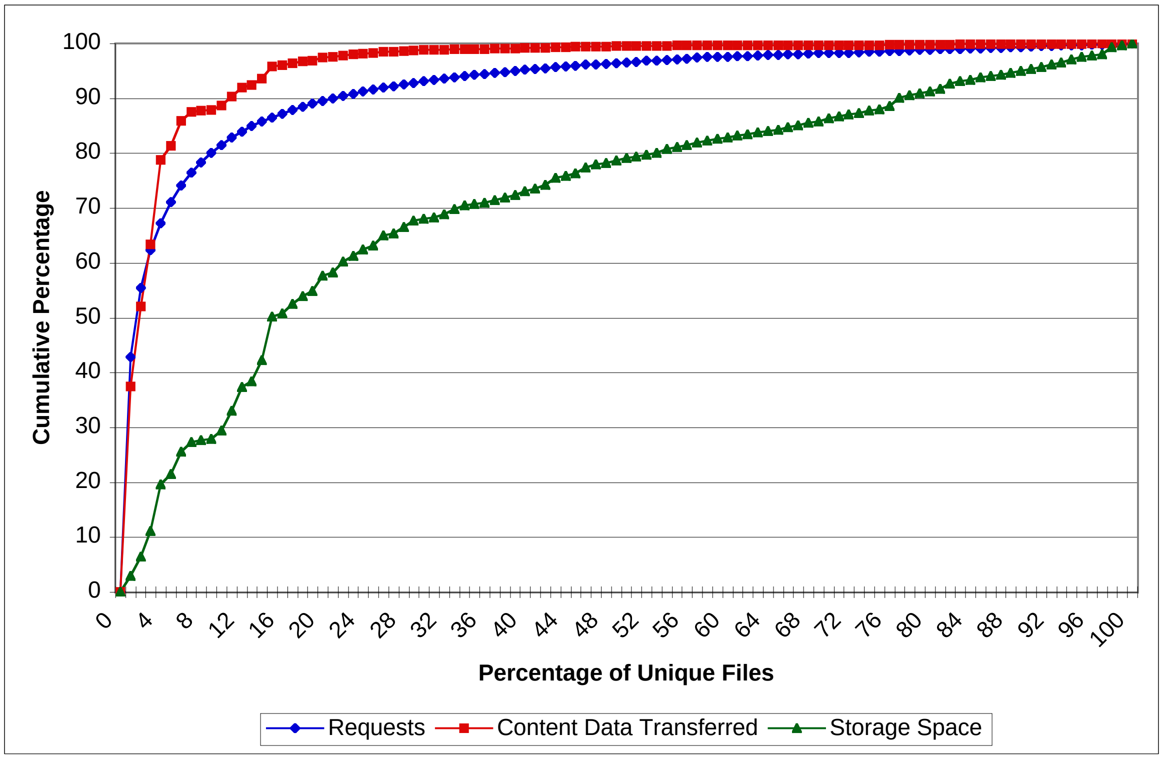


Fig. 5.2. Concentration of References for BR Trace

G Logfile Trace

The G logfile trace of the Virginia Tech trace set was collected at a popular workstation at which at least 25 Computer Science graduate students accessed the Web. This trace is classified as a small educational client proxy trace. In the trace there are 33,155 access and 486 megabytes were transferred which occurred over the course of two and a half months in the Spring 1995 semester. The first time requests (compulsory misses) were almost 60% of all the requests, while the percentage of unique files accessed only once was 80% as shown in Table 5.28. Also, fewer than 20% of the unique files were requested a second time. The access distribution is fairly diverse because Table 5.28 shows fairly high percentage for distinct objects that are accessed only once.

Tables 5.31 and 5.32 show that the majority of requests and bytes transferred are of type "image" and "html", while there is a significant percentage of type "other". Also, a significant percentage of total transferred bytes are of type "video". These videos are generally only accessed once, because such a high percentage does not show up in the bytes transferred of files that are accessed more than once in Table 5.32. In Table 5.35, the normalized mean stack depth is rather high while the normalized median stack depth is rather low for a client proxy trace. Finally, the Concentration of Reference graph lines in Figure 5.3 have a rather shallow attack as compared with other client proxy traces.

Table 5.27
Summary of Access Log Characteristics for G Trace

Trace start time:	Fri Jan 20 11:26:09 1995
Trace end time:	Fri Apr 7 23:50:27 1995
Total Requests:	33,155
Avg Requests/Minute:	0.30
Total Bytes Transferred:	486,368,347
Avg Bytes Transferred/Minute:	4,357.19

Table 5.28
Summary of Other Characteristics for G Trace

Maximum Size Item (bytes):	25,926,373
Minimum Size Item (bytes):	1
Unique Requests:	19,782
Unique Workload Size (bytes):	398,604,542
Distinct Requests/Total Requests (%):	59.67
Distinct Bytes/Total Bytes (%):	81.96
Second Requests/Distinct Requests (%):	19.96
Second Bytes/Distinct Bytes (%):	8.41
Distinct Files Accessed Only Once (%):	80.04
Distinct Bytes Accessed Only Once (%):	91.59

Table 5.29
Breakdown By File Size for G Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq 4$kb	68.42	77.77
4kb $< s \leq 8$kb	11.61	7.72
8kb $< s \leq 16$kb	7.81	6.33
16kb $< s \leq 32$kb	4.94	4.79
32kb $< s \leq 64$kb	3.57	2.33
64kb $< s \leq 128$kb	1.72	0.61
128kb $< s$	1.93	0.45

Table 5.30
Object Reaccess Information by File Size for G Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq 4$kb	13,534 (56.5 %)	2,912 (21.5 %)	7,488 (257.1 %)	23,934
4kb $< s \leq 8$kb	2,297 (69.0 %)	396 (17.2 %)	636 (160.6 %)	3,329
8kb $< s \leq 16$kb	1,544 (64.6 %)	268 (17.4 %)	579 (216.0 %)	2,391
16kb $< s \leq 32$kb	977 (60.4 %)	176 (18.0 %)	465 (264.2 %)	1,618
32kb $< s \leq 64$kb	707 (69.4 %)	123 (17.4 %)	189 (153.7 %)	1,019
64kb $< s \leq 128$kb	341 (80.8 %)	39 (11.4 %)	42 (107.7 %)	422
128kb $< s$	382 (86.4 %)	35 (9.2 %)	25 (71.4 %)	442

Table 5.31
Breakdown By File Type for G Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	0.81	8.75
audio	0.07	0.88
compressed	0.40	7.04
dynamic	5.15	1.34
html	25.89	10.73
image	49.70	33.50
java	0.01	0.00
other	16.93	8.24
text	0.63	0.77
video	0.41	28.75

Table 5.32
Multirequest Breakdown By File Type for G Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	0.67	8.40
audio	0.01	0.24
compressed	0.13	2.54
dynamic	3.95	1.72
html	22.28	18.02
image	55.23	59.03
java	0.00	0.00
other	17.33	7.08
text	0.36	0.40
video	0.04	2.58

Table 5.33
Object Reaccess Information by File Type for G Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	178 (66.4 %)	38 (21.3 %)	52 (136.8 %)	268
audio	21 (91.3 %)	2 (9.5 %)	0 (0.0 %)	23
compressed	116 (87.2 %)	12 (10.3 %)	5 (41.7 %)	133
dynamic	1,179 (69.1 %)	198 (16.8 %)	330 (166.7 %)	1,707
html	5,606 (65.3 %)	1,047 (18.7 %)	1,932 (184.5 %)	8,585
image	9,092 (55.2 %)	1,910 (21.0 %)	5,476 (286.7 %)	16,478
java	2 (100.0 %)	0 (0.0 %)	0 (0.0 %)	2
other	3,296 (58.7 %)	713 (21.6 %)	1,604 (225.0 %)	5,613
text	161 (77.0 %)	23 (14.3 %)	25 (108.7 %)	209
video	131 (95.6 %)	6 (4.6 %)	0 (0.0 %)	137

Table 5.34
Unique File Size Information by File Type for G Trace

file type	number	mean (bytes)	median (bytes)	maximum (bytes)	totalsize (bytes)
all	19,782	20,150	2,922	25,926,373	398,604,542
application	178	197,564	27,475	3,732,116	35,166,362
audio	21	193,455	299,669	1,743,400	4,062,560
compressed	116	275,844	309,033	2,909,871	31,997,949
dynamic	1,179	4,249	2,862	476,521	5,009,092
html	5,606	6,489	2,960	961,739	36,374,995
image	9,092	12,222	300	1,053,970	111,123,548
java	2	3,080	1,630	4,529	6,159
other	3,296	10,277	2,708	13,498,876	33,873,192
text	161	21,163	3,569	724,617	3,407,254
video	131	1,050,255	292,455	25,926,373	137,583,431

Table 5.35
Stack Depth Analysis for G Trace

mean stack depth:	1,601.41
median stack depth:	3
standard deviation:	2,489.5
maximum stack depth:	16,829
normalized mean depth:	0.048,301
normalized median depth:	0.0001

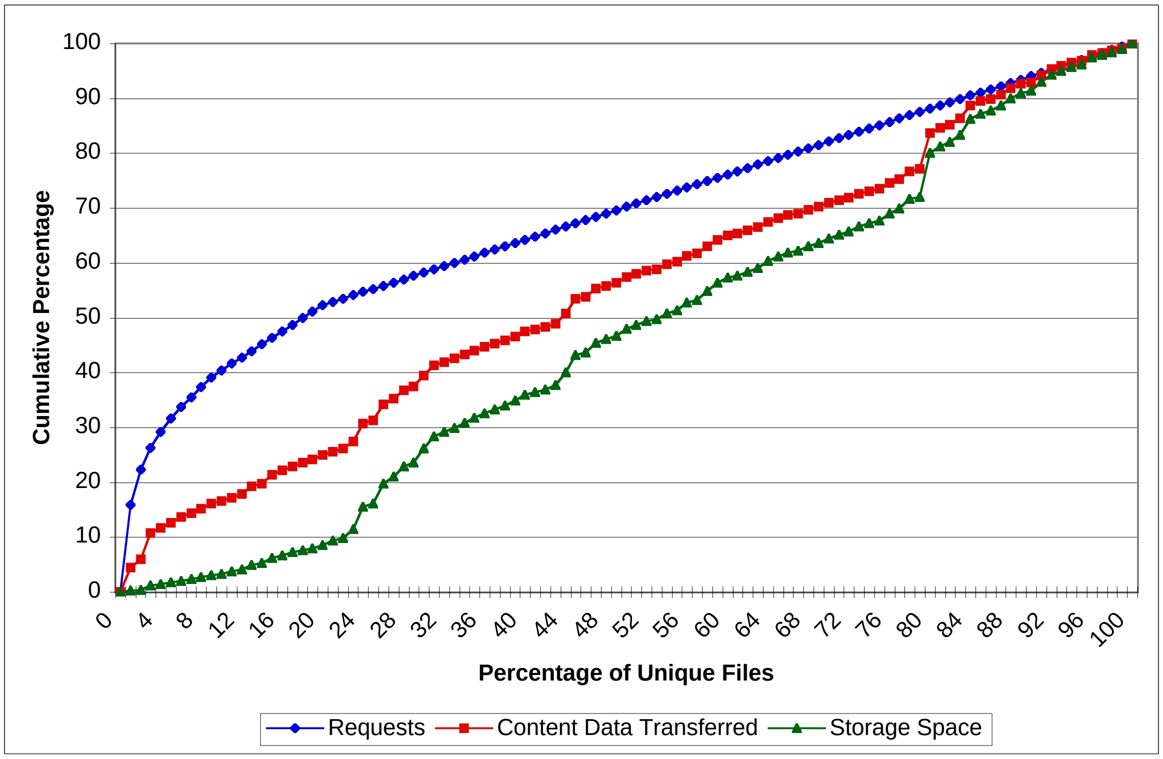


Fig. 5.3. Concentration of References for G Trace

5.4.2 Purdue Stack (ECN) Proxy Server

The Purdue Stack Proxy Server is maintained by the Purdue University Engineering Computer Network (ECN), and it is run on the `stack.ecn.purdue.edu` server. This proxy is used by many individuals on the ECN staff as well as a few engineering faculty members and students to cache objects that they are accessing from the Internet. Since this proxy does not serve a great number of people, its references per minute is relatively low. But this trace is comprised of fourteen months of accesses (from September 1998 to October 1999), and it is a complete and sizable data set for a smaller client set. A total of 915 megabytes of data were transferred in 131,823 requests.

Since the Stack proxy is a client proxy, the percentage of first time requests (compulsory misses) is somewhat high at 54.51%, and the percentage of distinct files that are accessed only once is high at 83% as is shown in Table 5.37. Table 5.40 reflects that almost all of the accessed objects and objects accessed more than once are of type "image" and "html". The normalized stack depth statistics in Table 5.44 show that there is moderate temporal locality among reaccesses of objects. Finally, the Concentration of References graph shows that the most popular 1% of the objects comprise 25.2% of all requests, while they only comprise 0.2% of the size of the entire unique data set. This implies that by caching the right objects, fairly low miss rates can be attained.

Table 5.36
Summary of Access Log Characteristics for stackproxy Trace

Trace start time:	Mon Aug 31 07:46:32 1998
Trace end time:	Thu Oct 28 16:14:03 1999
Total Requests:	131,823
Avg Requests/Minute:	0.22
Total Bytes Transferred:	914,688,843
Avg Bytes Transferred/Minute:	1,500.41

Table 5.37
Summary of Other Characteristics for stackproxy Trace

Maximum Size Item (bytes):	17,670,168
Minimum Size Item (bytes):	1
Unique Requests:	71,853
Unique Workload Size (bytes):	707,494,492
Distinct Requests/Total Requests (%):	54.51
Distinct Bytes/Total Bytes (%):	77.35
Second Requests/Distinct Requests (%):	16.93
Second Bytes/Distinct Bytes (%):	10.85
Distinct Files Accessed Only Once (%):	83.07
Distinct Bytes Accessed Only Once (%):	89.15

Table 5.38
Breakdown By File Size for stackproxy Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq$ 4kb	55.73	79.25
4kb $< s \leq$ 8kb	14.14	9.90
8kb $< s \leq$ 16kb	17.56	7.09
16kb $< s \leq$ 32kb	7.11	2.71
32kb $< s \leq$ 64kb	3.68	0.81
64kb $< s \leq$ 128kb	1.25	0.21
128kb $< s$	0.53	0.04

Table 5.39
Object Reaccess Information by File Size for stackproxy Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	40,043 (45.7 %)	8,146 (20.3 %)	39,379 (483.4 %)	87,568
4kb $< s \leq$ 8kb	10,159 (63.1 %)	1,513 (14.9 %)	4,422 (292.3 %)	16,094
8kb $< s \leq$ 16kb	12,617 (74.8 %)	1,499 (11.9 %)	2,753 (183.7 %)	16,869
16kb $< s \leq$ 32kb	5,112 (75.9 %)	674 (13.2 %)	952 (141.2 %)	6,738
32kb $< s \leq$ 64kb	2,641 (84.5 %)	249 (9.4 %)	237 (95.2 %)	3,127
64kb $< s \leq$ 128kb	901 (87.9 %)	67 (7.4 %)	57 (85.1 %)	1,025
128kb $< s$	380 (94.5 %)	14 (3.7 %)	8 (57.1 %)	402

Table 5.40
Breakdown By File Type for stackproxy Trace

File Type	% of Requests	% of Bytes Transferred
application	1.12	3.45
audio	0.09	0.23
compressed	0.01	0.27
dynamic	2.78	2.52
html	34.29	39.70
image	60.79	47.96
java	0.27	0.20
multipart	0.03	1.16
other	0.29	0.65
text	0.30	1.50
video	0.01	2.35

Table 5.41
Multirequest Breakdown By File Type for stackproxy Trace

File Type	% of Requests	% of Bytes Transferred
application	1.09	2.15
audio	0.04	0.11
compressed	0.00	0.00
dynamic	1.61	1.28
html	26.99	31.09
image	69.46	60.76
java	0.26	0.33
multipart	0.01	1.01
other	0.23	0.20
text	0.32	2.97
video	0.01	0.08

Table 5.42
Object Reaccess Information by File Type for stackproxy Trace

File Type	**First Accesses**	**Second Accesses**	**Other Accesses**	**Total Accesses**
application	822 (55.7 %)	274 (33.3 %)	380 (138.7 %)	1,476
audio	97 (80.2 %)	15 (15.5 %)	9 (60.0 %)	121
compressed	10 (100.0 %)	0 (0.0 %)	0 (0.0 %)	10
dynamic	2,707 (73.8 %)	252 (9.3 %)	711 (282.1 %)	3,670
html	29,013 (64.2 %)	3,722 (12.8 %)	12,464 (334.9 %)	45,199
image	38,486 (48.0 %)	7,763 (20.2 %)	33,890 (436.6 %)	80,139
java	207 (57.2 %)	57 (27.5 %)	98 (171.9 %)	362
multipart	29 (85.3 %)	4 (13.8 %)	1 (25.0 %)	34
other	266 (66.5 %)	36 (13.5 %)	98 (272.2 %)	400
text	206 (51.6 %)	38 (18.4 %)	155 (407.9 %)	399
video	10 (76.9 %)	1 (10.0 %)	2 (200.0 %)	13

Table 5.43
Unique File Size Information by File Type for stackproxy Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	71,853	9,846	2,951	17,670,168	707,494,492
application	822	32,913	260	4,538,214	27,054,704
audio	97	18,967	2,899	242,628	1,839,827
compressed	10	245,399	53,834	1,120,541	2,453,985
dynamic	2,707	7,526	366	2,986,038	20,372,821
html	29,013	10,295	2,567	705,284	298,698,146
image	38,486	8,128	325	843,452	312,823,559
java	207	5,426	2,446	30,581	1,123,098
multipart	29	293,561	4,178	1,439,523	8,513,281
other	400	235	249	279	24,439
text	206	36,909	2,900	4,866,818	7,603,266
video	10	2,134,526	258,981	17,670,168	21,345,258

Table 5.44
Stack Depth Analysis for stackproxy Trace

mean stack depth:	2,161.18
median stack depth:	325
standard deviation:	5,432.76
maximum stack depth:	71,075
normalized mean depth:	0.016,395
normalized median depth:	0.002,465

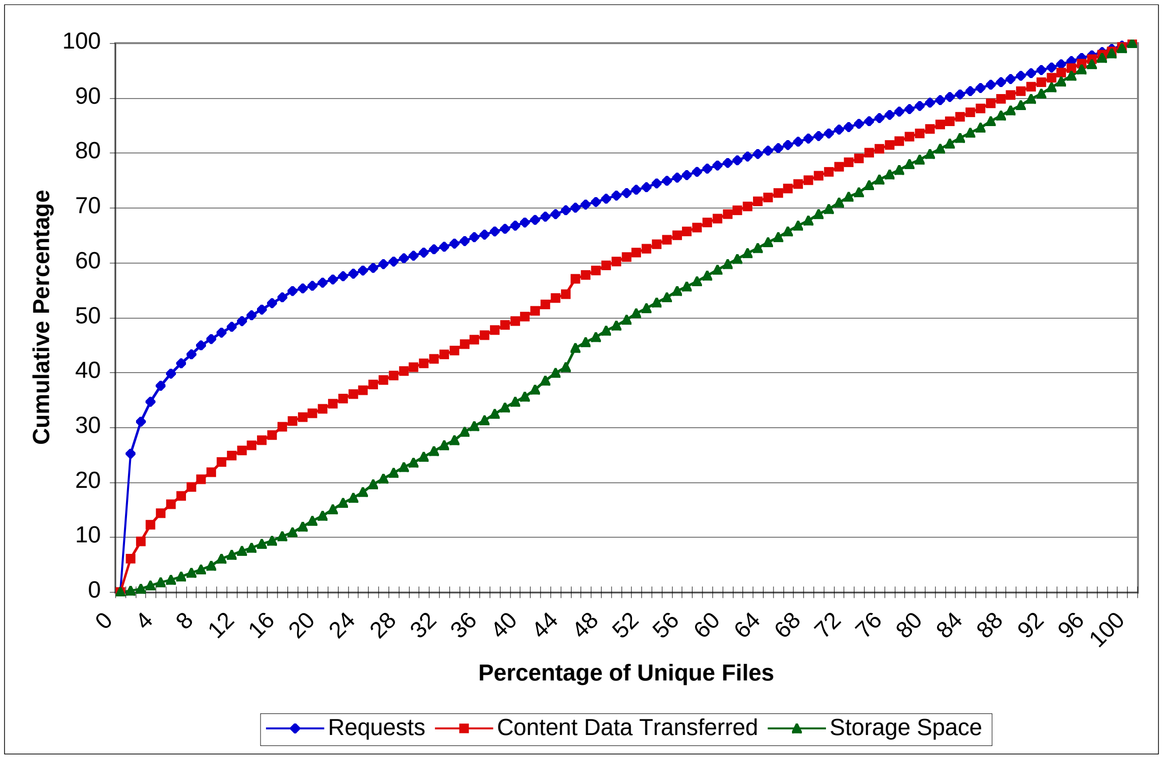


Fig. 5.4. Concentration of References for stackproxy Trace

5.4.3 Purdue DSML Web Server

The Digital Systems/Multimedia Learning Lab (DSML) is in the School of Electrical and Computer Engineering at Purdue University. The DSML web server is the origin of web pages not only for the Digital Systems Laboratory, which conducts research in Multimedia Education and Computer Architecture, but also for several classes offered by the School of Electrical and Computer Engineering. These classes include EE 266/7 *Digital Logic Design*, EE 362 *Microprocessor System and Interfacing*, EE 467 *Advanced Digital Systems/Embedded Microcontroller Design Lab*, EE 477 *Digital Systems Senior Project*, EE 566 *CISC Microprocessor System Design*, and EE 568 *RISC and DSP Microprocessor System Design.* The EE 266/7 and EE 362 classes use this web server extensively to view and download course documents, homework assignments, lab assignments, reference documents, and links to online digital videos (see Section 5.5). These digital video links spawn a helper application with which the students actually view the videos. This workload trace was collected during the Fall 1999 semester in which the DSML server had 520,783 requests and transferred 10.7 gigabytes of content. In Table 5.46, the percentage of first time requests (compulsory misses) was 4.88%, while the percentage of unique files that were accessed only once was 57.37%. And over 40% of the objects that were accessed for the first time were accessed for a second time.

From Table 5.49, one can see that most of the requests of this server are of type "image" and "html", though "application" and "other" types are also significantly represented. Most of the "application" type requests are for Adobe PDF files which is a common format for homework, lecture notes, and other course documents on this web server. It should be noted in Tables 5.49 and 5.50 that the "application" type comprises more than half of all bytes transferred by the server. Most of the "other" type requests are for ABEL hardware description language source files, assembly code source files, and html index pages that are referenced in the URL with an ending right slash. The large percentage of file types other than "images" and "html" types in this workload trace implies that PLRU will probably not perform well since PLRU gives priority to “images” and “html” types.

Because the server delivers educational material, a great deal of its content is time sensitive with respect to the time within a semester. Therefore, this workload trace shows a better temporal locality in the stack depth analysis of Table 5.53 and Correlation of Reference graph of Figure 5.5, and shows a relatively low percentage of documents that are referenced only once. Not surprisingly, the most popular 1% of all of the documents served by the DSML web server comprises 68.3% of all accesses, and the most popular 10% comprises 91.7% of all accesses. This implies that cache replacement algorithms based on LRU and LFU should perform well with this workload trace as input. The Content Data Transferred line of the Correlation of References graph implies that size-based cache replacement algorithms could help improve performance.

Table 5.45
Summary of Access Log Characteristics for dsl_log Trace

Trace start time:	Mon Aug 23 00:15:53 1999
Trace end time:	Fri Dec 17 14:13:22 1999
Total Requests:	520,783
Avg Requests/Minute:	3.10
Total Bytes Transferred:	10,668,883,959
Avg Bytes Transferred/Minute:	63.00

Table 5.46
Summary of Other Characteristics for dsl_log Trace

Maximum Size Item (bytes):	16,154,624
Minimum Size Item (bytes):	1
Unique Requests:	25,395
Unique Workload Size (bytes):	2,285,530,300
Distinct Requests/Total Requests (%):	4.88
Distinct Bytes/Total Bytes (%):	21.42
Second Requests/Distinct Requests (%):	42.63
Second Bytes/Distinct Bytes (%):	32.98
Distinct Files Accessed Only Once (%):	57.37
Distinct Bytes Accessed Only Once (%):	67.02

Table 5.47
Breakdown By File Size for dsl_log Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq 4$kb	22.17	65.76
4kb $< s \leq$ 8kb	7.01	12.70
8kb $< s \leq$ 16kb	6.97	4.13
16kb $< s \leq$ 32kb	9.84	9.47
32kb $< s \leq$ 64kb	21.45	3.79
64kb $< s \leq$ 128kb	19.96	2.37
128kb $< s$	12.59	1.78

Table 5.48
Object Reaccess Information by File Size for dsl_log Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq 4$kb	5,630 (1.7 %)	3,175 (56.4 %)	322,589 (10160.3 %)	331,394
4kb $< s \leq$ 8kb	1,781 (2.8 %)	949 (53.3 %)	61,958 (6528.8 %)	64,688
8kb $< s \leq$ 16kb	1,771 (8.0 %)	797 (45.0 %)	19,687 (2470.1 %)	22,255
16kb $< s \leq$ 32kb	2,499 (5.1 %)	1,045 (41.8 %)	45,858 (4388.3 %)	49,402
32kb $< s \leq$ 64kb	5,447 (22.5 %)	2,195 (40.3 %)	16,603 (756.4 %)	24,245
64kb $< s \leq$ 128kb	5,069 (30.2 %)	1,691 (33.4 %)	10,045 (594.0 %)	16,805
128kb $< s$	3,198 (26.7 %)	973 (30.4 %)	7,823 (804.0 %)	11,994

Table 5.49
Breakdown By File Type for dsl_log Trace

File Type	% of Requests	% of Bytes Transferred
application	17.35	63.37
compressed	0.01	0.03
dynamic	0.00	0.00
html	14.02	1.75
image	55.44	20.79
other	12.77	13.78
text	0.40	0.01
video	0.00	0.26

Table 5.50
Multirequest Breakdown By File Type for dsl_log Trace

File Type	% of Requests	% of Bytes Transferred
application	14.51	57.77
compressed	0.00	0.01
dynamic	0.00	0.00
html	14.46	2.15
image	57.62	25.33
other	12.98	14.74
text	0.41	0.01
video	0.00	0.00

Table 5.51
Object Reaccess Information by File Type for dsl_log Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	18,467 (20.4 %)	7,338 (39.7 %)	64,555 (879.7 %)	90,360
compressed	24 (66.7 %)	4 (16.7 %)	8 (200.0 %)	36
dynamic	4 (44.4 %)	3 (75.0 %)	2 (66.7 %)	9
html	1,380 (1.9 %)	949 (68.8 %)	70,706 (7450.6 %)	73,035
image	3,298 (1.1 %)	1,388 (42.1 %)	284,058 (20465.3 %)	288,744
other	2,169 (3.3 %)	1,099 (50.7 %)	63,223 (5752.8 %)	66,491
text	48 (2.3 %)	43 (89.6 %)	2,011 (4676.7 %)	2,102
video	5 (83.3 %)	1 (20.0 %)	0 (0.0 %)	6

Table 5.52
Unique File Size Information by File Type for dsl_log Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	25,395	89,999	3,693	16,154,624	2,285,530,300
application	18,467	103,876	42,050	4,519,129	1,918,281,185
compressed	24	123,383	5,402	1,809,428	2,961,186
dynamic	4	181	214	214	725
html	1,380	4,985	237	201,691	6,878,846
image	3,298	28,788	27,624	608,318	94,941,927
other	2,169	107,968	212	13,452,800	234,183,158
text	48	758	294	4,745	36,375
video	5	5,649,380	229	16,154,624	28,246,898

Table 5.53
Stack Depth Analysis for dsl_log Trace

mean stack depth:	413.84
median stack depth:	3,014
standard deviation:	1,300.11
maximum stack depth:	24,800
normalized mean depth:	0.000,795
normalized median depth:	0.005,787

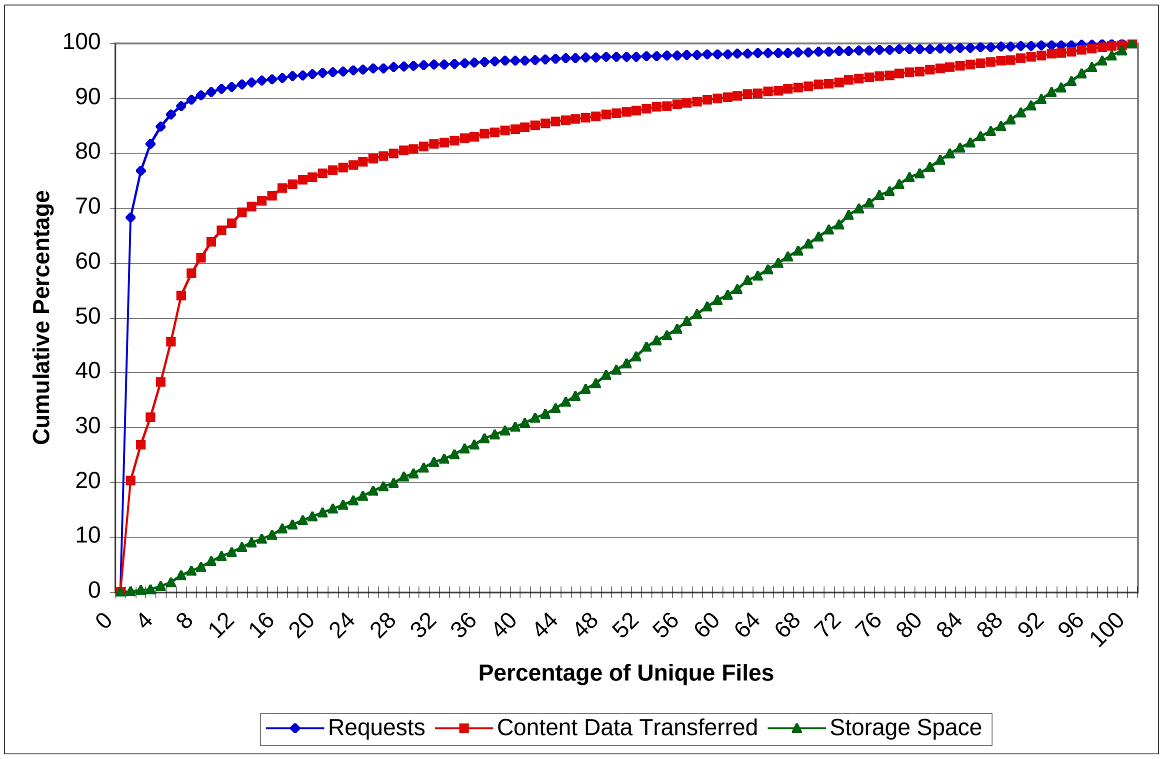


Fig. 5.5. Concentration of References for dsl_log Trace

5.4.4 Boeing Proxy Cache Server Traces

The Boeing Puget Sound firewall perimeter consists of six proxy cache servers for employee Web accesses. Boeing Corporate Headquarters is in the Puget Sound and Seattle areas and employs several tens of thousands of people at various locations in that area. Five of the six proxy servers operate in a round-robin fashion and do not record the file type of the objects that they cache. The sixth server is used for occasional internal proxy cache testing. It receives some of the proxy caching load when it is not in testing mode, and also gets certain web site and ftp traffic directed to it. Because of this, the sixth server's data files were manageably small number of accesses in each data file (under 500,000 per day). All of the other data files had around four million requests per day per server. The trace set consists of five days (from March 1 to March 5, 1999) of access data split into 24-hour long files. The data presented in the tables and figure below are from the March 1, 1999 trace workload data set.

On March 1, 1999 on the sixth Boeing server, there were 249,079 object requests totaling 8.72 gigabytes of data transferred. This turns out to be an average of 173 accesses per minute and 6.1 megabytes of data transferred per minute. Table 5.55 shows that 54% of the accesses are first time accesses (compulsory misses), while 84% of the objects were accessed only once. Also, only around 15% of all unique objects were accessed a second time. Most of the accesses were of type "image", "text", and "application" as shown by Tables 5.58 and 5.59. From examining the logfile, the subtypes of the "application" types are mostly "octet-stream" and "x-javascripts" which are downloaded executables. Also the "text" files are almost all of subtype "html". In the stack depth analysis of Table 5.62, the normalized mean stack depth is fairly large while the normalized median stack depth is very small. Looking at the actual numbers in the same table shows that the standard deviation and the maximum stack depth are quite large. This implies that most of an object's reaccesses are fairly soon after its previous access. However, there are a minority of reaccesses that occur a significant time later, which skews the mean stack depth. With such a low normalized mean stack depth, LRU-based cache replacement algorithms should perform well. Finally the Concentration of References graph in Figure 5.6 shows a moderate initial attack on the graph lines showing that the most popular files are among the most referenced. It show that the most popular 1% of the objects are accessed 31% of the time and account for 7.7% of all bytes transferred.

Table 5.54
Summary of Access Log Characteristics for boeing.990301 Trace

Trace start time:	Mon Mar 1 02:59:16 1999
Trace end time:	Tue Mar 2 02:58:54 1999
Total Requests:	248,228
Avg Requests/Minute:	172.42
Total Bytes Transferred:	8,776,017,746
Avg Bytes Transferred/Minute:	6,094,456.77

Table 5.55
Summary of Other Characteristics for boeing.990301 Trace

Maximum Size Item (bytes):	633,427,627
Minimum Size Item (bytes):	1
Unique Requests:	133,589
Unique Workload Size (bytes):	7,070,163,831
Distinct Requests/Total Requests (%):	53.82
Distinct Bytes/Total Bytes (%):	80.56
Second Requests/Distinct Requests (%):	15.54
Second Bytes/Distinct Bytes (%):	8.50
Distinct Files Accessed Only Once (%):	84.46
Distinct Bytes Accessed Only Once (%):	91.50

Table 5.56
Breakdown By File Size for boeing.990301 Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq$ 4kb	62.97	86.07
4kb $< s \leq$ 8kb	11.02	3.58
8kb $< s \leq$ 16kb	12.09	4.61
16kb $< s \leq$ 32kb	6.98	1.87
32kb $< s \leq$ 64kb	4.19	3.32
64kb $< s \leq$ 128kb	1.08	0.20
128kb $< s$	1.66	0.35

Table 5.57
Object Reaccess Information by File Size for boeing.990301 Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	84,127 (46.0 %)	15,986 (19.0 %)	82,687 (517.2 %)	182,800
4kb $< s \leq$ 8kb	14,726 (78.2 %)	1,475 (10.0 %)	2,626 (178.0 %)	18,827
8kb $< s \leq$ 16kb	16,152 (75.3 %)	1,758 (10.9 %)	3,530 (200.8 %)	21,440
16kb $< s \leq$ 32kb	9,326 (81.3 %)	737 (7.9 %)	1,412 (191.6 %)	11,475
32kb $< s \leq$ 64kb	5,594 (59.5 %)	516 (9.2 %)	3,287 (637.0 %)	9,397
64kb $< s \leq$ 128kb	1,443 (86.3 %)	119 (8.2 %)	110 (92.4 %)	1,672
128kb $< s$	2,221 (84.9 %)	168 (7.6 %)	228 (135.7 %)	2,617

Table 5.58
Breakdown By File Type for boeing.990301 Trace

File Type	**% of Requests**	**% of Bytes Transferred**
	1.01	1.77
(null)	0.00	0.00
*	0.01	0.01
-	13.96	0.14
application	9.00	30.73
audio	0.11	1.28
encoding	0.00	0.00
image	40.11	4.77
java	0.00	0.00
magnus-internal	0.00	0.00
multipart	0.01	0.02
text	35.77	60.35
video	0.02	0.92
www	0.00	0.00
x-world	0.00	0.00

Table 5.59
Multirequest Breakdown By File Type for boeing.990301 Trace

File Type	**% of Requests**	**% of Bytes Transferred**
	0.98	1.26
(null)	0.00	0.00
*	0.00	0.00
-	13.39	0.12
application	5.69	62.46
audio	0.03	0.01
encoding	0.00	0.00
image	43.83	4.90
java	0.00	0.00
magnus-internal	0.00	0.00
multipart	0.00	0.00
text	36.08	28.66
video	0.01	2.58
www	0.00	0.00
x-world	0.00	0.00

Table 5.60
Object Reaccess Information by File Type for boeing.990301 Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
	1,379 (55.2 %)	204 (14.8 %)	914 (448.0 %)	2,497
(null)	1 (100.0 %)	0 (0.0 %)	0 (0.0 %)	1
*	28 (100.0 %)	0 (0.0 %)	0 (0.0 %)	28
-	19,313 (55.7 %)	5,443 (28.2 %)	9,902 (181.9 %)	34,658
application	15,833 (70.8 %)	2,262 (14.3 %)	4,256 (188.2 %)	22,351
audio	238 (86.2 %)	19 (8.0 %)	19 (100.0 %)	276
encoding	1 (100.0 %)	0 (0.0 %)	0 (0.0 %)	1
image	49,314 (49.5 %)	6,207 (12.6 %)	44,036 (709.5 %)	99,557
java	1 (50.0 %)	0 (0.0 %)	1 (0.0 %)	2
magnus-internal	3 (75.0 %)	0 (0.0 %)	1 (0.0 %)	4
multipart	25 (92.6 %)	2 (8.0 %)	0 (0.0 %)	27
text	47,422 (53.4 %)	6,615 (13.9 %)	34,746 (525.3 %)	88,783
video	26 (68.4 %)	7 (26.9 %)	5 (71.4 %)	38
www	2 (100.0 %)	0 (0.0 %)	0 (0.0 %)	2
x-world	3 (100.0 %)	0 (0.0 %)	0 (0.0 %)	3

Table 5.61
Unique File Size Information by File Type for boeing.990301 Trace

file type	number	mean (bytes)	median (bytes)	maximum (bytes)	totalsize (bytes)
	1,379	96,809	2,429	15,585,280	133,499,048
(null)	1	1,332	1,332	1,332	1,332
*	28	27,393	36,211	85,425	767,011
-	19,313	545	121	4,803,156	10,530,052
all	133,589	52,925	2,458	633,427,627	7,070,163,831
application	15,833	103,047	3,547	34,210,686	1,631,547,323
audio	238	472,452	2,893	26,287,704	112,443,562
encoding	1	432	432	432	432
image	49,314	6,798	297	5,054,022	335,214,194
java	1	11,909	11,909	11,909	11,909
magnus-internal	3	239	219	331	718
multipart	25	66,049	4,305	754,169	1,651,215
text	47,422	101,382	24,961	633,427,627	4,807,750,858
video	26	1,412,112	617	28,659,255	36,714,917
www	2	4,369	3,153	5,585	8,738
x-world	3	7,507	15,545	15,545	22,522

Table 5.62
Stack Depth Analysis for boeing.990301 Trace

mean stack depth:	7,817.23
median stack depth:	3
standard deviation:	17,124.71
maximum stack depth:	132,213
normalized mean depth:	0.031,492
normalized median depth:	0.000,012

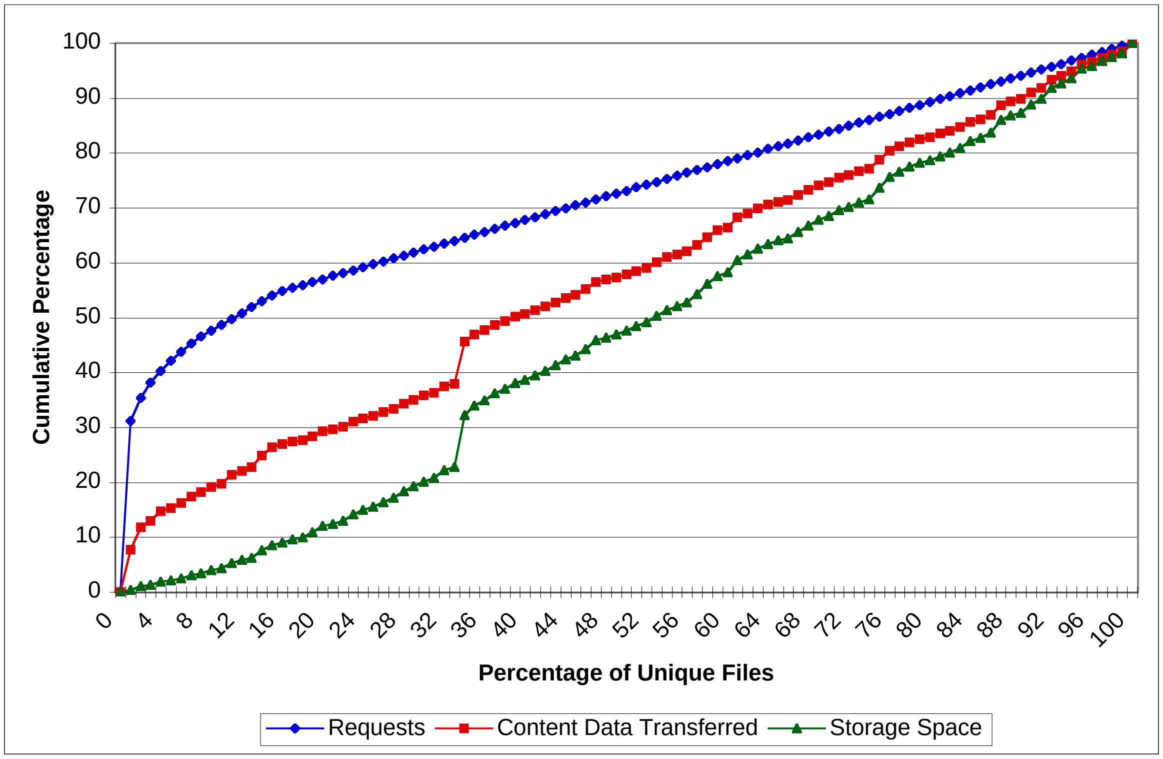


Fig. 5.6. Concentration of References for boeing.990301 Trace

5.4.5 NLANR Proxy Servers

The National Laboratory for Applied Network Research (NLANR) is a small U.S. Government agency that studies and maintains portions of the Internet backbone in the United States. The organization operates a number of backbone network Squid cache proxy servers on the very high performance Backbone Network Service (vBNS). These servers operate in Pittsburgh, Urbana-Champaign, Boulder, Silicon Valley, San Diego, and Palo Alto, and they are cooperative backbone servers which means that each of the servers specialize in one or more root domains like `.edu`, `.com`, `.gov`, etc. These servers share their content with other lower-tiered network cache proxy servers and with other backbone network cache proxy servers in other countries. Further, they also run a general network cache proxy server in San Jose which runs on the MAE-West network. NLANR posts sanitized logfiles from all of their proxy cache servers for seven days after they are collected. For this study, the traces of three proxy cache servers were taken over the course of three months: October, November, and December of 1999. The three proxies were from Urbana-Champaign, which is the primary cache of the `.com` domain; from one of the two Boulder proxies, which are the primary caches of the `.edu, .gov, .org, .mil` and `.us` domains; and from San Jose, which caches all of the backbone accesses for the Silicon Valley area.

From these three months of data, small sets of accesses were taken out for this study. Each of these sets is comprised of less than 500,000 requests. These sets provided typical loads and access patterns of these cache proxy servers without causing excessive computational resources to be used. (With these sets of less than 500,000 requests, the statistical data processing often took over six hours per set, while a simulation run for a set took as much as 240 hours (ten days) on a single processor Pentium3-Zeon machine.) Some of these sets are one day's worth of requests, while others contain three consecutive days of requests, and a few others contain seven days worth of data. For choosing a single day workload trace, a weekday trace was usually chosen since weekdays usually have more business traffic than the weekends. On multiple day traces, a Sunday-to-Monday transition was always included because the weekend traffic is generally different than weekday traffic in terms of the business traffic content. By having a Sunday-to-Monday transition, the cache proxy is challenged and exercised more than on any other day-to-day transition.

Below, the statistics of one of each of the proxy servers' sets will be presented. These sets are then also the basis for comparison of the cache replacement algorithms that will be presented in Chapter 6.

The National Laboratory for Applied Network Research is supported by the National Science Foundation on grants NCR-9616602 and NCR-9521745. The National Laboratory for Applied Network Research requires mentioning this funding when their logfiles are used for research or commercial purposes.

Boulder1 Proxy Logfile Traces

As mentioned in the above section, the Boulder bo1 cache proxy server is the primary root backbone network cache for the `.edu`, `.gov`, `.org`, `.mil` and `.us` domains. The logfile statistics in this section are from a 24 hour period on Wednesday, December 15, 1999. During this time, the bo1 server had 325,969 total requests which resulted in 3.2 gigabytes of data transferred as shown in Table 5.63. This translates into over 226 requests per minute and over 2.2 megabytes transferred per minute. Table 5.64 shows that over 75% of all accesses are first time accessed (compulsory misses), and 90% of all of the objects are only accessed once. Also, less than 10% of all unique objects are accessed a second time. In Tables 5.67 and 5.68, it is shown that most of the requests and requests past the first are for file types "image" and "html". The percentages of requests for file type "dynamic" is very interesting. The logfiles show that these files are actually cachable because they are mostly graphic advertising banners that were originally generated by a cgi script. Since the advertising banners had unique, reproducible URLs, they were often forwarded by the bo1 cache. It is also interesting that a significant percentage of transferred data is of type "application". Looking at the logfile, many of these "application" files are of subtype "cache-digest" which is a database file that the bo1 cache transmits to subordinate caches to share what files its cache currently contains. These “cache-digest” files are often as much as 800 kilobytes in size. The normalized mean stack depth in Table 5.71 is relatively large as would be expected from a network proxy cache. However, the normalized median stack depth is surprisingly small; this trace had good temporal locality. In the Concentration of References graph in Figure 5.7, the most popular 1% of the objects were 15.3% of all access requests and constituted 28.1% of all bytes transferred. These numbers are low but are expected for network proxy caches.

Table 5.63
Summary of Access Log Characteristics for bo1_day1.dat Trace

Trace start time:	Wed Dec 15 03:00:20 1999
Trace end time:	Thu Dec 16 03:00:00 1999
Total Requests:	325,969
Avg Requests/Minute:	226.42
Total Bytes Transferred:	3,275,168,537
Avg Bytes Transferred/Minute:	2,274,422.60

Table 5.64
Summary of Other Characteristics for bo1_day1.dat Trace

Maximum Size Item (bytes):	21,473,658
Minimum Size Item (bytes):	17
Unique Requests:	247,563
Unique Workload Size (bytes):	2,222,586,444
Distinct Requests/Total Requests (%):	75.95
Distinct Bytes/Total Bytes (%):	67.86
Second Requests/Distinct Requests (%):	9.34
Second Bytes/Distinct Bytes (%):	5.14
Distinct Files Accessed Only Once (%):	90.66
Distinct Bytes Accessed Only Once (%):	94.86

Table 5.65
Breakdown By File Size for bo1_day1.dat Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq$ 4kb	73.09	81.26
4kb $< s \leq$ 8kb	10.28	10.42
8kb $< s \leq$ 16kb	8.84	4.19
16kb $< s \leq$ 32kb	4.56	1.46
32kb $< s \leq$ 64kb	2.21	0.81
64kb $< s \leq$ 128kb	0.55	0.12
128kb $< s$	0.47	1.73

Table 5.66
Object Reaccess Information by File Size for bo1_day1.dat Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	180,951 (74.0 %)	18,309 (10.1 %)	45,406 (248.0 %)	244,666
4kb $< s \leq$ 8kb	25,445 (75.7 %)	2,280 (9.0 %)	5,892 (258.4 %)	33,617
8kb $< s \leq$ 16kb	21,892 (86.9 %)	1,659 (7.6 %)	1,629 (98.2 %)	25,180
16kb $< s \leq$ 32kb	11,277 (90.8 %)	403 (3.6 %)	739 (183.4 %)	12,419
32kb $< s \leq$ 64kb	5,469 (89.6 %)	291 (5.3 %)	346 (118.9 %)	6,106
64kb $< s \leq$ 128kb	1,358 (93.3 %)	44 (3.2 %)	53 (120.5 %)	1,455
128kb $< s$	1,171 (46.4 %)	131 (11.2 %)	1,224 (934.4 %)	2,526

Table 5.67
Breakdown By File Type for bo1_day1.dat Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	1.53	36.92
audio	0.19	1.88
compressed	0.14	7.85
dynamic	6.46	5.20
html	23.40	15.59
image	65.34	25.49
java	0.22	0.28
other	1.29	2.91
text	1.36	1.30
video	0.07	2.57

Table 5.68
Multirequest Breakdown By File Type for bo1_day1.dat Trace

File Type	% of Requests	% of Bytes Transferred
application	2.63	75.90
audio	0.05	0.05
compressed	0.06	0.07
dynamic	17.90	5.50
html	30.22	7.80
image	43.97	5.97
java	0.12	0.05
other	2.26	4.43
text	2.79	0.20
video	0.02	0.03

Table 5.69
Object Reaccess Information by File Type for bo1_day1.dat Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	2,934 (58.7 %)	316 (10.8 %)	1,746 (552.5 %)	4,996
audio	590 (94.2 %)	28 (4.7 %)	8 (28.6 %)	626
compressed	419 (89.7 %)	27 (6.4 %)	21 (77.8 %)	467
dynamic	7,029 (33.4 %)	2,605 (37.1 %)	11,426 (438.6 %)	21,060
html	52,590 (68.9 %)	5,934 (11.3 %)	17,759 (299.3 %)	76,283
image	178,513 (83.8 %)	13,382 (7.5 %)	21,093 (157.6 %)	212,988
java	618 (87.0 %)	37 (6.0 %)	55 (148.6 %)	710
other	2,361 (57.2 %)	430 (18.2 %)	1,335 (310.5 %)	4,126
text	2,250 (50.7 %)	344 (15.3 %)	1,840 (534.9 %)	4,434
video	205 (94.5 %)	11 (5.4 %)	1 (9.1 %)	217

Table 5.70
Unique File Size Information by File Type for bo1_day1.dat Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	247,563	8,978	256	21,473,658	2,222,586,444
application	2,934	139,693	274	21,473,658	409,860,148
audio	590	103,579	25,541	5,007,124	61,111,796
compressed	419	611,861	2,916,749	14,409,474	256,369,947
dynamic	7,029	16,006	494	11,632,510	112,508,239
html	52,590	8,149	388	1,402,982	428,568,999
image	178,513	4,326	2,524	706,244	772,181,112
java	618	13,948	250	962,464	8,619,964
other	2,361	19,283	15,292	8,098,157	45,525,985
text	2,250	17,990	345	4,279,419	40,477,766
video	205	408,891	35,385	7,675,780	83,822,627

Table 5.71
Stack Depth Analysis for bo1_day1.dat Trace

mean stack depth:	16,561.94
median stack depth:	3
standard deviation:	35,998.11
maximum stack depth:	245,924
normalized mean depth:	0.050,808
normalized median depth:	0.000,038

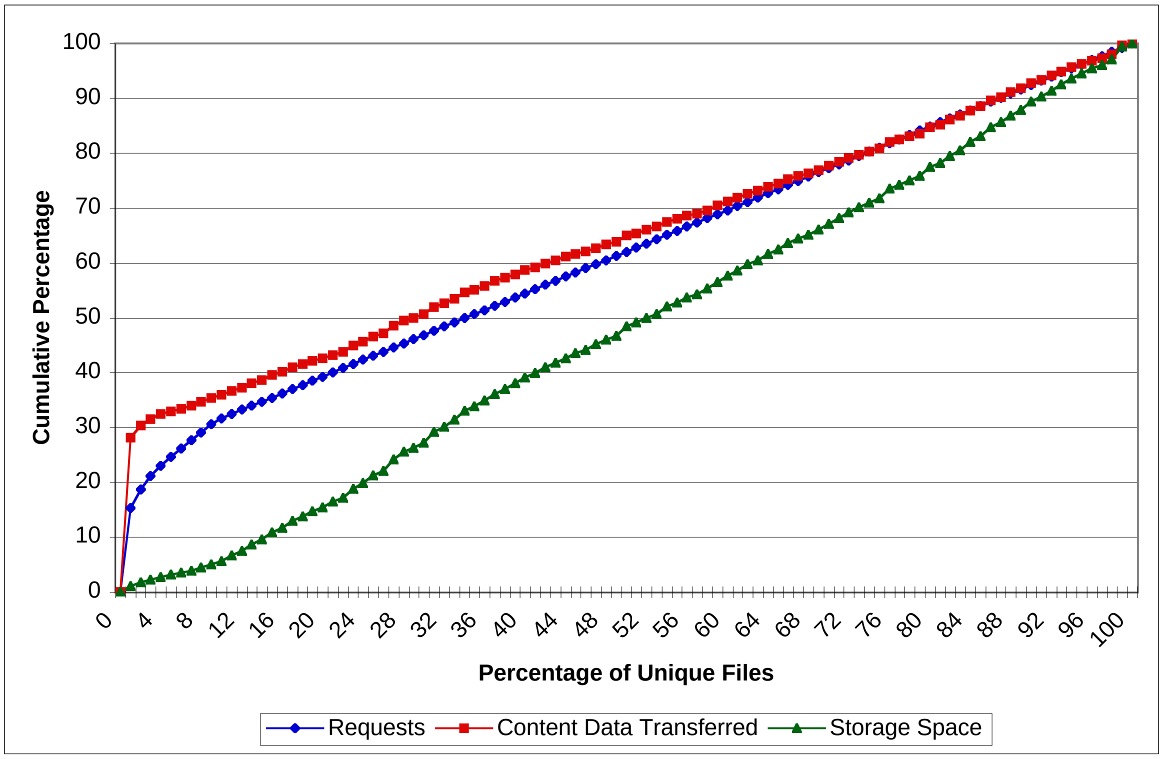


Fig. 5.7. Concentration of References for bo1_day1.dat Trace

San Jose Proxy Logfile Traces

The San Jose cache proxy server is a network proxy server that is located in San Jose, California situated right off of the MAE-West network. It caches all Internet traffic that is requested in the San Jose area over the MAE-West network. The statistics of a San Jose workload trace from Saturday, November 27, 1999 to Friday, December 3, 1999 are presented here. Over this week, the San Jose proxy cache server received 343,468 requests totaling 4.3 gigabytes of transferred data. This translates to 34.1 requests per minute and over 384 kilobytes per minutes. This is shown in Table 5.72. Over 77% of all requests are first time requests (compulsory misses), while 93% of all unique files are accessed only once as is shown in Table 5.73. These percentages should be expected for a network proxy server.

In Tables 5.76 and 5.77, one can see that most of the requests and rerequests are for file types "image" and "html". Note that the "application" type constitutes a fairly significant percentage of data transferred for the first time and for requests after the first. These "application" type objects are some cache-digests as in the Boulder trace which are database files that the San Jose cache proxy server sends to its client cache proxy servers to share what items it has in its cache. Also note that almost 7% of the data transferred for first-time requests in Table 5.76 are for file type "video". However, these videos are not rerequested since they don't show up in Table 5.77. In the stack depth analysis of Table 5.80, the normalized mean stack depth is rather high, while the normalized median stack depth is low. As with the Boulder trace discussed before, this trace has good temporal locality based on the normalized median stack depth. Finally in Figure 5.8, the most popular 1% of the objects were 16.7% of all access requests and constituted 6.1% of all bytes transferred. These numbers are low but are expected for network proxy caches.

Table 5.72
Summary of Access Log Characteristics for sj_novwk5.dat Trace

Trace start time:	Sat Nov 27 03:00:18 1999
Trace end time:	Sat Dec 4 03:00:00 1999
Total Requests:	329,222
Avg Requests/Minute:	32.66
Total Bytes Transferred:	3,871,038,504
Avg Bytes Transferred/Minute:	384,031.60

Table 5.73
Summary of Other Characteristics for sj_novwk5.dat Trace

Maximum Size Item (bytes):	55,647,563
Minimum Size Item (bytes):	88
Unique Requests:	253,928
Unique Workload Size (bytes):	3,495,355,673
Distinct Requests/Total Requests (%):	77.13
Distinct Bytes/Total Bytes (%):	90.30
Second Requests/Distinct Requests (%):	6.96
Second Bytes/Distinct Bytes (%):	3.25
Distinct Files Accessed Only Once (%):	93.04
Distinct Bytes Accessed Only Once (%):	96.75

Table 5.74
Breakdown By File Size for sj_novwk5.dat Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq 4$kb	64.67	85.66
4kb $< s \leq 8$kb	12.56	5.22
8kb $< s \leq 16$kb	10.55	6.04
16kb $< s \leq 32$kb	5.78	1.92
32kb $< s \leq 64$kb	2.97	0.77
64kb $< s \leq 128$kb	2.33	0.14
128kb $< s$	1.13	0.25

Table 5.75
Object Reaccess Information by File Size for sj_novwk5.dat Trace

File Type	First Accesses	Second Accesses	Three+ Accesses	Total Accesses
$s \leq 4$kb	164,222 (71.8 %)	14,133 (8.6 %)	50,361 (356.3 %)	228,716
4kb $< s \leq$ 8kb	31,903 (89.0 %)	1,172 (3.7 %)	2,762 (235.7 %)	35,837
8kb $< s \leq$ 16kb	26,801 (85.5 %)	1,362 (5.1 %)	3,186 (233.9 %)	31,349
16kb $< s \leq$ 32kb	14,667 (91.0 %)	675 (4.6 %)	769 (113.9 %)	16,111
32kb $< s \leq$ 64kb	7,543 (92.9 %)	257 (3.4 %)	322 (125.3 %)	8,122
64kb $< s \leq$ 128kb	5,926 (98.3 %)	62 (1.0 %)	43 (69.4 %)	6,031
128kb $< s$	2,866 (93.8 %)	20 (0.7 %)	170 (850.0 %)	3,056

Table 5.76
Breakdown By File Type for sj_novwk5.dat Trace

File Type	% of Requests	% of Bytes Transferred
application	0.88	17.84
audio	0.23	1.90
compressed	0.23	6.27
dynamic	1.10	0.18
html	23.39	13.40
image	70.87	49.68
java	0.24	0.15
magnus-internal	0.00	0.00
model	0.01	0.01
multipart	0.00	0.02
other	2.09	1.07
text	0.89	1.95
video	0.07	7.51
x-shockwave-flash	0.00	0.01
x-world	0.00	0.00

Table 5.77
Multirequest Breakdown By File Type for sj_novwk5.dat Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	1.49	47.47
audio	0.08	0.42
compressed	0.37	1.54
dynamic	1.69	0.42
html	30.09	23.65
image	61.58	23.47
java	0.09	0.04
magnus-internal	0.00	0.00
model	0.00	0.00
multipart	0.00	0.00
other	2.77	2.29
text	1.84	0.70
video	0.00	0.00
x-shockwave-flash	0.00	0.00
x-world	0.00	0.00

Table 5.78
Object Reaccess Information by File Type for sj_novwk5.dat Trace

File Type	**First Accesses**	**Second Accesses**	**Other Accesses**	**Total Accesses**
application	1,761 (61.1 %)	217 (12.3 %)	906 (417.5 %)	2,884
audio	704 (91.9 %)	38 (5.4 %)	24 (63.2 %)	766
compressed	472 (62.8 %)	65 (13.8 %)	214 (329.2 %)	751
dynamic	2,348 (64.8 %)	345 (14.7 %)	929 (269.3 %)	3,622
html	54,365 (70.6 %)	6,254 (11.5 %)	16,402 (262.3 %)	77,021
image	186,964 (80.1 %)	9,774 (5.2 %)	36,590 (374.4 %)	233,328
java	741 (92.0 %)	55 (7.4 %)	9 (16.4 %)	805
magnus-internal	3 (100.0 %)	0 (0.0 %)	0 (0.0 %)	3
model	26 (100.0 %)	0 (0.0 %)	0 (0.0 %)	26
multipart	4 (100.0 %)	0 (0.0 %)	0 (0.0 %)	4
other	4,781 (69.6 %)	769 (16.1 %)	1,315 (171.0 %)	6,865
text	1,530 (52.5 %)	163 (10.7 %)	1,224 (750.9 %)	2,917
video	219 (99.5 %)	1 (0.5 %)	0 (0.0 %)	220
x-shockwave-flash	1 (100.0 %)	0 (0.0 %)	0 (0.0 %)	1
x-world	2 (100.0 %)	0 (0.0 %)	0 (0.0 %)	2

Table 5.79
Unique File Size Information by File Type for sj_novwk5.dat Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**max (bytes)**	**totalsize (bytes)**
all	253,928	13,765	270	55,647,563	3,495,355,673
application	1,761	290,981	2,806	55,647,563	512,418,309
audio	704	101,989	261	8,238,947	71,800,257
compressed	472	502,236	3,680,312	9,171,722	237,055,307
dynamic	2,348	2,377	376	463,762	5,581,845
html	54,365	7,910	3,823	9,360,068	430,023,691
image	186,964	9,815	264	2,337,129	1,835,041,166
java	741	7,580	251	877,025	5,617,024
magnus-internal	3	415	415	415	1,244
model	26	15,769	20,559	37,293	410,004
multipart	4	153,908	300,610	300,610	615,632
other	4,781	6,891	1,296	1,207,593	32,946,730
text	1,530	47,546	254	8,374,803	72,745,789
video	219	1,327,012	2,403,660	15,866,696	290,615,622
x-shockwave-flash	1	427,953	427,953	427,953	427,953
x-world	2	11,202	12,853	12,853	22,404

Table 5.80
Stack Depth Analysis for sj_novwk5.dat Trace

mean stack depth:	13,597.9
median stack depth:	30
standard deviation:	32,566.1
maximum stack depth:	250,569
normalized mean depth:	0.041,303
normalized median depth:	0.000,091

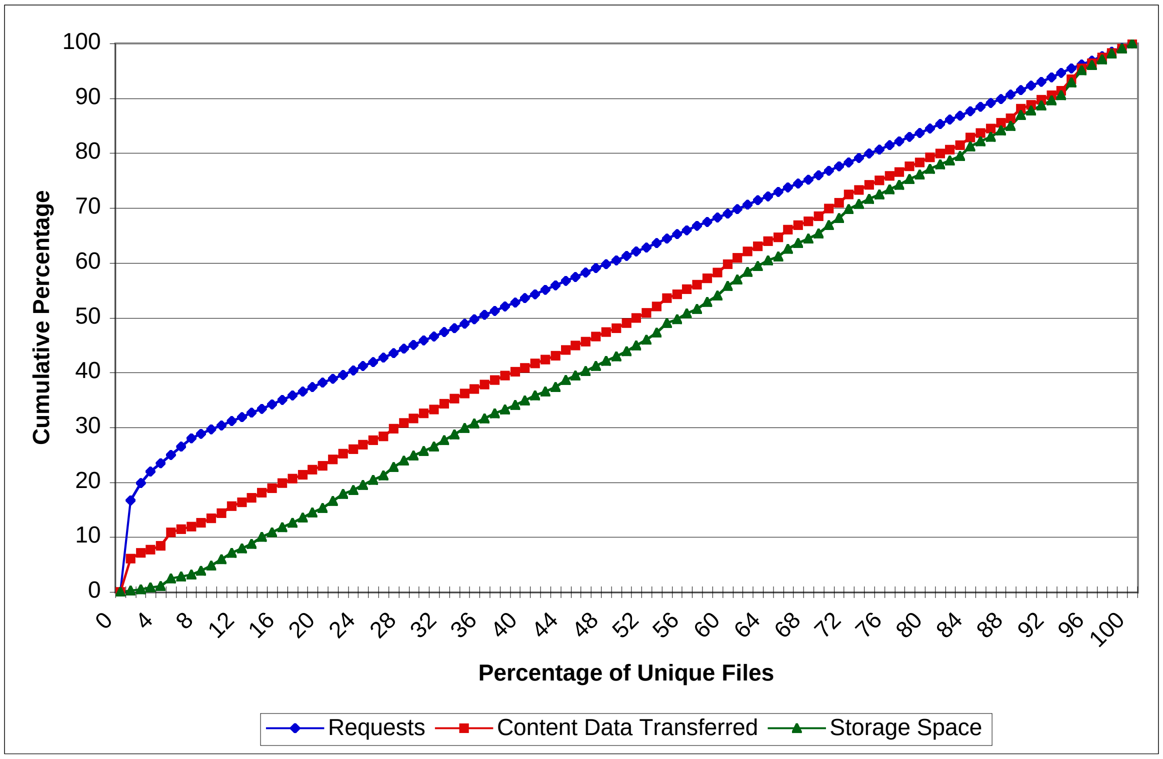


Fig. 5.8. Concentration of References for sj_novwk5.dat Trace

Urbana-Champaign Logfile Traces

The third NLANR cache proxy server from which data has been collected is the Urbana-Champaign (UC) server. It is the primary root backbone network cache for the `.com` domain. The logfile statistics in this section are from a 24 hour period on Saturday, December 18, 1999. In these 24 hours, the UC server received 312,465 requests which totaled 5.6 gigabytes of transferred data. In terms of server loading, the UC server averaged 217 requests per minute and transferred nearly 3.9 megabytes of data per minute as shown in Table 5.81. Table 5.82 shows that almost 81% of all requests were first-time requests (compulsory misses), and 93% of all unique objects were accessed only once. Again, these percentages should be expected for a network proxy server.

In Tables 5.85 and 5.86, one can see that most of the requests and rerequests are for file types “image” and “html”. As in the Boulder and San Jose workload traces, note that the “application” type constitutes a fairly significant percentage of data transferred for the first time and for requests after the first. These applications are some cache-digests as in the Boulder and San Jose traces which are database files that the Urbana Champaign cache proxy server sends to its client cache proxy servers to share what items it has in its cache. The normalized statistics in the stack depth analysis in Table 5.89 are quite similar to those in the Boulder and San Jose traces; the normalized mean stack depth is rather high, while the normalized median stack depth is low. The low normalized median stack depth suggests that this trace has good temporal locality. The Concentration of References graph in Figure 5.9 is also similar to the Boulder and San Jose traces. The most popular 1% of the objects were 13.8% of all access requests and constituted 16.3% of all bytes transferred. These numbers are low but are expected for network proxy caches.

Table 5.81
Summary of Access Log Characteristics for uc_day1.dat Trace

Trace start time:	Sat Dec 18 03:00:18 1999
Trace end time:	Sun Dec 19 03:00:01 1999
Total Requests:	312,465
Avg Requests/Minute:	217.03
Total Bytes Transferred:	5,593,480,281
Avg Bytes Transferred/Minute:	3,885,125.74

Table 5.82
Summary of Other Characteristics for uc_day1.dat Trace

Maximum Size Item (bytes):	55,659,143
Minimum Size Item (bytes):	97
Unique Requests:	253,376
Unique Workload Size (bytes):	4,641,571,917
Distinct Requests/Total Requests (%):	81.09
Distinct Bytes/Total Bytes (%):	82.98
Second Requests/Distinct Requests (%):	6.79
Second Bytes/Distinct Bytes (%):	2.16
Distinct Files Accessed Only Once (%):	93.21
Distinct Bytes Accessed Only Once (%):	97.84

Table 5.83
Breakdown By File Size for uc_day1.dat Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq 4$kb	70.36	72.97
4kb $< s \leq$ 8kb	11.08	17.77
8kb $< s \leq$ 16kb	8.48	4.37
16kb $< s \leq$ 32kb	4.86	1.84
32kb $< s \leq$ 64kb	2.72	0.84
64kb $< s \leq$ 128kb	1.13	0.13
128kb $< s$	1.37	2.08

Table 5.84
Object Reaccess Information by File Size for uc_day1.dat Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq$ 4kb	178,269 (80.5 %)	13,734 (7.7 %)	29,381 (213.9 %)	221,384
4kb $< s \leq$ 8kb	28,074 (72.8 %)	1,381 (4.9 %)	9,121 (660.5 %)	38,576
8kb $< s \leq$ 16kb	21,493 (89.3 %)	1,085 (5.0 %)	1,495 (137.8 %)	24,073
16kb $< s \leq$ 32kb	12,309 (91.9 %)	554 (4.5 %)	531 (95.8 %)	13,394
32kb $< s \leq$ 64kb	6,899 (93.3 %)	273 (4.0 %)	223 (81.7 %)	7,395
64kb $< s \leq$ 128kb	2,860 (97.3 %)	65 (2.3 %)	14 (21.5 %)	2,939
128kb $< s$	3,472 (73.8 %)	108 (3.1 %)	1,124 (1040.7 %)	4,704

Table 5.85
Breakdown By File Type for uc_day1.dat Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	1.53	31.12
audio	0.18	1.24
compressed	0.43	17.42
dynamic	0.80	0.10
html	22.03	8.02
image	71.50	36.47
java	0.22	0.17
other	1.59	1.45
text	1.65	2.26
video	0.05	1.74

Table 5.86
Multirequest Breakdown By File Type for uc_day1.dat Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	2.93	76.85
audio	0.04	0.00
compressed	0.14	0.11
dynamic	2.63	0.16
html	44.07	11.16
image	41.34	5.72
java	0.12	0.01
other	5.25	5.72
text	3.47	0.27
video	0.01	0.00

Table 5.87
Object Reaccess Information by File Type for uc_day1.dat Trace

File Type	**First Accesses**	**Second Accesses**	**Other Accesses**	**Total Accesses**
application	3,061 (63.9 %)	324 (10.6 %)	1,407 (434.3 %)	4,792
audio	526 (96.0 %)	17 (3.2 %)	5 (29.4 %)	548
compressed	1,255 (93.7 %)	34 (2.7 %)	50 (147.1 %)	1,339
dynamic	944 (37.8 %)	282 (29.9 %)	1,270 (450.4 %)	2,496
html	42,800 (62.2 %)	5,329 (12.5 %)	20,713 (388.7 %)	68,842
image	198,970 (89.1 %)	10,451 (5.3 %)	13,976 (133.7 %)	223,397
java	627 (90.0 %)	48 (7.7 %)	22 (45.8 %)	697
other	1,854 (37.5 %)	319 (17.2 %)	2,777 (870.5 %)	4,950
text	3,111 (60.2 %)	386 (12.4 %)	1,667 (431.9 %)	5,164
video	165 (96.5 %)	6 (3.6 %)	0 (0.0 %)	171

Table 5.88
Unique File Size Information by File Type for uc_day1.dat Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	253,376	18,319	254	55,659,143	4,641,571,917
application	3,061	329,659	3,005	55,659,143	1,009,085,514
audio	526	131,274	256,331	9,432,994	69,050,191
compressed	1,255	775,694	226,333	29,342,094	973,495,666
dynamic	944	4,233	341	374,877	3,995,615
html	42,800	8,000	3,797	1,594,877	342,393,756
image	198,970	9,980	253	10,804,620	1,985,790,813
java	627	15,120	2,512	4,478,779	9,480,022
other	1,854	13,371	230	2,102,374	24,789,961
text	3,111	39,850	350	3,174,409	123,973,981
video	165	590,897	254	8,360,090	97,498,082

Table 5.89
Stack Depth Analysis for uc_day1.dat Trace

mean stack depth:	14,466.71
median stack depth:	295
standard deviation:	35,153.37
maximum stack depth:	251,073
normalized mean depth:	0.046,299
normalized median depth:	0.000,944

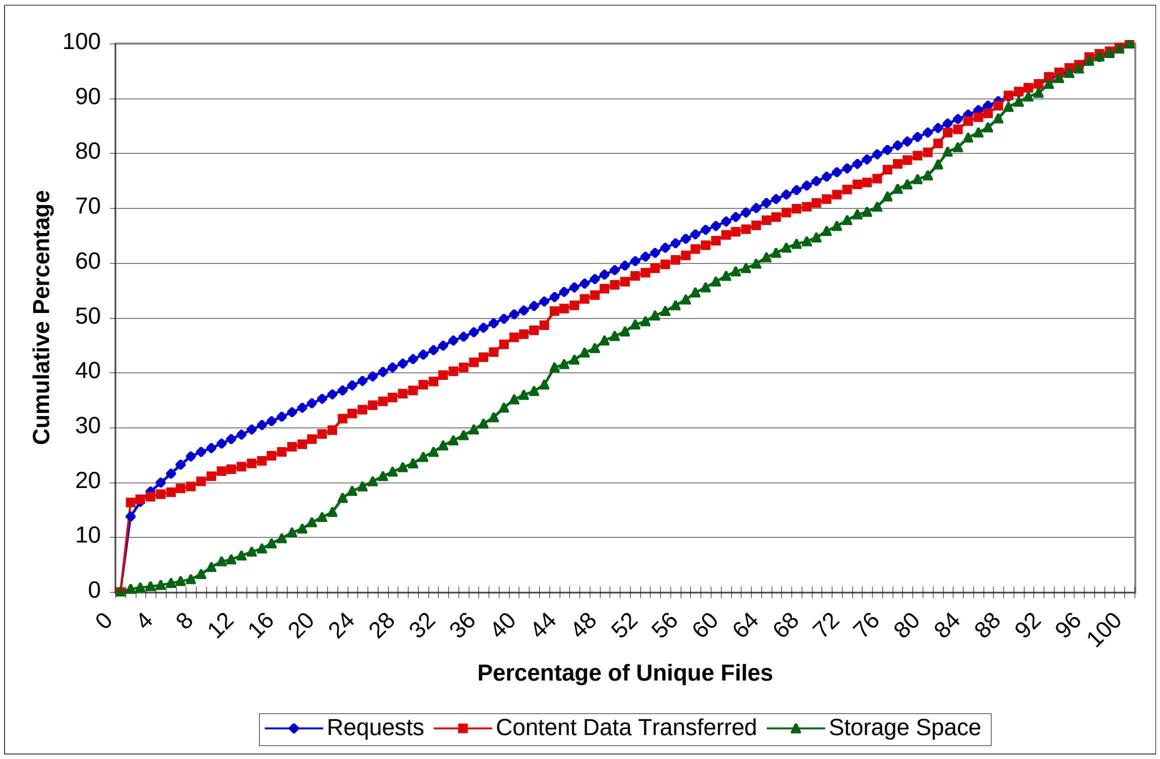


Fig. 5.9. Concentration of References for uc_day1.dat Trace

5.4.6 HP World Cup 1998 Web Server Farm

In 1998, Hewlett-Packard hosted the Web site for the 1998 World Cup Soccer Championship. World Cup fans could access game scores in real time as well as previous game results, player statistics and biographies, team histories, stadium information and many photos and sound clips. They used 30 machines to host the entire site in both French and English. Of the 30 servers, four were located in Paris, France; 10 were located in Herndon, Virginia; 10 were located in Plano, Texas; and six were located in Santa Clara, California. Accesses from all of these servers were combined and collated by date and time. So this data set is an access pattern for a origin server proxy cache. A full description of the entire 14 week logfile is in [AJ99].

The access log presented here and used in this study is for the last two days that the World Cup 1998 site was up: Saturday, July 25, 1998 and Sunday, July 26, 1998. The number of accesses per day made it virtually impossible to thoroughly analyze any of the other 13 weeks of workload data that was collected. This logfile was the only one which had less than one million accesses for the week, and that was only because this last week file only contained one 24 hour period.

In this 24 hour period, there were 876,530 requests totalling 5.7 gigabytes of transferred data. That translated to over 608 requests per minute on average and over 4.0 megabytes of data transferred per minute across the 30 servers. In Table 5.91, the first time request percentage (distinct requests/total requests) is very low at 1.61%, as is the distinct files accessed only once at 26.21%. Tables 5.94 and 5.95 show that the overwhelming majority of requests were for files of type "image" with a distant second most requested file type being "html". The file type "compressed" constituted a significant percentage of the total bytes transferred though they were not a significant percentage of the total requests. Most of the files of type "compressed" were compressed screen saver files for PCs and Macintoshes. Many of these screen saver files were larger than one megabyte in size which provides an explanation for the low request percentage while still being a significant portion of the bytes transferred.

As expected for a origin server proxy cache, the normalized mean and median stack depths are very small as shown in Table 5.10. Also the Concentration of References graph in Figure 5.10 shows that the most popular 1% of the objects that were requested constituted 55.4% of all requests and 30.2% of all data bytes transferred.

Table 5.90
Summary of Access Log Characteristics for wc_week14 Trace

Trace start time:	Sat Jul 25 21:59:51 1998
Trace end time:	Sun Jul 26 21:59:55 1998
Total Requests:	876,530
Avg Requests/Minute:	608.67
Total Bytes Transferred:	5,655,973,870
Avg Bytes Transferred/Minute:	3,927,759.63

Table 5.91
Summary of Other Characteristics for wc_week14 Trace

Maximum Size Item (bytes):	2,891,887
Minimum Size Item (bytes):	2
Unique Requests:	14,111
Unique Workload Size (bytes):	224,184,681
Distinct Requests/Total Requests (%):	1.61
Distinct Bytes/Total Bytes (%):	3.96
Second Requests/Distinct Requests (%):	73.79
Second Bytes/Distinct Bytes (%):	54.08
Distinct Files Accessed Only Once (%):	26.21
Distinct Bytes Accessed Only Once (%):	45.92

Table 5.92
Breakdown By File Size for wc_week14 Trace

Size Range	**Unique Files (%)**	**Multirequests (%)**
$s \leq 4$kb	38.93	71.71
4kb $< s \leq$ 8kb	34.55	12.38
8kb $< s \leq$ 16kb	14.78	9.06
16kb $< s \leq$ 32kb	6.53	4.91
32kb $< s \leq$ 64kb	4.22	1.61
64kb $< s \leq$ 128kb	0.23	0.11
128kb $< s$	0.76	0.21

Table 5.93
Object Reaccess Information by File Size for wc_week14 Trace

File Type	**First Accesses**	**Second Accesses**	**Three+ Accesses**	**Total Accesses**
$s \leq 4$kb	5,493 (0.9 %)	3,785 (68.9 %)	614,698 (16240.4 %)	623,976
4kb $< s \leq$ 8kb	4,876 (4.4 %)	3,663 (75.1 %)	103,139 (2815.7 %)	111,678
8kb $< s \leq$ 16kb	2,085 (2.6 %)	1,581 (75.8 %)	76,569 (4843.1 %)	80,235
16kb $< s \leq$ 32kb	922 (2.1 %)	795 (86.2 %)	41,555 (5227.0 %)	43,272
32kb $< s \leq$ 64kb	595 (4.1 %)	536 (90.1 %)	13,335 (2487.9 %)	14,466
64kb $< s \leq$ 128kb	33 (3.3 %)	23 (69.7 %)	944 (4104.3 %)	1,000
128kb $< s$	107 (5.6 %)	29 (27.1 %)	1,767 (6093.1 %)	1,903

Table 5.94
Breakdown By File Type for wc_week14 Trace

File Type	**% of Requests**	**% of Bytes Transferred**
application	0.75	1.59
audio	0.02	0.15
compressed	0.20	29.72
dynamic	0.01	0.00
html	12.28	23.79
image	83.21	41.92
java	1.04	0.67
other	2.45	1.79
text	0.05	0.00
video	0.00	0.37

Table 5.95
Multirequest Breakdown By File Type for wc_week14 Trace

File Type	% of Requests	% of Bytes Transferred
application	0.75	1.62
audio	0.01	0.00
compressed	0.19	29.51
dynamic	0.00	0.00
html	11.64	23.39
image	83.86	42.83
java	1.06	0.70
other	2.43	1.86
text	0.05	0.00
video	0.00	0.09

Table 5.96
Object Reaccess Information by File Type for wc_week14 Trace

File Type	First Accesses	Second Accesses	Other Accesses	Total Accesses
application	43 (0.7 %)	22 (51.2 %)	6,484 (29472.7 %)	6,549
audio	49 (31.4 %)	39 (79.6 %)	68 (174.4 %)	156
compressed	88 (5.1 %)	22 (25.0 %)	1,601 (7277.3 %)	1,711
dynamic	18 (40.0 %)	13 (72.2 %)	14 (107.7 %)	45
html	7,291 (6.8 %)	5,474 (75.1 %)	94,899 (1733.6 %)	107,664
image	6,140 (0.8 %)	4,589 (74.7 %)	718,639 (15660.0 %)	729,368
java	27 (0.3 %)	12 (44.4 %)	9,119 (75991.7 %)	9,158
other	441 (2.1 %)	238 (54.0 %)	20,753 (8719.7 %)	21,432
text	3 (0.7 %)	1 (33.3 %)	429 (42900.0 %)	433
video	11 (78.6 %)	2 (18.2 %)	1 (50.0 %)	14

Table 5.97
Unique File Size Information by File Type for wc_week14 Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	14,111	15,887	4,055	2,891,887	224,184,681
application	43	37,336	5,450	654,490	1,605,453
audio	49	168,513	134	1,396,314	8,257,132
compressed	88	889,513	2,082,593	2,891,887	78,277,130
dynamic	18	306	207	892	5,504
html	7,291	10,317	4,604	159,258	75,224,423
image	6,140	7,298	3,390	1,380,876	44,812,648
java	27	1,581	207	6,507	42,681
other	441	431	207	4,930	190,050
text	3	280	311	322	840
video	11	1,433,529	1,367,199	1,946,584	15,768,820

Table 5.98
Stack Depth Analysis for wc_week14 Trace

mean stack depth:	771.07
median stack depth:	3,051
standard deviation:	1,529.25
maximum stack depth:	14,041
normalized mean depth:	0.000,880
normalized median depth:	0.003,481

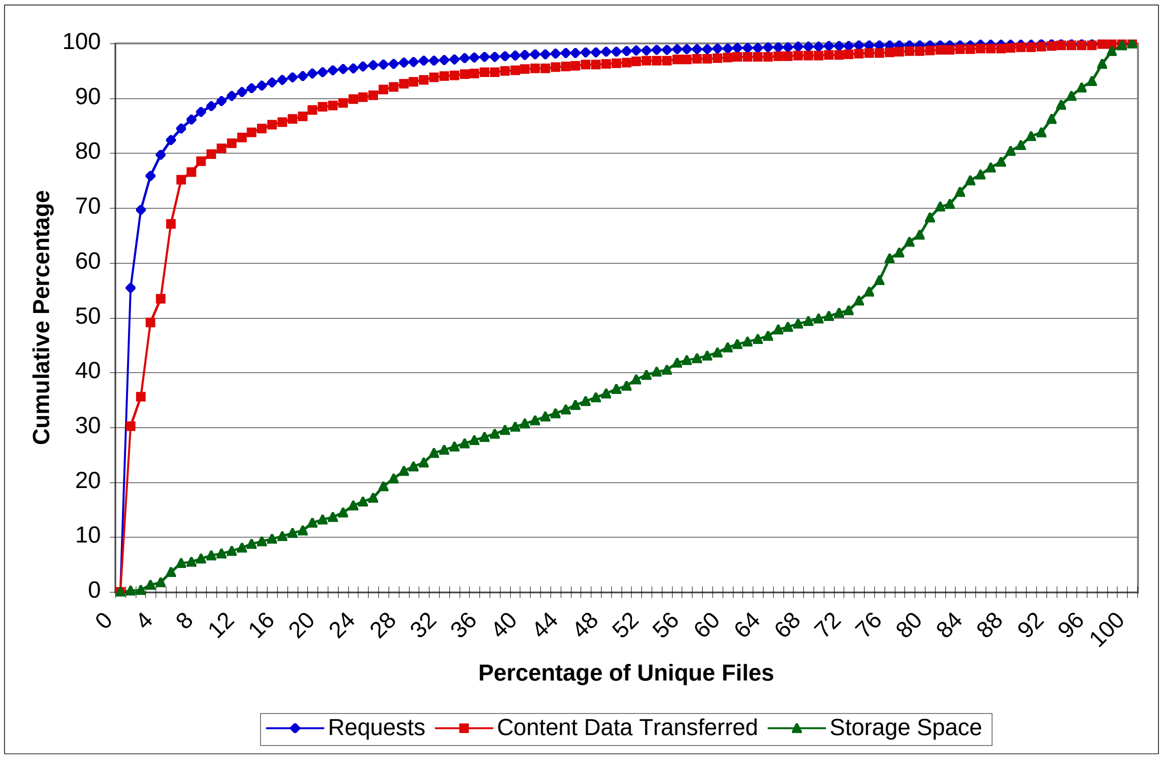


Fig. 5.10. Concentration of References for wc_week14 Trace

5.5 DVJ2 Educational Multimedia Video Server

Over the past five years, educational multimedia delivery testbed systems have been in operation in the Digital Systems/Multimedia Learning Lab in the School of Electrical and Computer Engineering at Purdue University. These systems are used to augment course lectures, to study how students use educational multimedia, and to study technology-based course formats. The analog video jockey (AVJ), digital video jockey (DVJ), and digital video jockey version 2 (DVJ2) systems are "a multi-user, . . . , interactive multimedia delivery system" [Bho95]. The AVJ system was a digitally-controlled analog video delivery system, and it is thoroughly described in [Bho95]. The DVJ and DVJ2 systems are both fully digital educational multimedia delivery systems that serve digital video lectures and class support material over an Ethernet local area network and multimedia personal computers [Ban95]. The DVJ system was thoroughly described in [Ban95]. Several papers, [MK94, MBN96, MNR97], describe further development and utilization of the AVJ and DVJ systems at Purdue University, as well as experimental course formats designed around using these systems. The current version of the system is the DVJ2 system which grew out of the original DVJ system implementation. It is also a fully digital system that serves lectures, help session, and tutorials for several classes offered in the the School of Electrical and Computer Engineering at Purdue University including EE 266/7 *Digital Logic Design*, EE 362 *Microprocessor System and Interfacing*, and EE 467 *Advanced Digital Systems/Embedded Microcontroller Design Lab*. The videos are all MPEG videos and are served by StarWorks video server software over 100Base-T Ethernet networks. There are about 60 client workstations at which students can view the videos. Students with sufficiently high bandwidth connections (e.g., residence hall network) can also stream the videos to their own computers. For all of these systems, all of the actions of users are recorded while they are interacting with the systems. Software was written to compile the usage characteristics of the users, and further software was written to create input files for this study.

From these testbed systems, we have many semesters of student usage data [RM97].

For this study the data files from the DVJ2 system are used. The tables and graphs presented in this section are from the Fall 1999 semester. The trace records accesses from Monday, August 23, 1999 to Thursday, December 16, 1999. During this time, the DVJ2 video server received 10,298 requests to view videos which resulted in transferring almost 8.6 terabytes of data (see Table 5.99), assuming that the entire video was watched each time it was downloaded. This assumption is generally true in an educational environment, but this issue will be discussed further in Chapter 6. Table 5.100 shows that only 1.08% of all accesses were first-time accesses (compulsory misses), and almost all of the videos were accessed more than once. Finally, the stack depth analysis in Table 5.103 shows that both the normalized mean stack depth and the normalized median stack depths are relatively low. This implies that this workload trace has very good temporal locality.

Table 5.99
Summary of Access Log Characteristics for dvj2_99f Trace

Trace start time:	Mon Aug 23 14:47:37 1999
Trace end time:	Thu Dec 16 14:50:59 1999
Total Requests:	10,298
Avg Requests/Minute:	0.06
Total Kilobytes Transferred:	8,578,738,595
Avg Kilobytes Transferred/Minute:	51,802.92

Table 5.100
Summary of Other Characteristics for dvj2_99f Trace

Maximum Size Item (kilobytes):	1,232,620
Minimum Size Item (kilobytes):	141,260
Unique Requests:	113
Unique Workload Size (kilobytes):	92,665,903
Distinct Requests/Total Requests (%):	1.10
Distinct Bytes/Total Bytes (%):	1.08
Second Requests/Distinct Requests (%):	95.58
Second Bytes/Distinct Bytes (%):	96.16
Distinct Files Accessed Only Once (%):	4.42
Distinct Bytes Accessed Only Once (%):	3.84

Table 5.101
Object Reaccess Information by File Type for dvj2_99f Trace

File Type	**First Accesses**	**Second Accesses**	**Other Accesses**	**Total Accesses**
video	113 (1.1 %)	108 (95.6 %)	10,077 (9330.6 %)	10,298

Table 5.102
Unique File Size Information by File Type for dvj2_99f Trace

file type	**number**	**mean (bytes)**	**median (bytes)**	**maximum (bytes)**	**totalsize (bytes)**
all	113	820,052	785,050	1,232,620	92,665,903
video	113	820,052	785,050	1,232,620	92,665,903

Table 5.103
Stack Depth Analysis for dvj2_99f Trace

mean stack depth:	6.5
median stack depth:	17
standard deviation:	10.17
maximum stack depth:	98
normalized mean depth:	0.000,632
normalized median depth:	0.001,651

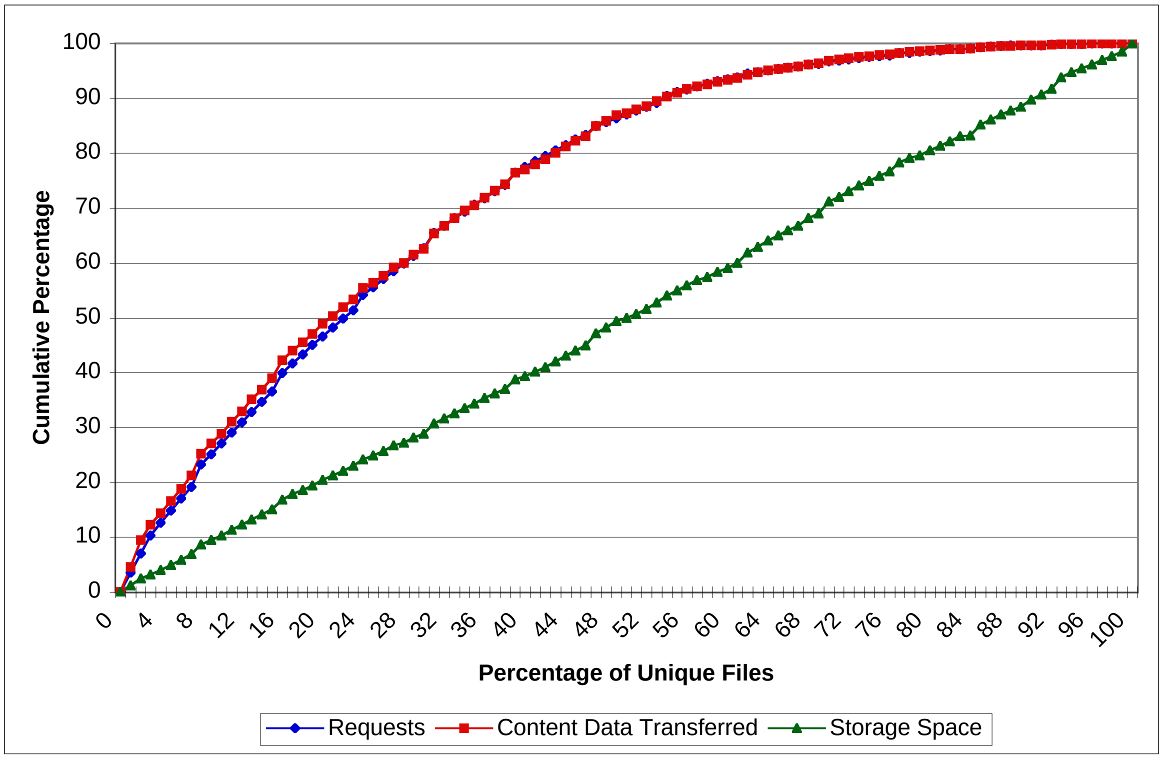


Fig. 5.11. Concentration of References for dvj2_99f Trace

5.6 OnCommand On-Demand Movie Server

The On Command system is a on-demand hotel video delivery system that allows guests to select from a set of movies that the system has available [OnC99a]. The On Command workload trace is from a hotel that has 722 rooms for which 33 titles were available. These titles are in five categories: action & adventure, comedy, kids & family, drama, and adult. The system that is deployed in this particular hotel is called a “Modular 600 D”. On this system, the most popular titles are stored on a hard disk cache and support nearly unlimited access. Less popular titles are stored using another technology that limits access to one user for a period of approximately one half hour. That is, a single two-hour title may have four users (spaced one half hours apart each) while users who wished to access the titles in the meantime may be denied (given an out-of-copy message). Fortunately, this occurs infrequently. The system supports a well-known, but unpublicized, preview feature. Movies may be started, but will not be billed for two minutes. Titles cancelled before the preview time elapses will not be billed. This causes users to behave in a particular way. Since an transaction record for reaching the end of a movie is not alway recorded, an assumption must be made that a movie that was not cancelled during the preview period was watched in its entirety. It is assumed that each movie is 120 minutes long.

In Table 5.104, the number of total requests for videos was 1479, totaling 2,958.0 hours of video. The average minutes of video transferred per minute means that on average there were over 16 videos playing at a time. Every video was accessed more than once during the course of the week, and only 2.23% of all of the accesses were first time accesses (compulsory misses) according to Table 5.105. Tables 5.106 and 5.107 show that all of the accesses were of type “video” as expected. In Table 5.109, the normalized mean and median stack depths are rather small. This should be expected since there are not many unique videos available; yet these videos have a very random access pattern as opposed to what would be expected from an educational video delivery system. Out of all the accesses in the week, 390 out of 1479 requests or 26.37% were for preview only.

The OnCommand system trace has been graciously provided by Don Power, the Chief Technologist at OnCommand Corporation of San Jose, California.

Table 5.104
Summary of Access Log Characteristics for daysum1 Trace

Trace start time:	Tue Nov 9 00:06:03 1999
Trace end time:	Wed Nov 17 13:15:11 1999
Total Requests:	1,479
Avg Requests/Minute:	0.12
Total Minutes of Video Transferred:	177,480
Avg Minutes of Video Transferred/Minute:	16.33

Table 5.105
Summary of Other Characteristics for daysum1 Trace

Maximum Size Item (seconds):	7,200
Minimum Size Item (seconds):	7,200
Unique Requests:	33
Unique Workload Size (seconds):	237,600
Distinct Requests/Total Requests (%):	2.23
Distinct Bytes/Total Bytes (%):	2.23
Second Requests/Distinct Requests (%):	100.00
Second Bytes/Distinct Bytes (%):	100.00
Distinct Files Accessed Only Once (%):	0.00
Distinct Bytes Accessed Only Once (%):	0.00

Table 5.106
Breakdown By File Type for daysum1 Trace

File Type	**% of Requests**	**% of Bytes Transferred**
video	100.00	100.00

Table 5.107
Multirequest Breakdown By File Type for daysum1 Trace

File Type	**% of Requests**	**% of Bytes Transferred**
video	100.00	100.00

Table 5.108
Object Reaccess Information by File Type for daysum1 Trace

File Type	**First Accesses**	**Second Accesses**	**Other Accesses**	**Total Accesses**
video	33 (2.2 %)	33 (100.0 %)	1,413 (4281.8 %)	1,479

Table 5.109
Stack Depth Analysis for daysum1 Trace

mean stack depth:	7.28
median stack depth:	2
standard deviation:	7.13
maximum stack depth:	31
normalized mean depth:	0.004,919
normalized median depth:	0.001,352

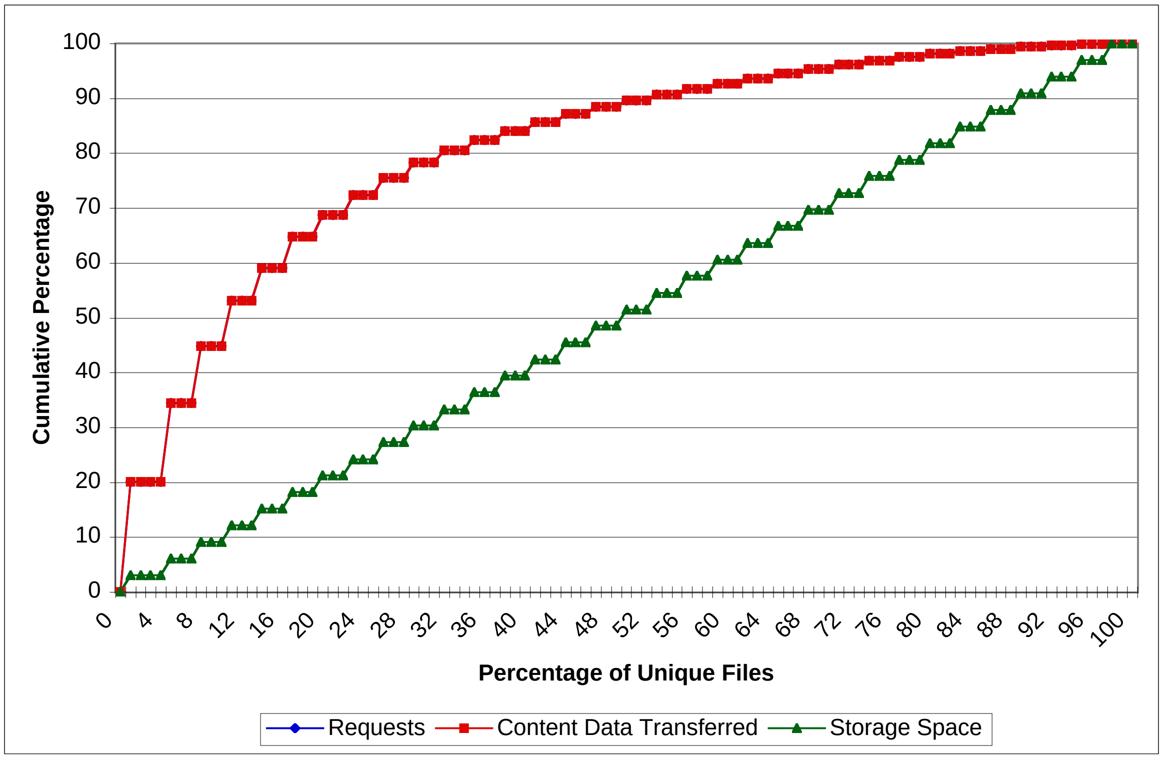


Fig. 5.12. Concentration of References for daysum1 Trace

5.7 Stochastically Generated Video Server Model Traces

As part of my Masters thesis, I developed several stochastic models that simulate different student access patterns that we have seen on the DVJ2 educational multimedia testbed systems [Reu96]. Using these stochastic models, simulated usage data was also generated. Over the course of a semester, the number of accesses will rise as the topic of a given object is being covered in the sequence of the course. The popularity of that given object will wane after its topic has been covered in class, and thereafter will only be accessed for review of the material. So in terms of the probability, say an object, m_i is chosen, where $1 \leq i \leq M$, and the object numbers progress sequentially as the semester proceeds. If the object m_i is one of the first objects of the sequence, its probability of being chosen is very high at the beginning of the semester and then fades. Similarly a object later in the sequence will have little or no probability of being chosen early in the sequence and gains probability later in the sequence. For each of these distributions, the simulation parameters were presented in Section 4.3.2.

5.7.1 Defining the Sequential Distribution Data Sets

With this in mind, a simulation can be built that attempts to model these characteristics. The intention of this simulation is to use a set of conditional discrete distributions with a differing parameter t. This t is an integer, $0 \leq t < M$, which is an index to the t^{th} time period of the semester and tracks some aspect of the distribution. For instance, with the binomial set of distributions, t is the numerator of the equation $p = t/(M-1)$ that is then used to calculate the individual probabilities for that given t. Since this t can be seen as a marking point relative to where a class is in a semester, it may indicate what video object is the most popular or the video object most likely to be chosen, depending on the distribution.

The composition of these data sets then entails traversing sequentially over all possible values of t, $0 \leq t < M$, where M is the number of media objects and the total number of time periods in the semester. For each value of t, the t^{th} conditional discrete distribution is used to choose a certain number of random integers which correspond to objects that are chosen to be viewed by students.

Define $p_{i,j} = \mathcal{P}\{m = i|t = j\}$ to be the probability that the random variable m, which is the object currently chosen, is equal to i given the time period index t is equal to j. Let $\overline{p}_j$ be the column vector whose i^{th} entry is $p_{i,j}$. Each $\overline{p}_j$ is then a conditional distribution.

With this, define the matrix:

$$\mathcal{P} = \left[\begin{array}{cccc} \overline{p}_0 & \overline{p}_1 & \dots & \overline{p}_{M-1} \end{array} \right]. \tag{5.1}$$

This makes $\mathcal{P}$ an $M \times M$ matrix. Now define d_i as the probability that object i is chosen at any point in the entire semester. Define:

$$d = \left[\begin{array}{cccc} d_0 & d_1 & \dots & d_{M-1} \end{array} \right]^T. \tag{5.2}$$

So, $d_i = \mathcal{P}\{m = i\}$. Finally define x_j as the probability of each t occurring, $x_j = \mathcal{P}\{t = j\}$, and

$$x = \left[\begin{array}{cccc} x_0 & x_1 & \dots & x_{M-1} \end{array} \right]^T. \tag{5.3}$$

Now, in terms of conditional probability:

$$\textstyle\sum_{j=0}^{M-1} \mathcal{P}\{m = i|t = j\} \cdot \mathcal{P}\{t = j\} = \mathcal{P}\{m = i\}, \quad 0 \leq i < M \ . \tag{5.4}$$

And in terms of the vectors and matrices defined above:

$$\textstyle\sum_{j=0}^{M-1} p_{i,j} x_j = d_i \quad 0 \leq i < M \ . \tag{5.5}$$

From these equations, one can see that the probabilities, t, for choosing object m, $p_{i,j}$, multiplied times the probability that the t^{th} distribution is used overall, x_j, summed together equals the overall expected probability of object m being used in the entire experiment, d_i. From this it can be written:

$$\mathcal{P}x = d. \tag{5.6}$$

$\mathcal{P}$ will be a $M \times M$ matrix while $\overline{x}$ and $\overline{d}$ are both $M \times 1$ vectors.

5.7.2 Truncated Discrete Exponential Distribution $\mathcal{P}$ Matrix

Now a truncated discrete exponential distribution can be defined. $p_{i,j}$ is the j^{th} element of the conditional distribution , $0 \leq j \leq i$. For a given i, $\lambda = c/(i+1)$ and

$$\hat{p}_{i,j} = \lambda e^{-\lambda(i-(j+1))} - \lambda e^{-\lambda(i-j)}, \quad 0 < i \leq j \tag{5.7}$$

$$p_{i,j} = 0, \quad i < j < M \ . \tag{5.8}$$

Because this exponential distribution goes to negative infinity past $j = 0$, the distribution must be truncated. To realize the truncation, take the modulo of any objects that fall out of the bounds, $0 \leq j \leq i$, as modulo i thereby distributing the probability almost evenly over the range 0 to i. (Actually the modulo i probability over the range 0 to i is not even but with $c = 8$, out-of-bounds objects only account for 0.0335% of all occurrences, so the approximation is justifiable.) In terms of the probabilities, $p_{i,j}$,

$$p_{i,j} = \hat{p}_{i,j} + \frac{1-\sum_{k=0}^{M-1} \hat{p}_{i,k}}{i+1}, \quad 0 \leq j \leq i \ . \tag{5.9}$$

Basically, the $p_{i,j}$'s share the leftovers of the vector to make the vector components sum to 1, but the leftovers are usually fairly small. Yet, they are significant because the sum of the vector components of $\overline{p}_j$ must each sum to 1 to satisfy one of the axioms of probability [Pap91].

This distribution models the peak in popularity of a object when the object's topic is being covered in class and the possibility of students wanting to access a very early object late in the semester. To better visualize these conditional distributions, Figure 5.13 shows selected discrete probability density functions, while Figure 5.14 shows selected discrete cumulative distribution functions of the truncated discrete exponential distributions.

To obtain the $\overline{x}$ vector, assume that each i is equally probable, $x_i = 1/M$, $0 \leq i < M$. Knowing the $\mathcal{P}$ matrix from Equations 5.6, 5.7, and 5.8 and the values of the $\overline{x}$ vector, it is possible to solve for the $\overline{d}$ vector, the overall distribution of object choices for the entire experiment. Figure 5.15 illustrates the values of the $\overline{d}$ vector. This resulting $\overline{d}$ vector can be explained by some student usage characteristics. The very

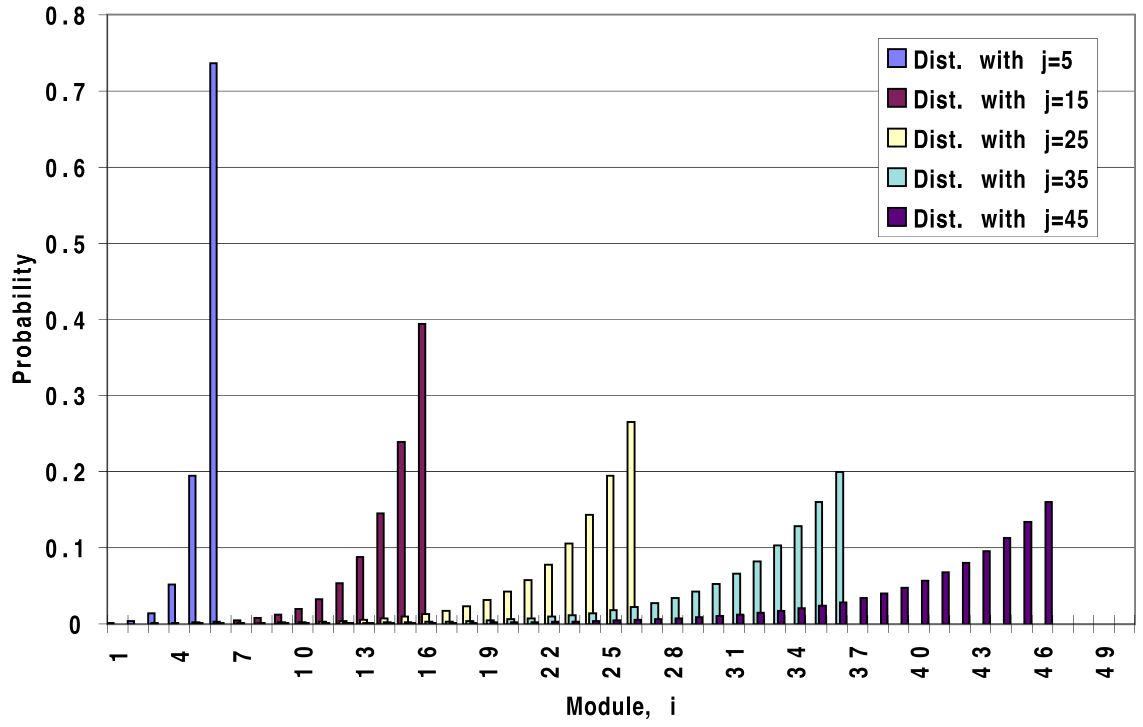


Fig. 5.13. Selected conditional probability density functions for truncated discrete exponential distribution $\mathcal{P}$ matrix.

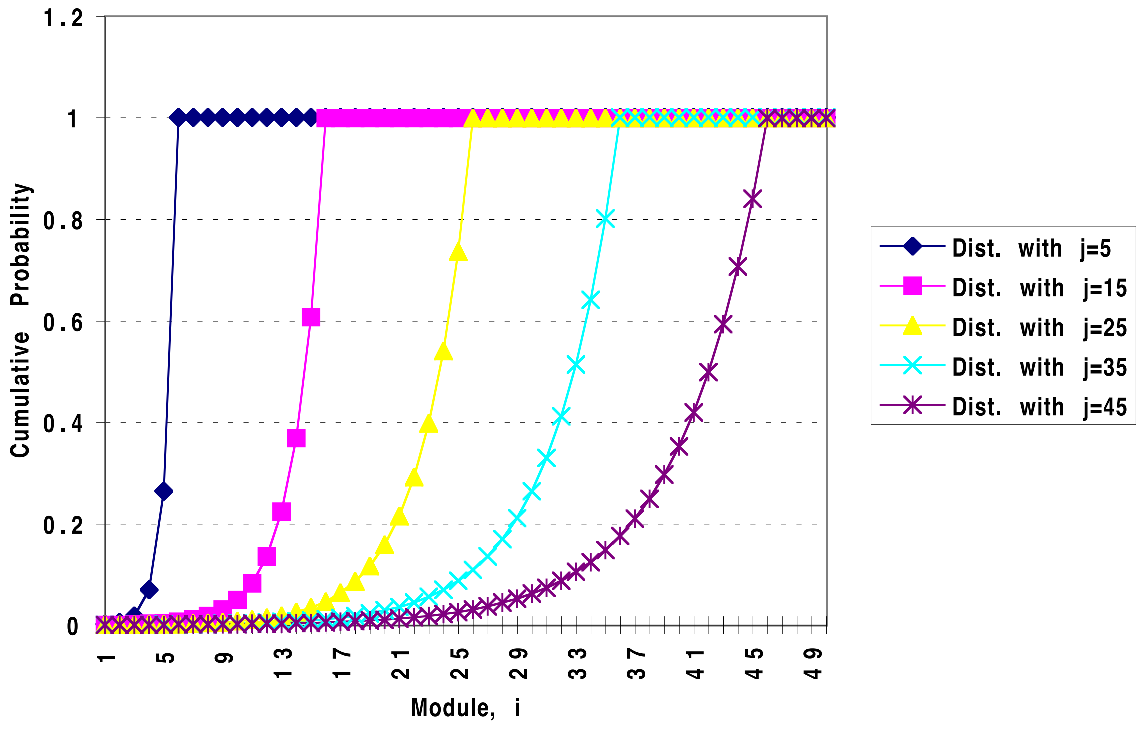


Fig. 5.14. Selected conditional cumulative distribution functions for truncated discrete exponential distribution $\mathcal{P}$ matrix.

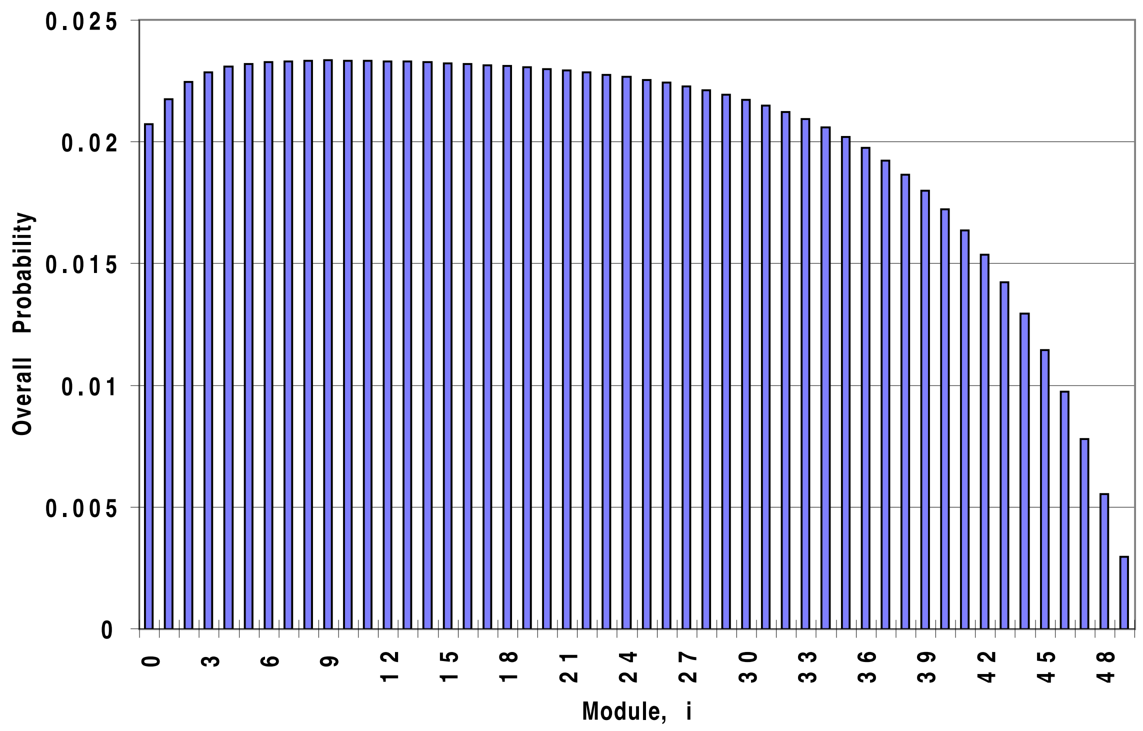


Fig. 5.15. Overall object choice distribution, $\overline{d}$, for truncated discrete exponential distribution $\mathcal{P}$ matrix.

early video objects may not be quite as popular than others because they contain review material. Early in the semester, students are dedicated to keeping up in the class and they watch every object that corresponds to the current course material. But as the semester progresses and other courses invade on the students' time, they find less time to watch the video objects and keep up with the class. The usage of the objects reflect this by tapering off.

5.7.3 Binomial distribution $\mathcal{P}$ matrix

For a conditional set of binomial distributions, define distribution parameter $q = i/(M-1)$, $0 \leq i < M$. For each i, define:

$$p_{i,j} = \left(\begin{array}{c} M \\ j \end{array}\right) \left(\begin{array}{c} i \\ M \end{array}\right)^{j} (1 - \frac{i}{M})^{M-j} = \left(\begin{array}{c} M \\ j \end{array}\right) q^{j} (1-q)^{M-j} \tag{5.10}$$

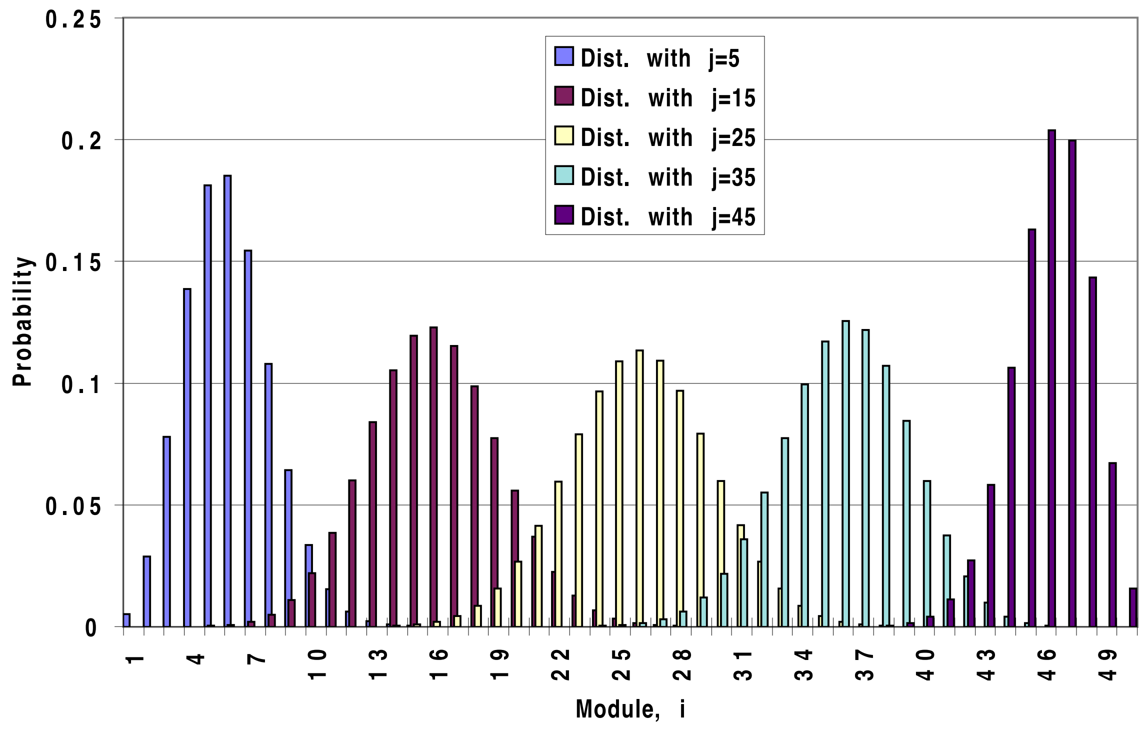


Fig. 5.16. Selected conditional probability density functions for binomial distribution $\mathcal{P}$ matrix.

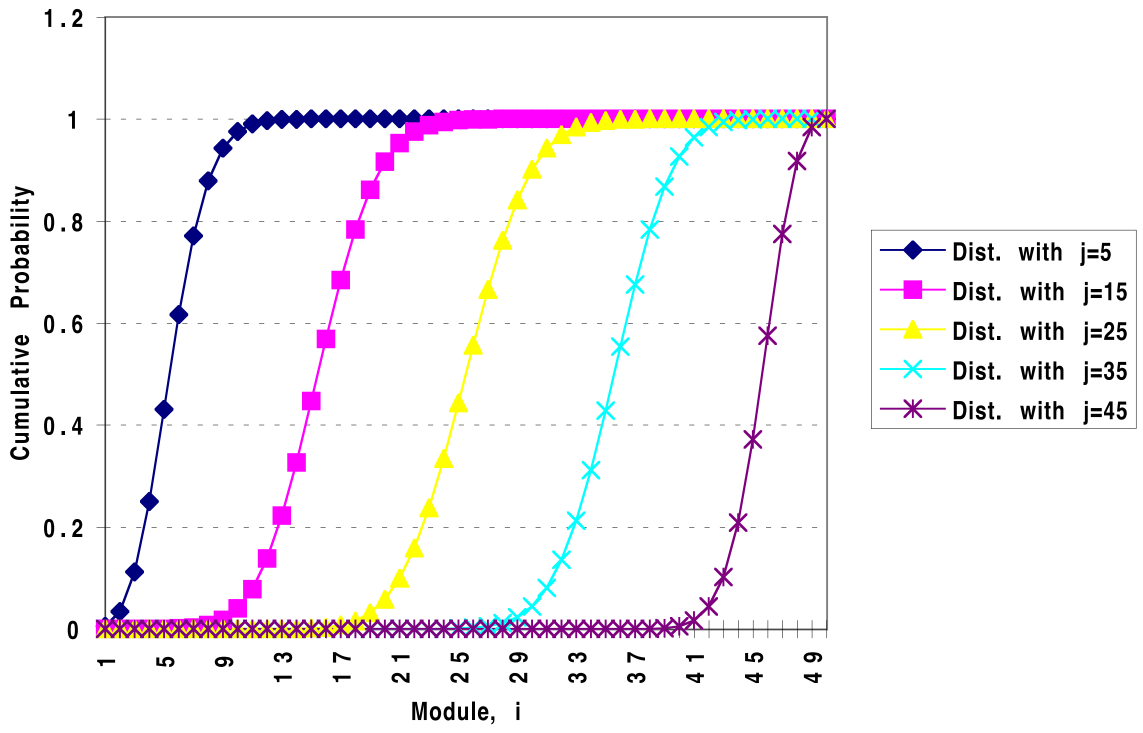


Fig. 5.17. Selected conditional cumulative distribution functions for binomial distribution $\mathcal{P}$ matrix.

Since the conditional distributions, p_j, are discrete Gaussian distributions, this set of distributions models the expectation that objects on similar topics will all be popular around the same time period. It also does well with the possibility of a student watching a object with a topic from early in the semester or looking ahead to a object whose topic is covered late in the semester. To better visualize these conditional distributions, Figure 5.16 shows selected discrete probability density functions, while Figure 5.17 shows selected discrete cumulative distribution functions of the binomial distributions.

To obtain the $\overline{x}$ vector, assume that each value of i is equally probable, $x_i = 1/M$, $0 \leq i < M$. Knowing the $\mathcal{P}$ matrix from Equation5.10 and the values of the $\overline{x}$ vector, it is possible to solve for the $\overline{d}$ vector, the overall distribution of object choices for the entire experiment. Figure 5.18 illustrates the values of the $\overline{d}$ vector. This resulting $\overline{d}$ vector models a student usage pattern where they access each video object with equal probability over the entire semester.

5.7.4 Triangular Window Distribution $\mathcal{P}$ Matrix

The truncated discrete exponential distribution and the binomial distribution each have some limitations in how they model expected student activity on a distributed multimedia system. The exponential distribution does not model any students' choices of objects that are greater than parameter t. The binomial distribution makes it equally likely that a object $j + k$ will be access as the object $j - k$ for a given integer j and arbitrary interval integer k. This is not what would be expected because students generally are more likely to access the object $j - k$ rather than the object $j + k$. Thus, a third distribution was developed. The set of triangular window distributions accounts for students being more likely to access objects $j - k$ than accessing objects $j + k$ and accounts for students working ahead as well as lagging behind. It is comprised of the area under a triangle. This triangle is depicted in Figure 3.13 as a dotted line while the probabilities of choosing the individual objects is the underlying bar graph.

The triangle is determined by how many objects to the left of parameter j have

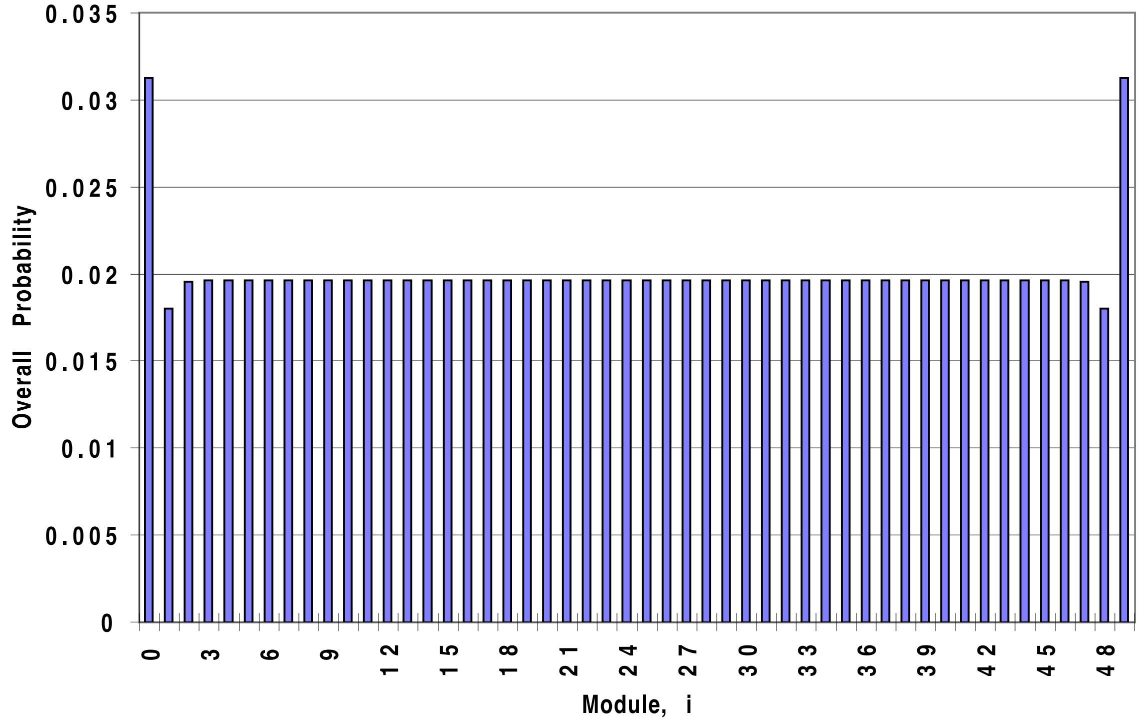


Fig. 5.18. Overall object choice distribution, $\overline{d}$, for binomial distribution $\mathcal{P}$ matrix.

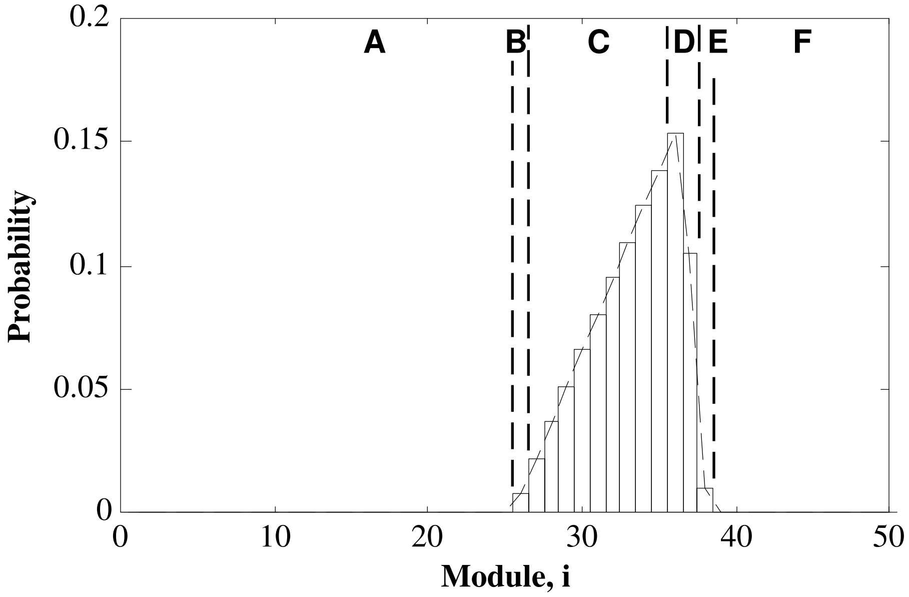


Fig. 5.19. Triangular window distribution for $j = 35$ with $M = 50$.

a nonzero probability, which is variable a, and how many objects to the right of parameter m have a nonzero probability, which is variable b. There are two input parameters, *amax* and *bmax*. *amax* is the largest fraction of M that variable a can become (when $j = M - 1$), while *bmax* is the largest fraction of M that variable b can become (when $j = 0$). The variables a and b scale linearly from 0 to $M-1$. So as j increases, b decreases linearly simulating the likelihood of students working ahead which is expected to decrease as the semester progresses. Also a increases linearly as t increases simulating the higher probability of students reviewing material as the semester advances. To calculate the individual probabilities, $p_{i,j}$, first the values of a and b are calculated:

$$a = \left(\frac{amax - 1}{M - 1}\right) j + \left(\frac{M - amax}{M - 1}\right) \tag{5.11}$$

and

$$b = -\left(\frac{bmax}{M - 1}\right) j + \left(\frac{M \cdot bmax}{M - 1}\right). \tag{5.12}$$

By knowing a and b, the maximum height, h, of the triangle can be determined:

$$h = \frac{2}{a + b}. \tag{5.13}$$

Now the lines that bound the triangle can then be determined:

$$ay(x) = \left(\frac{h}{a}\right) x + \left(h - \frac{j \cdot h}{a}\right) \tag{5.14}$$

and

$$bz(w) = -\left(\frac{h}{b}\right) w + \left(h + \frac{j \cdot h}{b}\right). \tag{5.15}$$

The ay equation defines the line that connects $(j-a, 0)$ to (j, h), while the bz equation defines the line that connects $(j + b, 0)$ to (j, h).

With these lines, the probabilities for each object can be calculated for each $p_{i,j}$ given $t = j$. As illustrated in Figure 5.19, the conditional probability density calculations can be divided into six parts, labeled A through F. In region A, where $i < \lfloor j - a \rfloor$, $p_{i,j} = 0$. ($j - a$ and $j + b$ are a real numbers while j is always an integer.) The leftmost tip of the triangle is region B where $i = \lfloor j - a \rfloor$. Here,

$$p_{i,j} = \frac{1}{2}(a - (j - (i + 1))) * ay(i + 1). \tag{5.16}$$

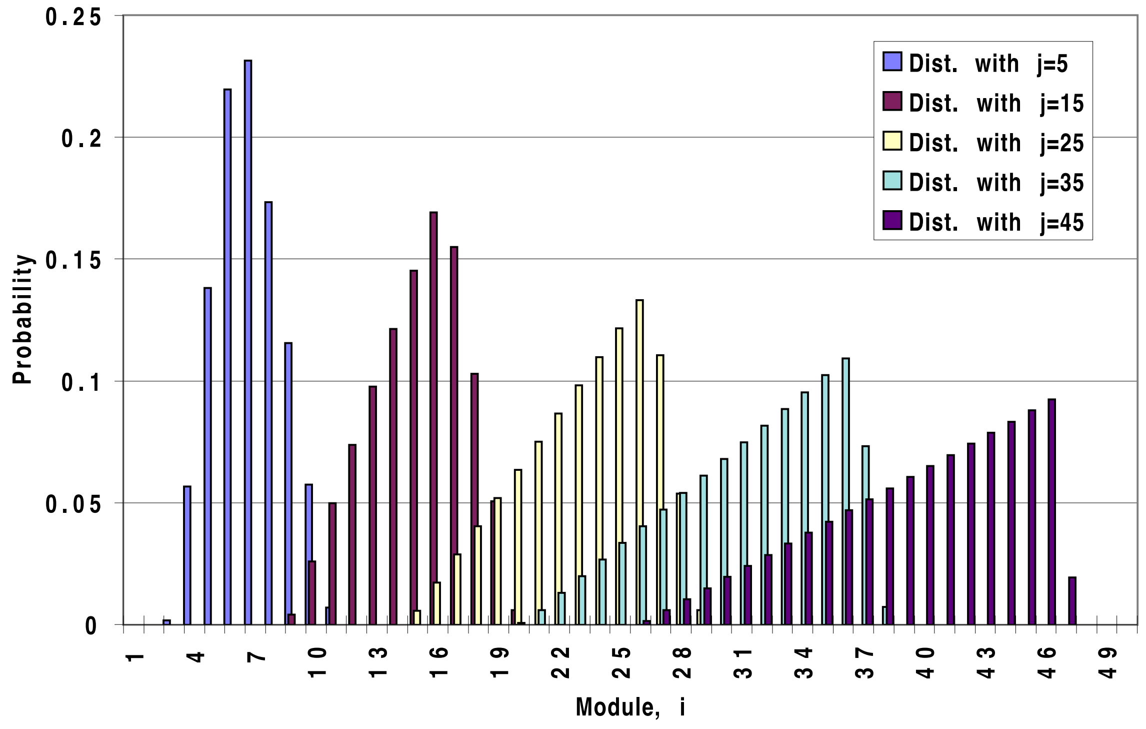


Fig. 5.20. Selected conditional probability density functions for triangular window distribution $\mathcal{P}$ matrix with $amax = 0.45$ and $bmax = 0.1$.

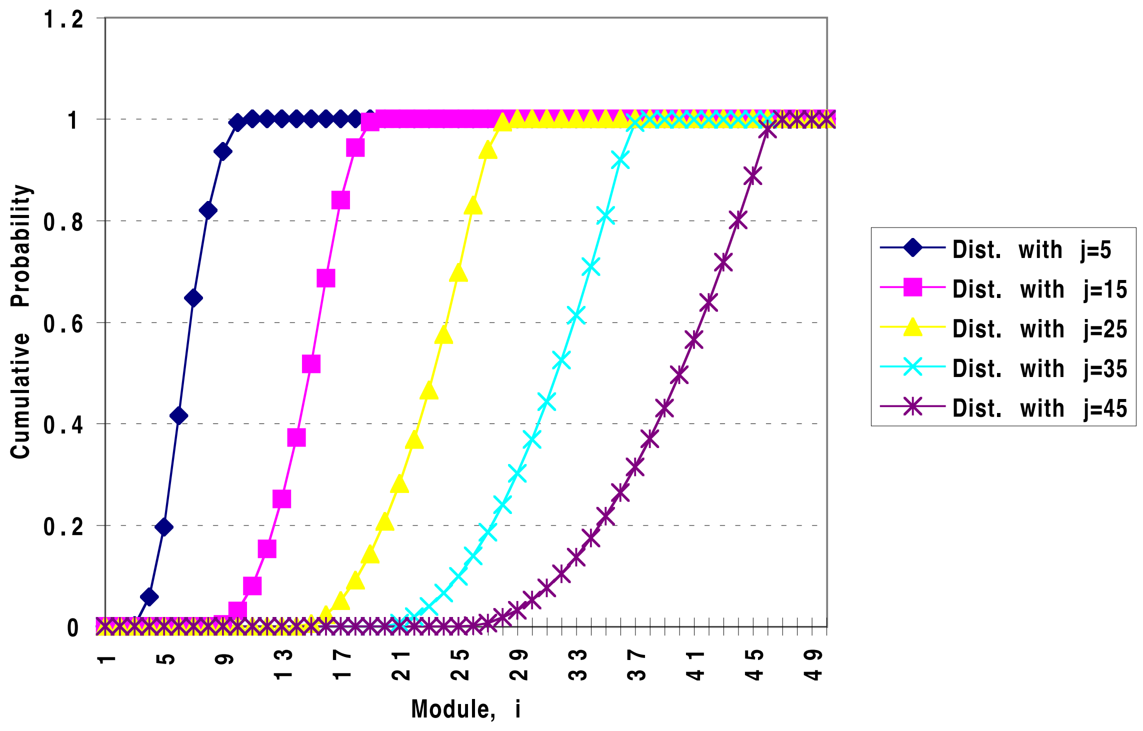


Fig. 5.21. Selected conditional cumulative distribution functions for triangular window distribution $\mathcal{P}$ matrix with $amax = 0.45$ and $bmax = 0.1$.

Region C is where $\lfloor j - a \rfloor < i < j$ and

$$p_{i,j} = \left[\frac{1}{2}(at(i+1) - ay(i)) * 1\right] + [ay(i) * 1]\,. \tag{5.17}$$

In the above equation, the first bracketed portion is the triangular part of the area on top of the column, while the second bracketed part is the column below. Region D is defined where $j \leq i < \lfloor j + b \rfloor$ and

$$p_{i,j} = \left[\frac{1}{2}(bz(i) - bz(i+1)) * 1\right] + [bz(i+1) * 1]\,. \tag{5.18}$$

Again in this equation, the first bracketed portion is the triangular part of the area on top of the column, while the second bracketed part is the column below. The rightmost tip of the triangle is region E, where $i = \lfloor j + b \rfloor$ and

$$p_{i,j} = \frac{1}{2}(b - (i - j)) * bz(i) \tag{5.19}$$

Finally, Region F is where $i > \lfloor j + b \rfloor$ and $p_{i,j} = 0$. To better visualize these conditional distributions, Figure 5.20 shows selected discrete conditional probability density functions, while Figure 5.21 shows selected discrete conditional cumulative distribution functions of the triangular window distributions.

To obtain the $\overline{x}$ vector, assume that each value of i is equally probable, $x_i = 1/M$, $0 \leq i < M$. Knowing the $\mathcal{P}$ matrix as defined above and the values of the $\overline{x}$ vector, it is possible to solve for the $\overline{d}$ vector, the overall distribution of object choices for the entire experiment. Figure 5.22 illustrates the values of the $\overline{d}$ vector. This resulting $\overline{d}$ vector can be explained by some student usage characteristics. As with the overall object choice distribution of the truncated discrete exponential distributions, the very early video objects may not be quite as popular than others because they contain review material. Early in the semester, students are dedicated to keeping up in the class and they watch every object that corresponds to the current course material. But as the semester progresses and other courses invade on the students' time, they find less time to watch the video objects and keep up with the class. The usage of the objects reflect this by tapering off even more dramatically than in the overall object choice distribution of the truncated discrete exponential distributions.

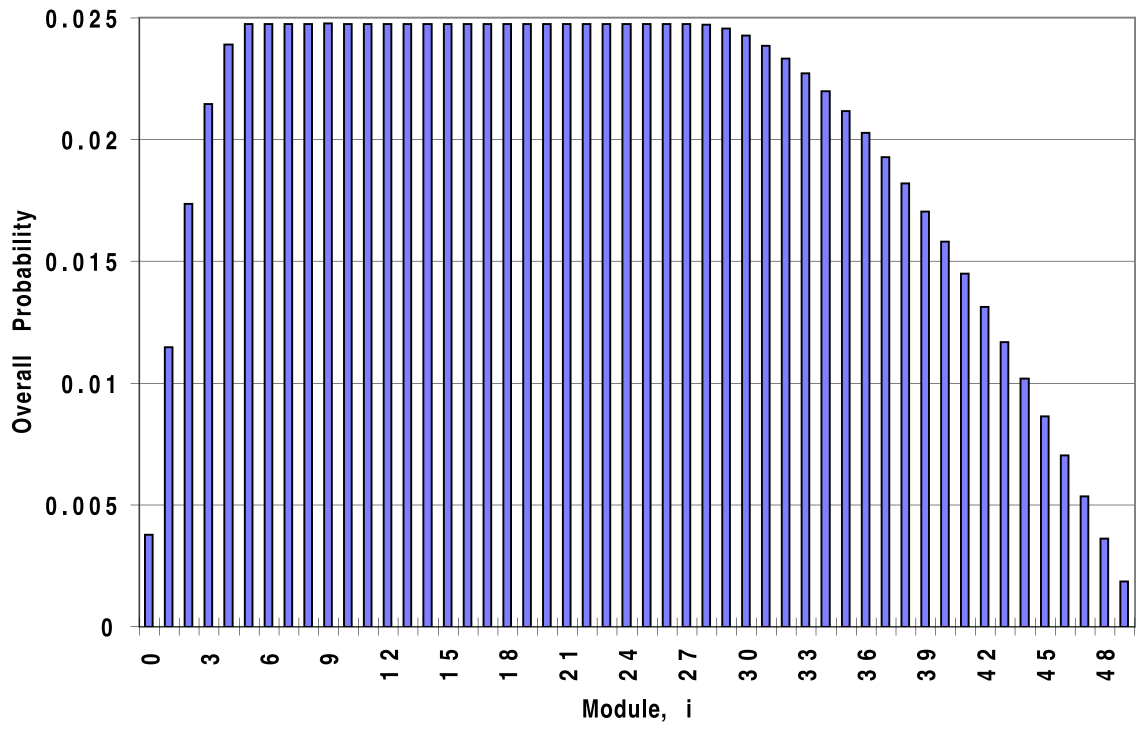


Fig. 5.22. Overall object choice distribution,$\overline{d}$, for triangular window distribution $\mathcal{P}$ matrix with $amax = 0.45$ and $bmax = 0.1$.

6. Simulation Results

Now that an understanding of the characteristics of the workload traces from Chapter 5 has been achieved, the results of the cache replacement algorithm simulations will be presented. This chapter is divided into three sections which correspond to the three questions that were posed in Section 2.9. These questions asked which cache replacement algorithms perform best for which workload traces, inquired whether admission policies are effective in increasing performance in a cache, and questioned whether entire videos should be cached in a video proxy cache server.

6.1 Which Cache Replacement Algorithms Are Best for Each Workload Trace?

The first set of questions in Section 2.9 asked the following: By including more statistics of the cache object in the determination of cache removal, do the more complex cache replacement algorithms better predict the expected demand for objects in the cache? And which object statistics are involved in the algorithms that perform better, i.e., which algorithms produce a lower cache miss rate and which produce a lower cache byte hit rate? Is the additional computational complexity of using more cache object statistics worth the decrease in cache miss rates and byte miss rates?

To answer these questions, the workload traces are further segmented into the domains that were developed in Section 5.3. Then within each segment, the results are discussed for its constituent workload traces. For each workload, eight graphs have been generated; four compare the miss rate results, and four compare the byte miss rate results.

To better compare how the cache replacement algorithms perform for different cache sizes, three line graphs are then presented. On each of these three line graphs, the COMP (compulsory miss), OPT (Belady's optimal) and RND (random) cache

Table 6.1
Comparison of Object Statistics in Cache Replacement Algorithms

Policy	on chart	no. of stats	first access	last access	access count	size size	file type	admission affect
COMP	all	0						no
OPT	all	0						yes
RND	all	0						no
FIFO	1	1	x					no
LRU	1	1		x				no
SizeL	1	1				x		yes
LFU	1	1			x			no
PrfctLFU	1	1			x			yes
LFU-DA	1	1			x			yes
LRD	2	2	x		x			no
SizeLRD	3	3	x		x	x		no
LRU-MIN	1	2		x		x		yes
SLRU	2	2		x	x			yes
PLRU	2	2		x			x	no
SPCxAGE	2	2		x		x		no
SPCxAGEtc	2	2		x		x		no
LFLRU	2	2		x	x			no
SizeLFLRU	3	3		x	x	x		no
GDS1	3	1				x		yes
GDSPacket	3	1				x		yes
GDSHits	3	2			x	x		yes
LRV	3	3		x	x	x		no

replacement policy lines are displayed for the sake of comparison. COMP and OPT indicate how close to optimal the performance results are while RND provides an indication of how bad a poorly performing algorithm is. If a cache replacement algorithm is performing worse than random, it is performing quite poorly on that particular workload trace. The first of the three line graphs compares those algorithms that use only one object statistic or a simple combination of two statistics. These algorithms are FIFO, LRU, SizeL, LFU, PerfectLFU, LFU-DA, and LRU-MIN. The results of the best performing of these algorithms (except for PerfectLFU, see next paragraph for an explanation) are forwarded to the second line graph. On the second line graph, algorithms that use two statistics and are somewhat more complex are compared. These algorithms are LRD, SLRU, PLRU, SPACExAGE, SPACExAGEtc, and LFLRU. The results of the two top performing algorithms from the second line graph are forwarded to the third line graph. The third line graph displays the final comparison and includes the algorithms that use three object statistics and are yet more complex. These include SizeLRD, SizeLFLRU, GDS1, GDSPacket, GDSHits, and LRV. Table 6.1 summarizes which algorithms are compared on each of the line graphs, but it does not include the forwarded algorithms. Also for each given workload, an overview miss rate bar graph is first presented which displays the average miss rates over all five cache sizes. These average miss rates are a very good indication of the overall performance of the algorithms. For instance, one policy may be very effective for smaller cache sizes and then perform relatively poorly for larger cache sizes. After the miss rate results have been discussed for a given workload trace, the byte miss rate graphs are displayed and discussed in a parallel manner.

For all of the line graphs, the COMP, OPT, RND, and PerfectLFU cache replacement algorithms are not forwarded because they are ideal algorithms that are in this study for the sake of comparison. The matter that COMP, OPT, and RND should be comparison algorithms is quite straightforward and follows from the discussion in Section 2.8. However, the choice to make PerfectLFU a comparison algorithm is not as clear. This choice was made because PerfectLFU does not have any mechanism for aging the score of its objects, whether the object is in the cache or not. This means that an object that was very popular months ago would probably still occupy the cache because most other objects had not been accessed as many times. Also as mentioned in Section 2.6, PerfectLFU can be prohibitively expensive in memory consumption because it requires the cache proxy server software to maintain access counts of *all* of the objects that it has cached since it was started. The PerfectLFU algorithm could be modified so that it purges the access count data structure when it seems that an object would no longer be accessed, but such a modification would no longer be implementing the PerfectLFU algorithm in its purest sense. Therefore, the PerfectLFU algorithm results are not forwarded among the line graphs.

For each of the proxy server domains, a summary will be drawn for both miss rate and byte miss rate trends. In these domain summaries, a table is presented showing the top performing algorithms for each trace and both measures. Then, a number of comparisons will be made among the cache replacement algorithms which will elucidate certain performance advantages and disadvantages of the object statistics as part of the cache replacement algorithms. Among these comparisons are:

- A comparison of LRU to LRU-MIN, SPACExAGE, LFLRU, SLRU, and PLRU for influences of size, access count, segmenting, and file type with the given workload trace.

- A comparison of LFU to LFU-DA and PerfectLFU which provides an indication of how well the algorithms work with access counts.

- A comparison of FIFO to LRD and SizeLRD and LRU to LFLRU and Size-LFLRU which helps determine whether access count and object size improve the performance of the FIFO algorithm.

- A comparison of SPACExAGE versus SPACExAGEtc which gives indication of how sensitive a trace is to time measured either by number of other requests since a certain object was requested versus the actual seconds elapsed since a certain object was requested.

- A comparison of LFU-DA, GDS1, GDSPacket, and GDSHits with the other algorithms to determine the effectiveness of the inflation parameter in these algorithms.

- A comparison of LRV with the other algorithms to determine whether the complexity of the LRV algorithm is worth the computational effort.

6.1.1 Origin Server Traces

By using origin server traces, the simulation acts like an origin proxy cache server.

HP World Cup 1998 Web Server Farm Trace

For the simple cache replacement algorithms on the HP World Cup 1998 Web Server Farm wc_week14 trace, Figure 6.1 shows that the SizeL algorithm on the whole performs best. PerfectLFU is the top performing algorithm, but its results are not forwarded to the next graph. It is interesting to note that both SizeL and LRU-MIN outperform Belady's OPT optimal algorithm. The reason that OPT is not completely optimal is that it doesn't take object size into account; it assumes that all objects are of the same size. This situation comes up for the HP World Cup 1998 Web Server Farm Trace and several other traces that will be discussed in this chapter.

With the more complex algorithms in Figure 6.2, SPACExAGE and SPACExAGEtc perform exactly the same and are the best. Since those two algorithms have the exact same miss rates, only the results of one of them are forwarded, and the SizeL results, the next best algorithm, are forwarded on to the final comparison graph.

In the final comparison graph, Figure 6.3, and the average comparison graph, Figure 6.4, the GDSHits algorithm performs best by an average of 2% better than its next competitor. Overall, there is a three way tie for second between SPACExAGE, SPACExAGEtc, and SizeLFLRU.

For the simple algorithms with byte miss rate as the measure in Figure 6.5, the PerfectLFU is the best algorithm while the LFU-DA algorithm is a close second. The LFU-DA algorithm results are forwarded to the next graph.

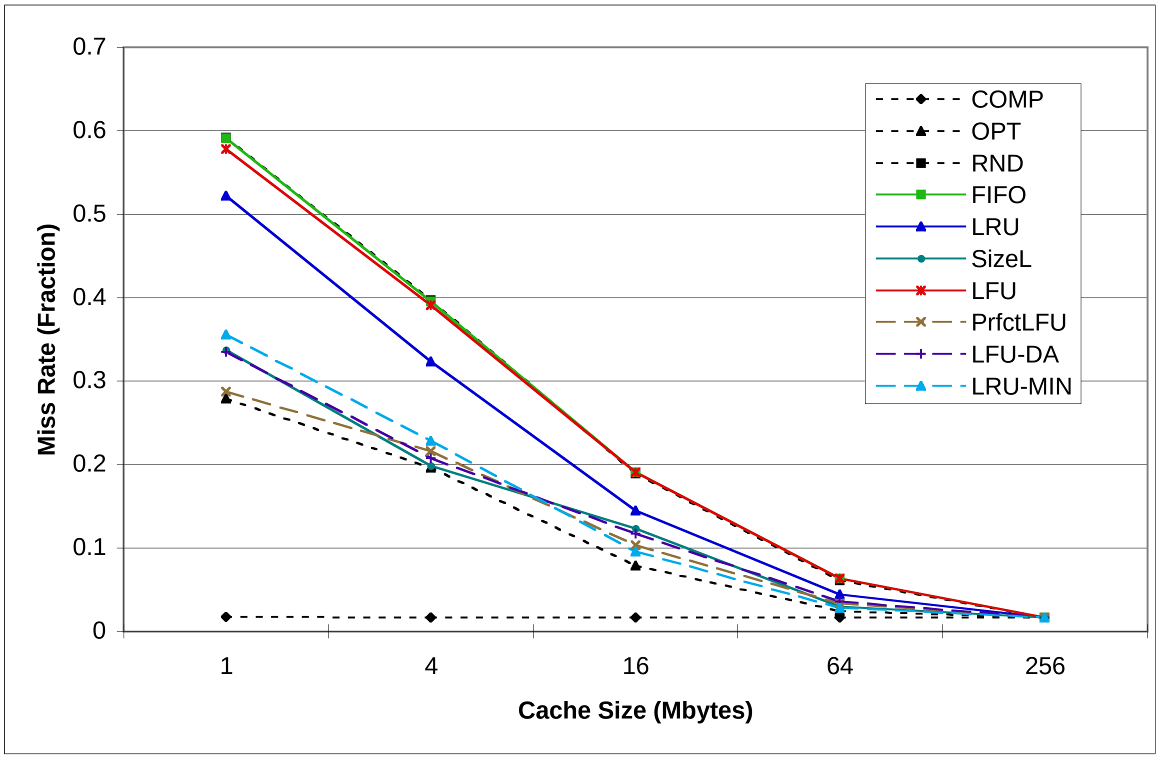


Fig. 6.1. Miss Rates of Simple Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

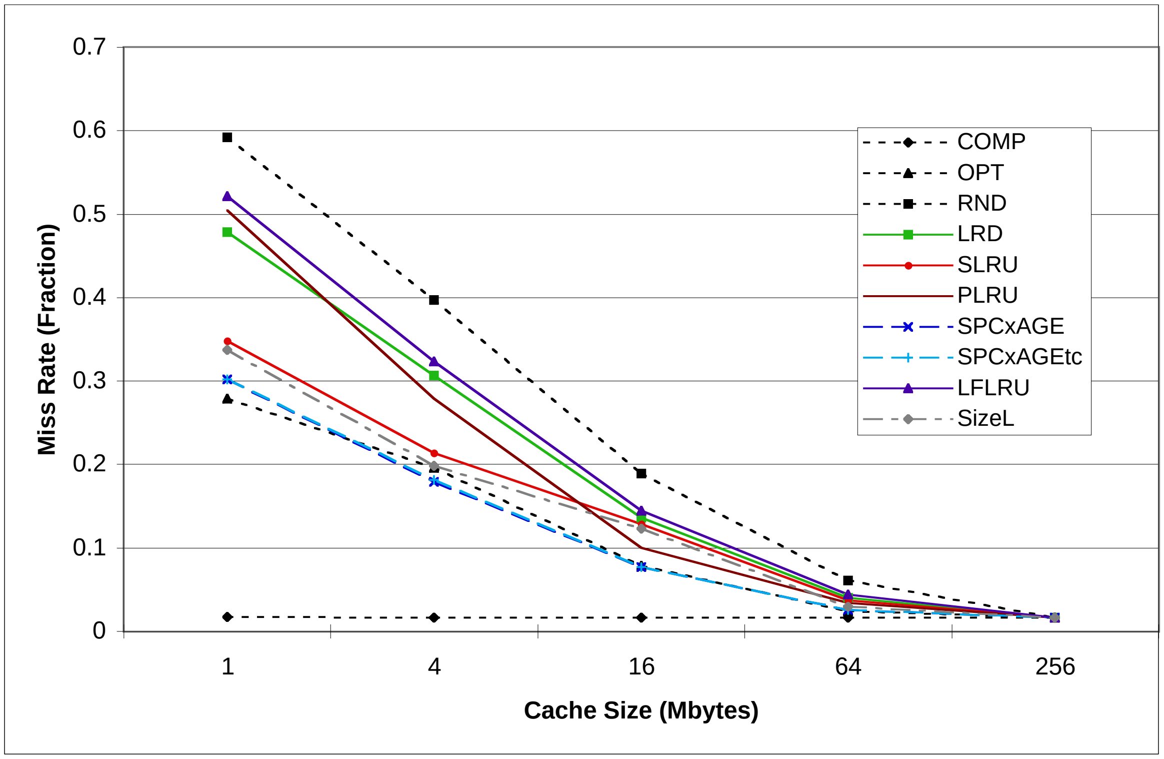


Fig. 6.2. Miss Rates of More Complex Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

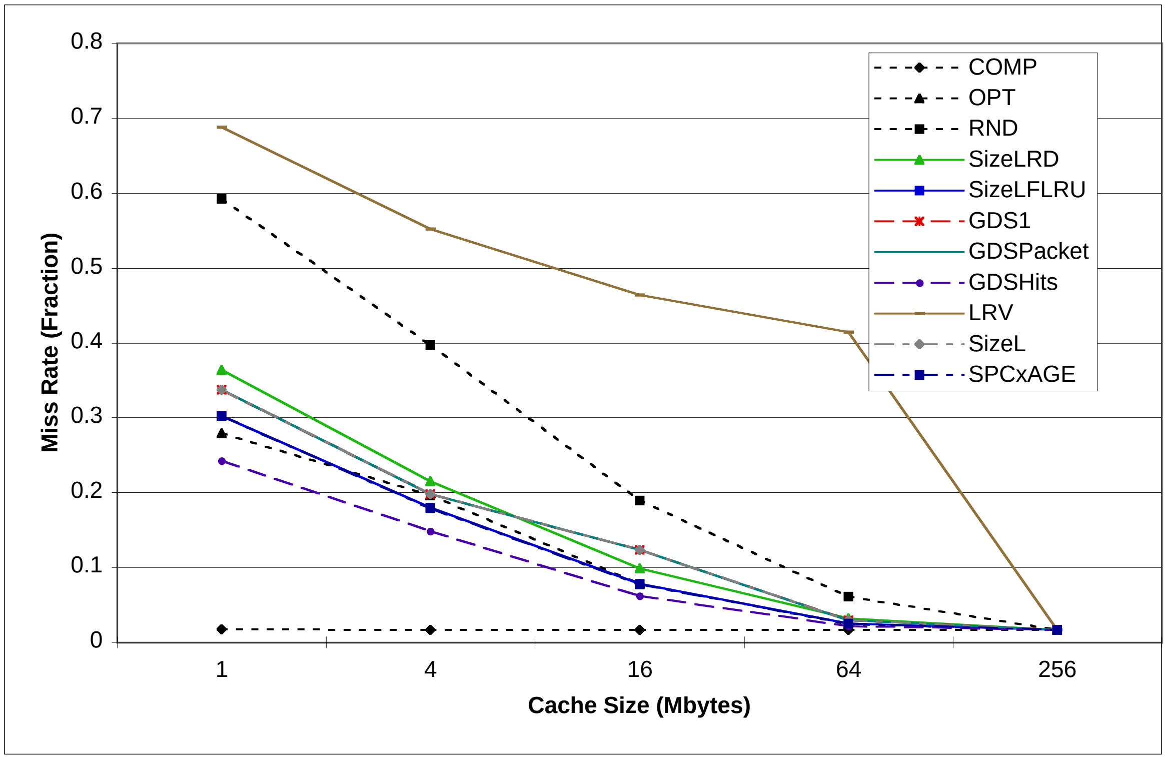


Fig. 6.3. Miss Rates of Complex Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

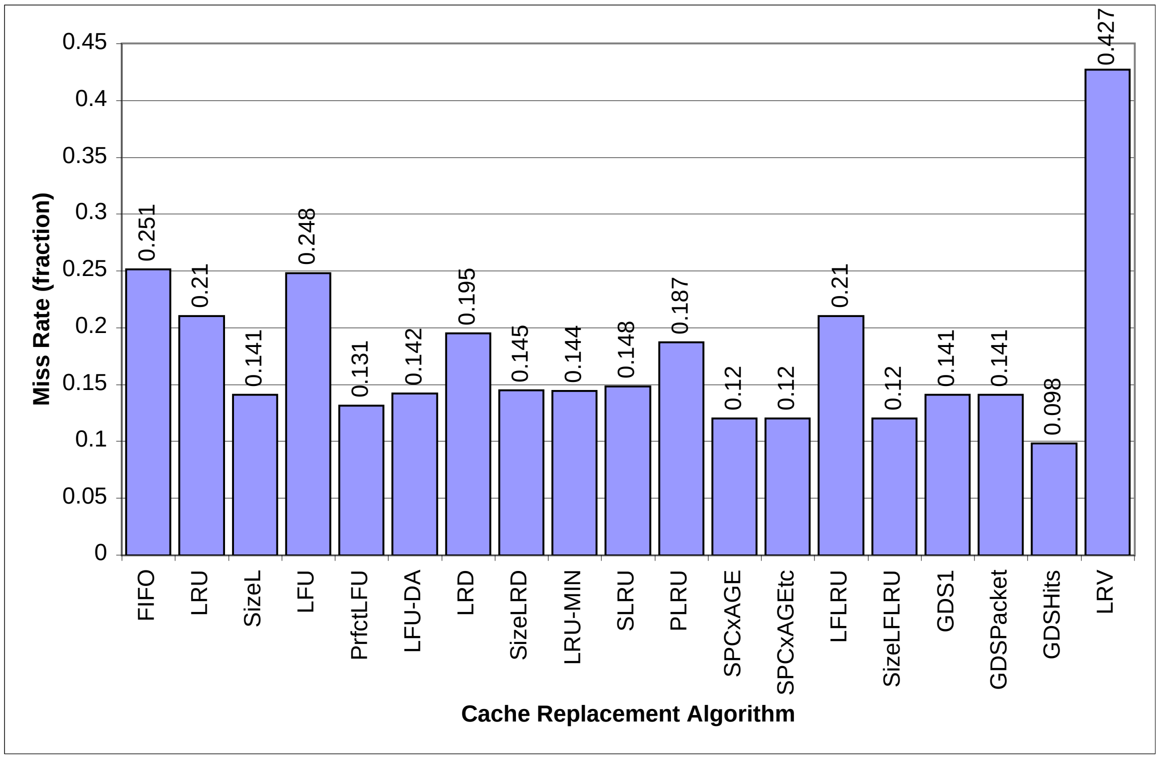


Fig. 6.4. Average Miss Rates on HP World Cup 1998 Web Server Farm wc_week14 Trace.

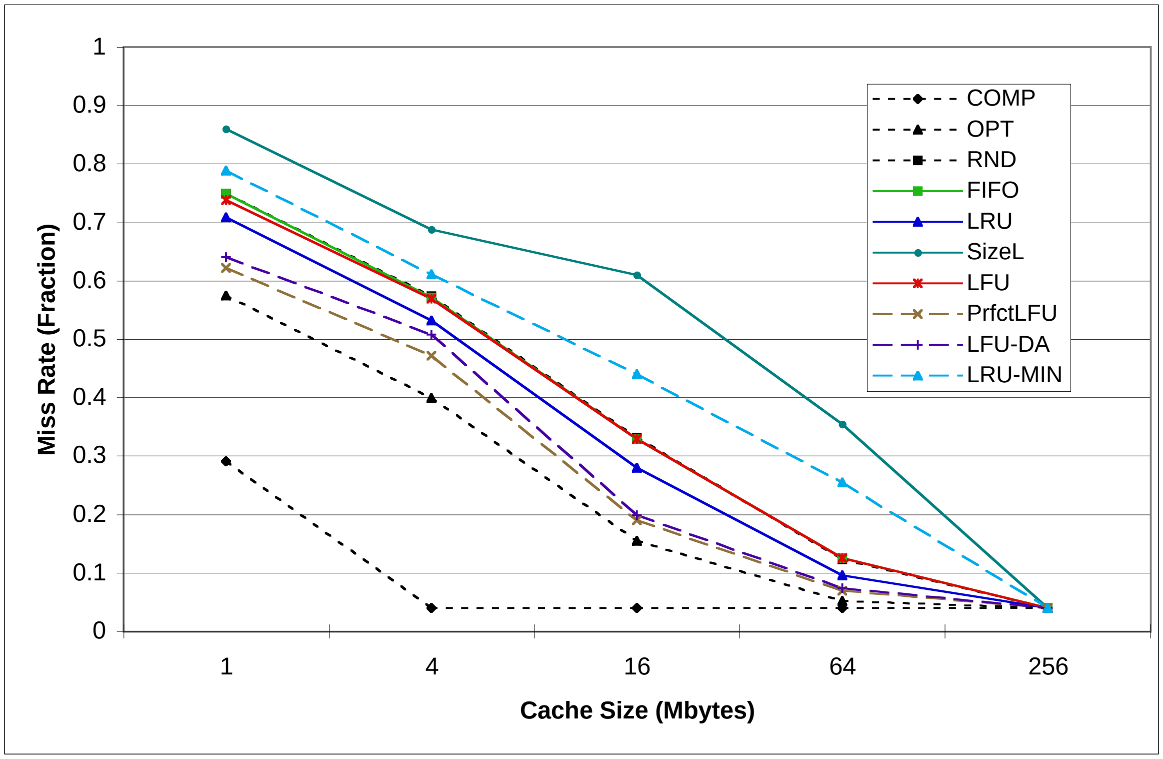


Fig. 6.5. Byte Miss Rates of Simple Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

LFU-DA and LRD performed best among the more complex cache replacement policies in Figure 6.6, and their results are forwarded to the next graph.

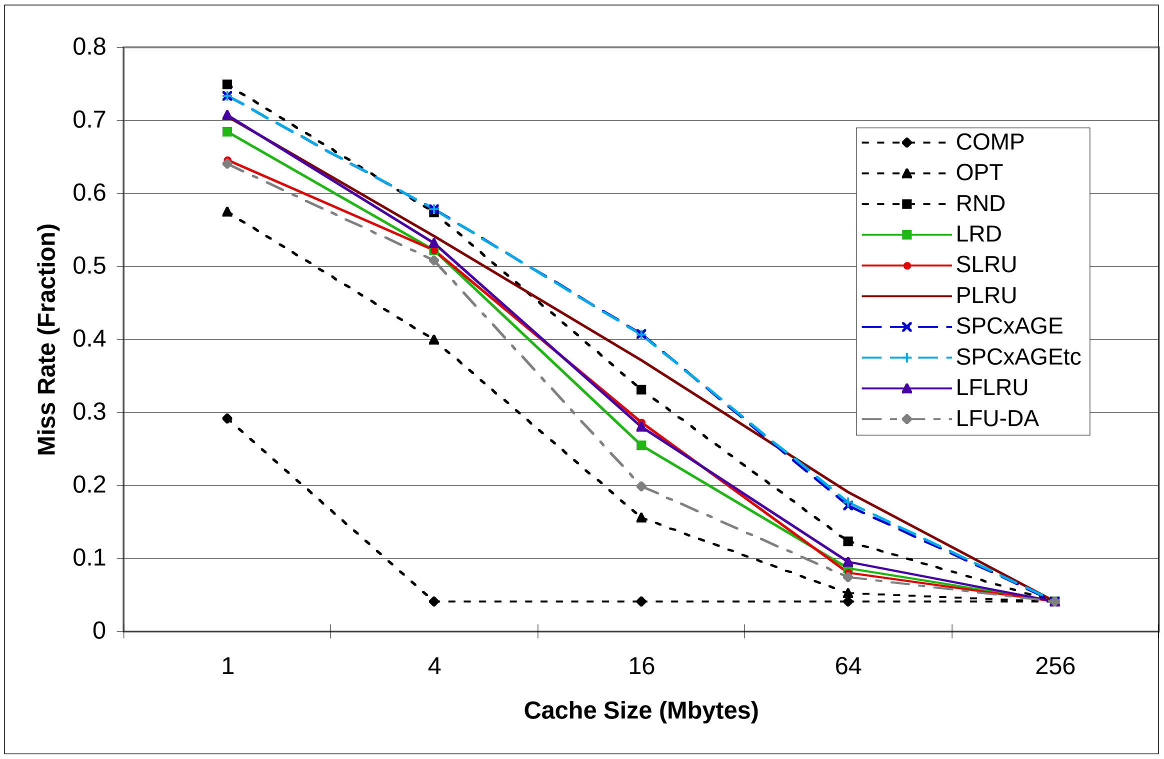


Fig. 6.6. Byte Miss Rates of More Complex Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

Finally, the two forwarded algorithms performed the best among the complex algorithms in Figure 6.7. On the average overview graph of Figure 6.8, the LFU-DA algorithm performed best, while SLRU was second, and LRD was third.

Purdue DSML Web Server Trace

For the simple cache replacement algorithms on the Purdue DSML Web Server dsl_log trace in Figure 6.9, the SizeL algorithm had the lowest miss rate, and its results are forwarded to the next graph. LRU generally beats both LFU and FIFO but it did not better any of the other frequency based algorithms.

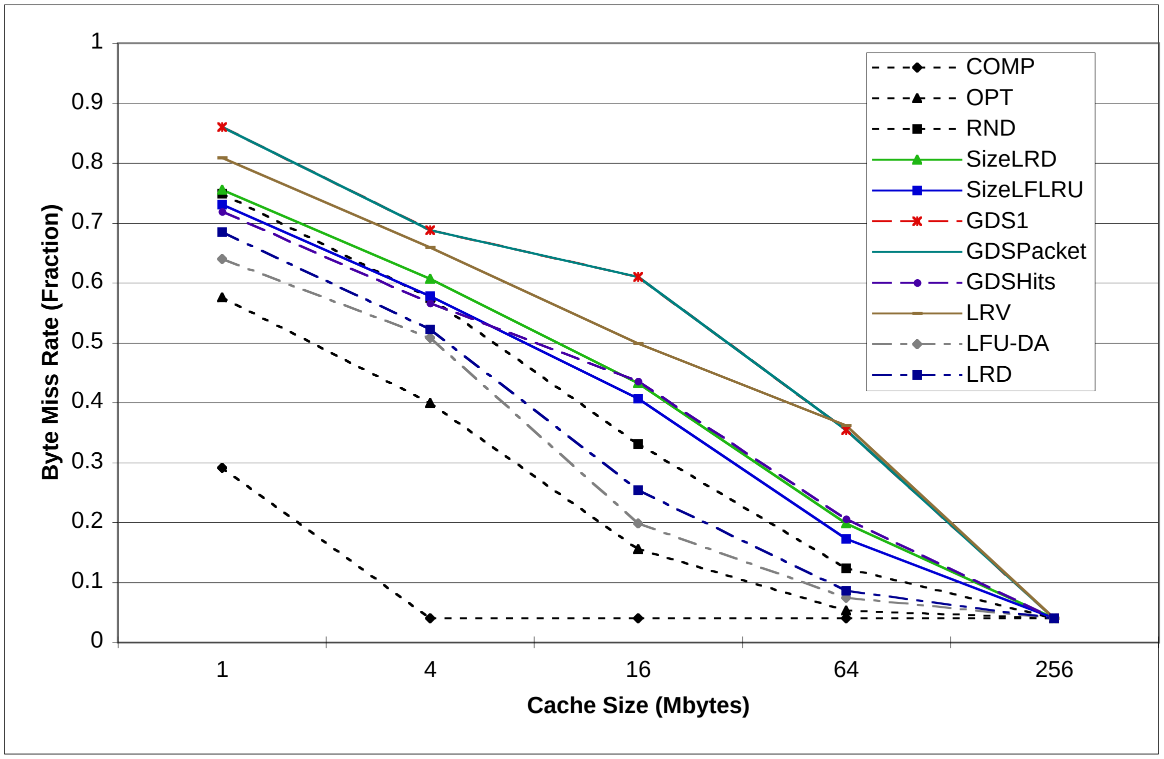


Fig. 6.7. Byte Miss Rates of Complex Cache Replacement Algorithms on HP World Cup 1998 Web Server Farm wc_week14 Trace.

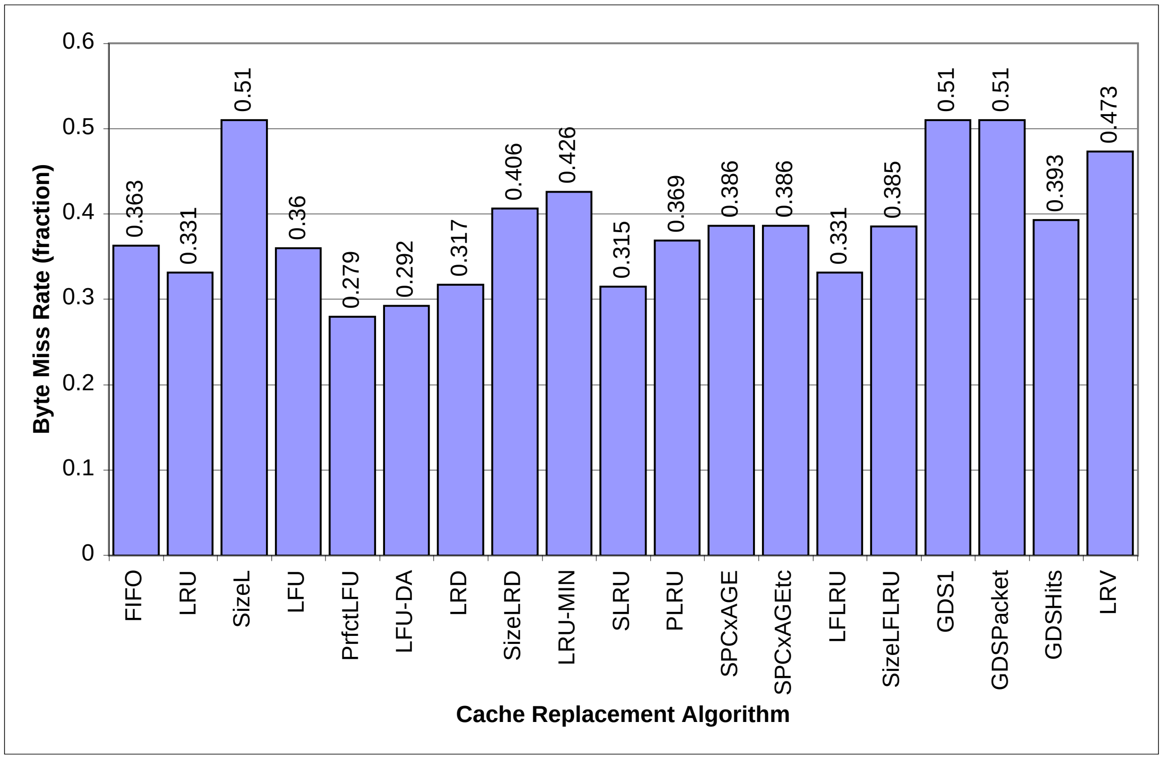


Fig. 6.8. Average Byte Miss Rates on HP World Cup 1998 Web Server Farm wc_week14 Trace.

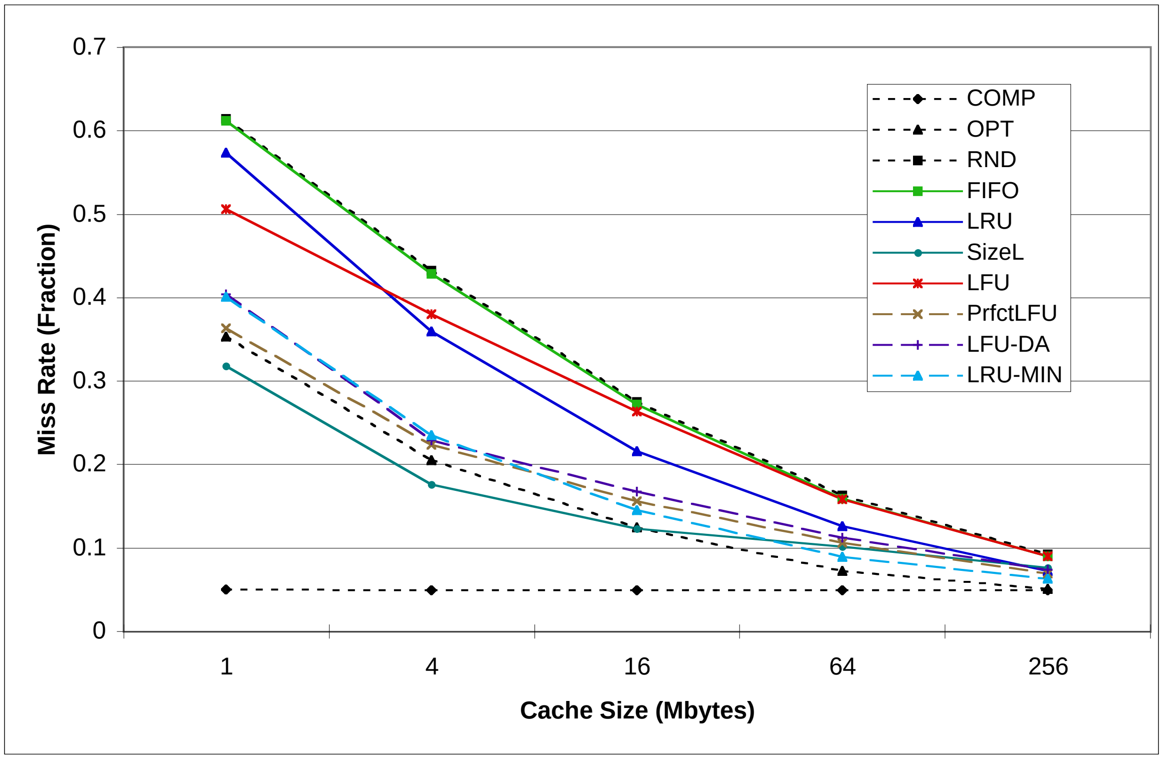


Fig. 6.9. Miss Rates of Simple Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

Both the SizeL and SPACExAGE results are forwarded to the final graph by barely outperforming LFLRU and SPACExAGEtc as shown in Figure 6.10. SizeL performs better on the smaller cache sizes while SPACExAGE performs better on the larger cache sizes.

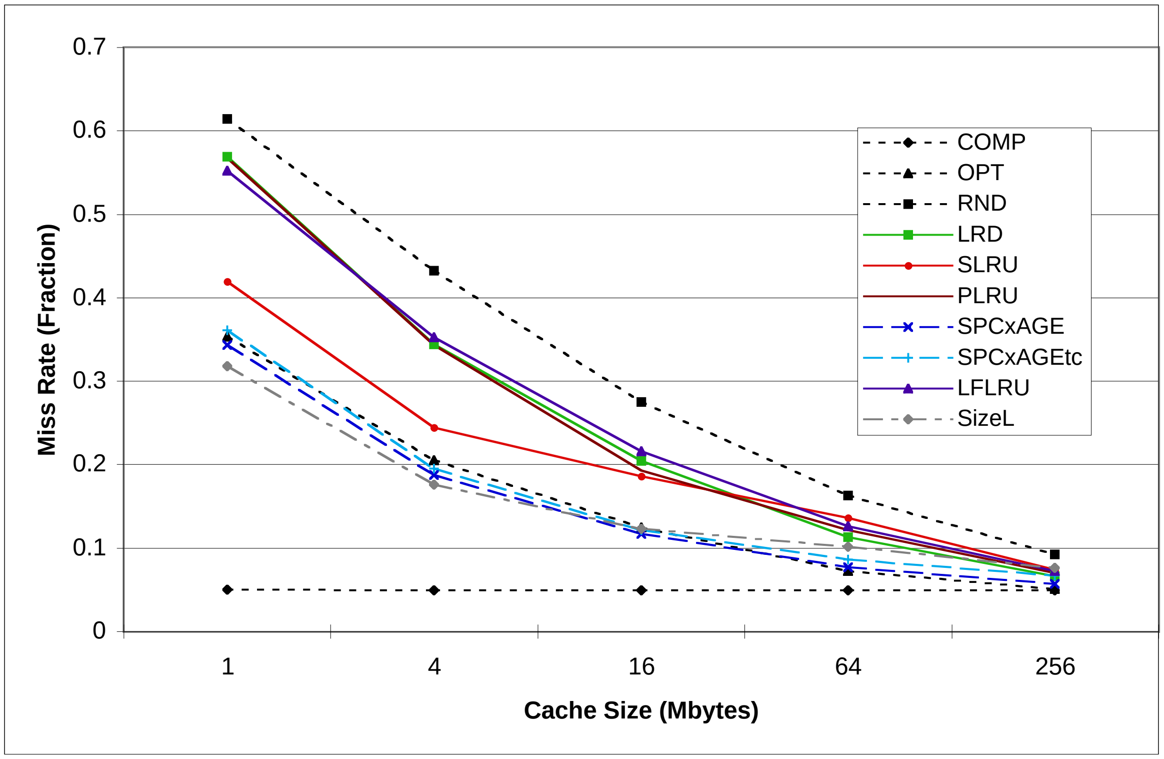


Fig. 6.10. Miss Rates of More Complex Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

In the final miss rate comparison on Figure 6.11, GDSHits outperforms all of the other algorithms by a significant margin for smaller caches. On the average miss rate comparison graph of Figure 6.12, we see that GDSHits performs best with SPACEx-AGE and SizeLFLRU tied for second, and having about 2% higher average miss rate. The fact that SPACExAGE and SizeLFLRU tied indicates that for SizeLFLRU, the frequency component was not as important as the size and LFU components.

On the graph for the simple algorithms for byte miss rate in Figure 6.13, the PerfectLFU algorithm had the best miss rate followed closely by the LFU-DA algorithm.

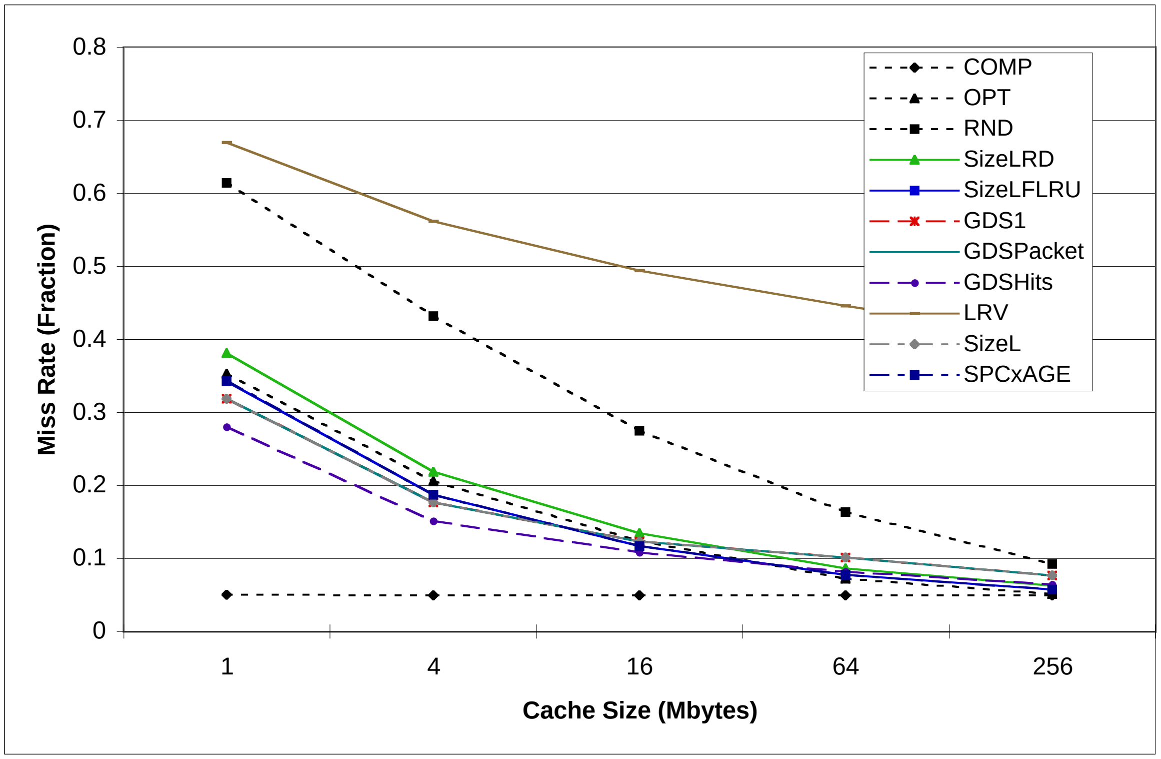


Fig. 6.11. Miss Rates of Complex Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

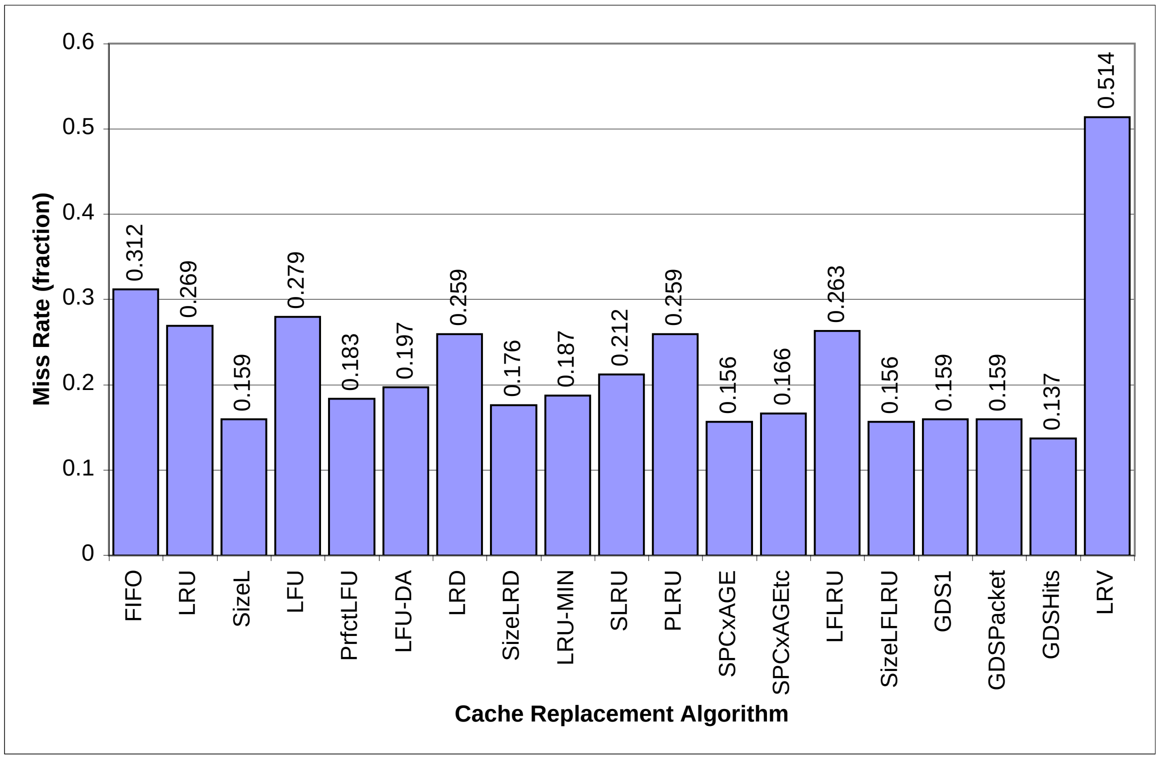


Fig. 6.12. Average Miss Rates on Purdue dsl_log DSML Web Server Trace.

Hence the LFU-DA algorithm results are forwarded to the next line graph.

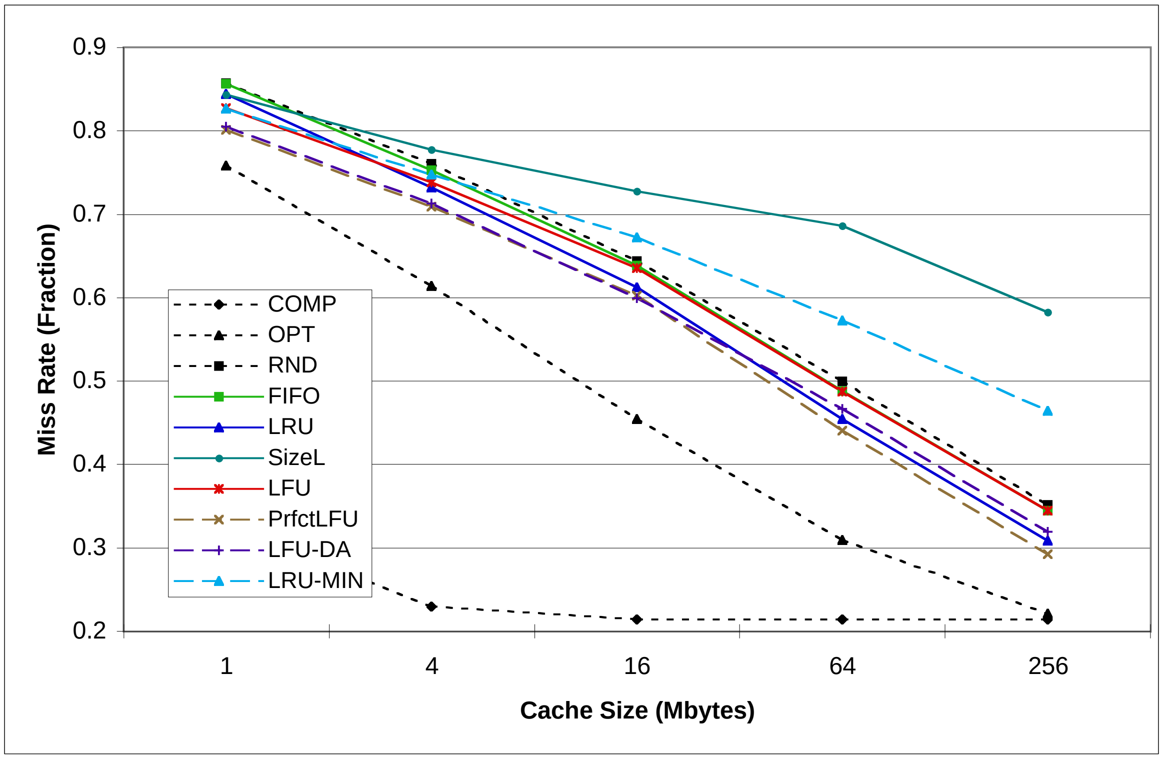


Fig. 6.13. Byte Miss Rates of Simple Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

Among the more complex cache replacement algorithms on Figure 6.14, the LFU-DA algorithm performs best for smaller cache sizes, while the LRD algorithm performs best for larger cache sizes. The results of both of these algorithms are forwarded to the final graph.

Overall, these two forwarded algorithms, LRD placing first and LFU-DA placing second, perform best with LRU placing third on average as shown in Figure 6.16. One should note that SLRU came in fourth with only a 0.1% worse average byte miss rate than LRU.

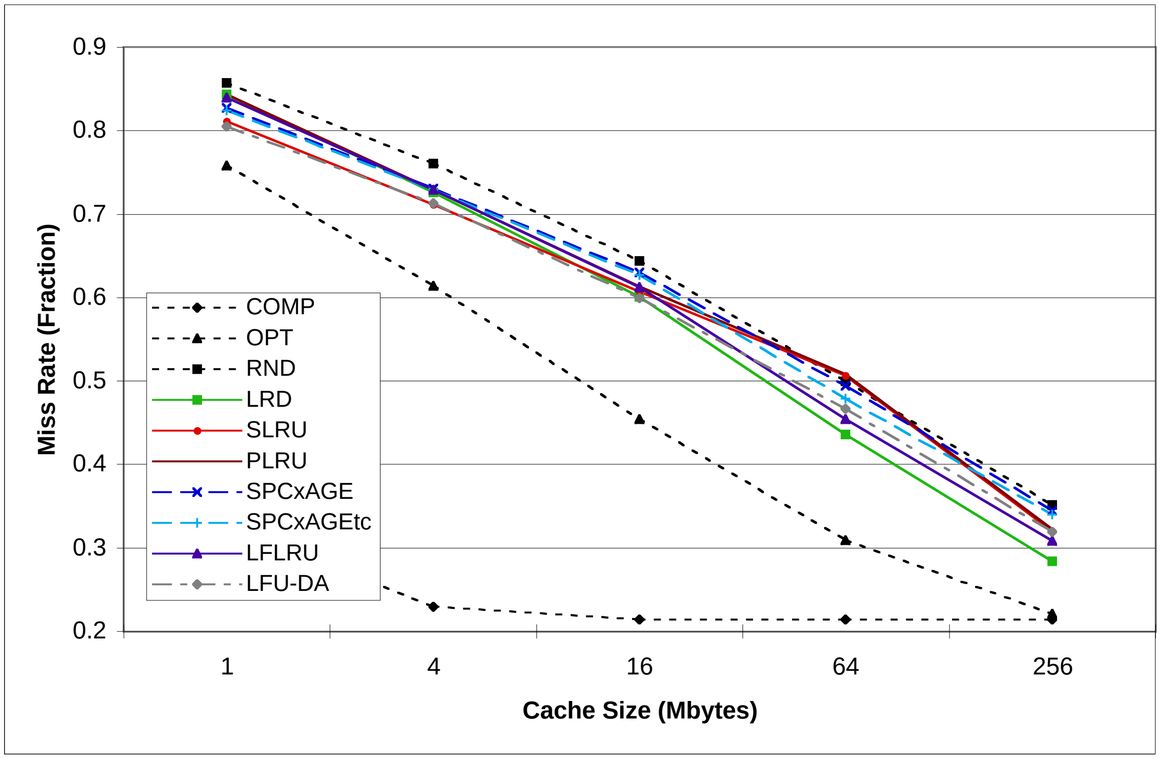


Fig. 6.14. Byte Miss Rates of More Complex Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

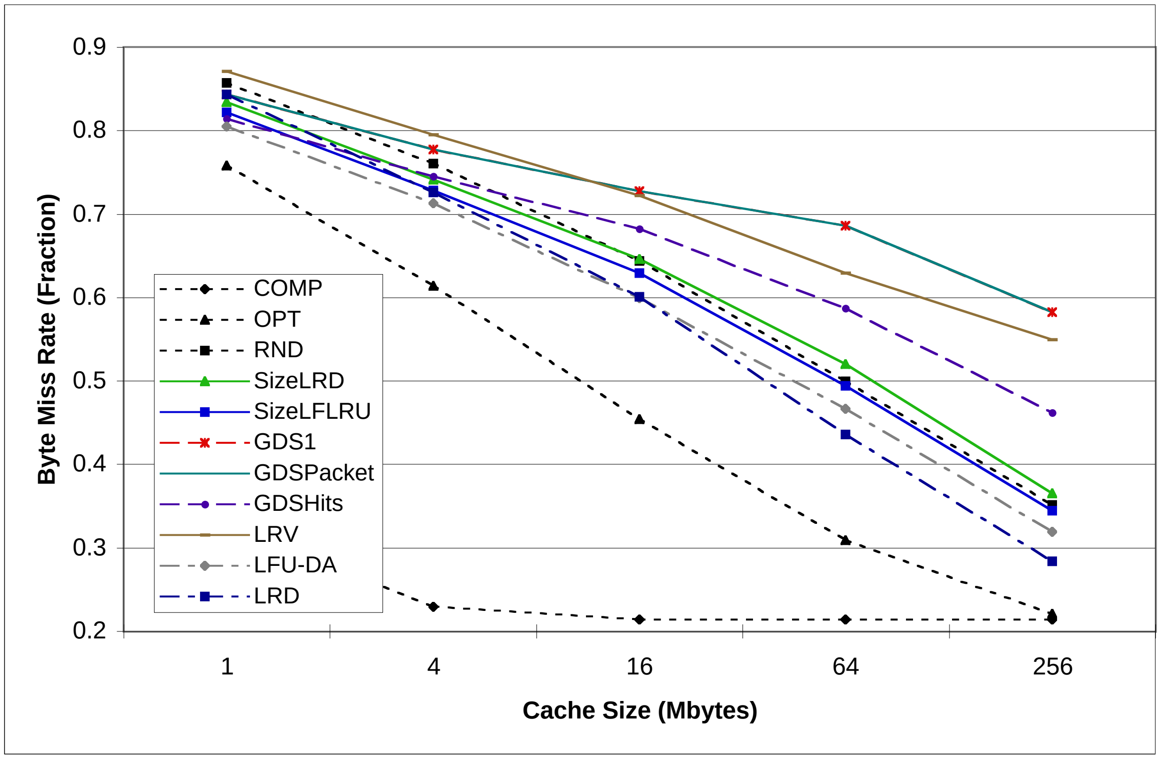


Fig. 6.15. Byte Miss Rates of Complex Cache Replacement Algorithms on Purdue dsl_log DSML Web Server Trace.

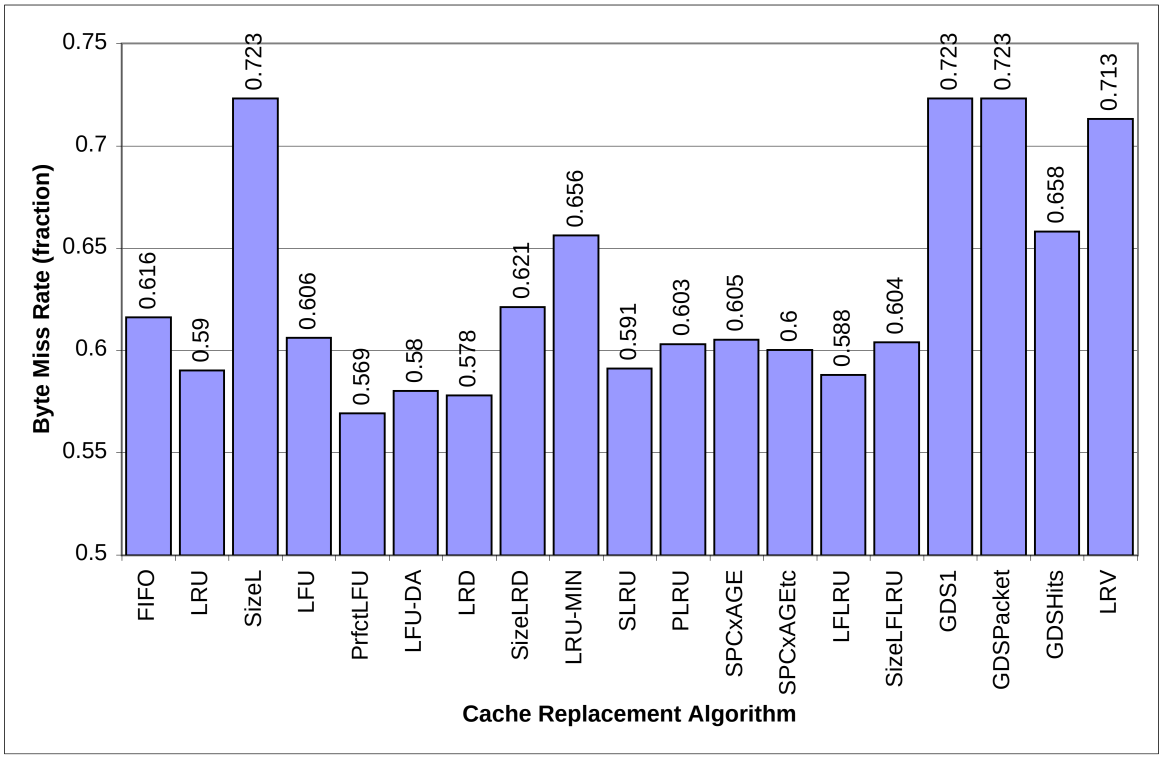


Fig. 6.16. Average Byte Miss Rates on Purdue dsl_log DSML Web Server Trace.

Virginia Tech BR Trace

Among the simple cache replacement algorithms with miss rate as the measure in Figure 6.17, SizeL outperforms all of the other algorithms so its results are forwarded to the next graph.

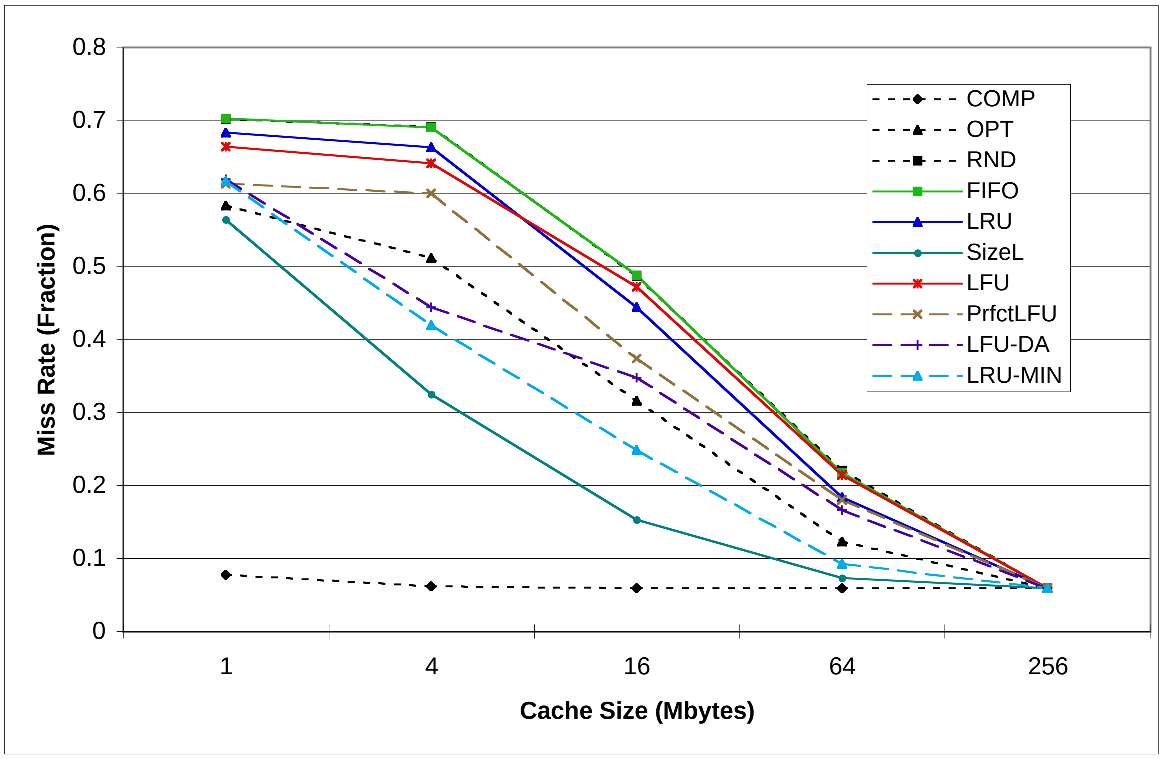


Fig. 6.17. Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech BR Trace.

SPACExAGE and SPACExAGEtc are the top performers among the more complex cache replacement algorithms on the Virginia Tech BR trace in Figure 6.18 so their results are forwarded to the final line graph.

The GDSHits algorithm performs best overall on the Virginia Tech BR trace as shown in Figure 6.19. This is also evident on the average miss rate graph in Figure 6.20. This isn't surprising because GDSHits takes access count, size, and age (with the inflation factor) into account for the replacement decision and it has been

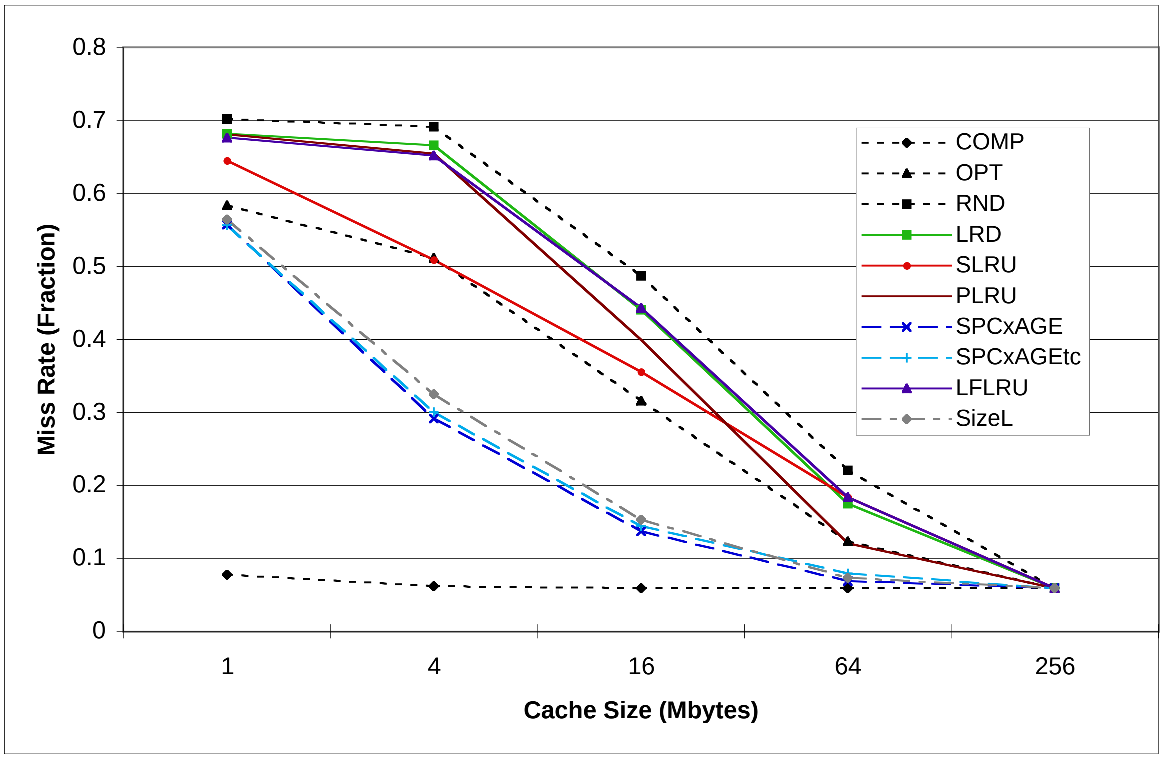


Fig. 6.18. Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech BR Trace.

shown to be quite effective for decreasing miss rates [ACD+99]. The second and third best algorithms are SPACExAGE and SizeLFLRU, respectively, with their average miss rates being around 2.5% more than GDSHits.

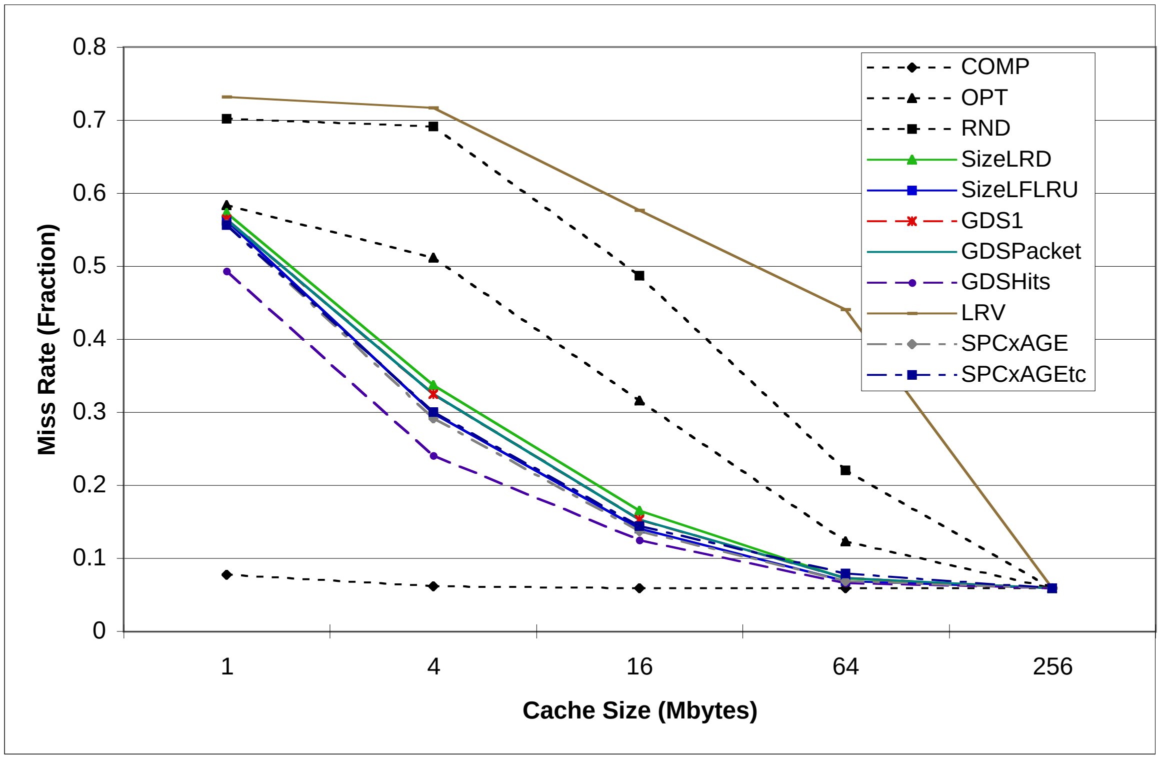


Fig. 6.19. Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech BR Trace.

For the simple cache replacement algorithms with byte miss rate as the measure in Figure 6.21, PerfectLFU is the best by a small margin over LFU-DA. Since the PerfectLFU results are not forwarded, the LFU-DA results are forwarded to Figure 6.22. Note, though, SizeL performs particularly poorly when byte miss rate is the measure. SizeL does not keep the large files that incur a huge cache miss loss in terms of transferring bytes.

Among the more complex replacement policies in Figure 6.22, LFU-DA and SLRU are the top performers, and their results are forwarded to the final graph.

In Figure 6.23, the two forwarded algorithms performed best. All of the complex

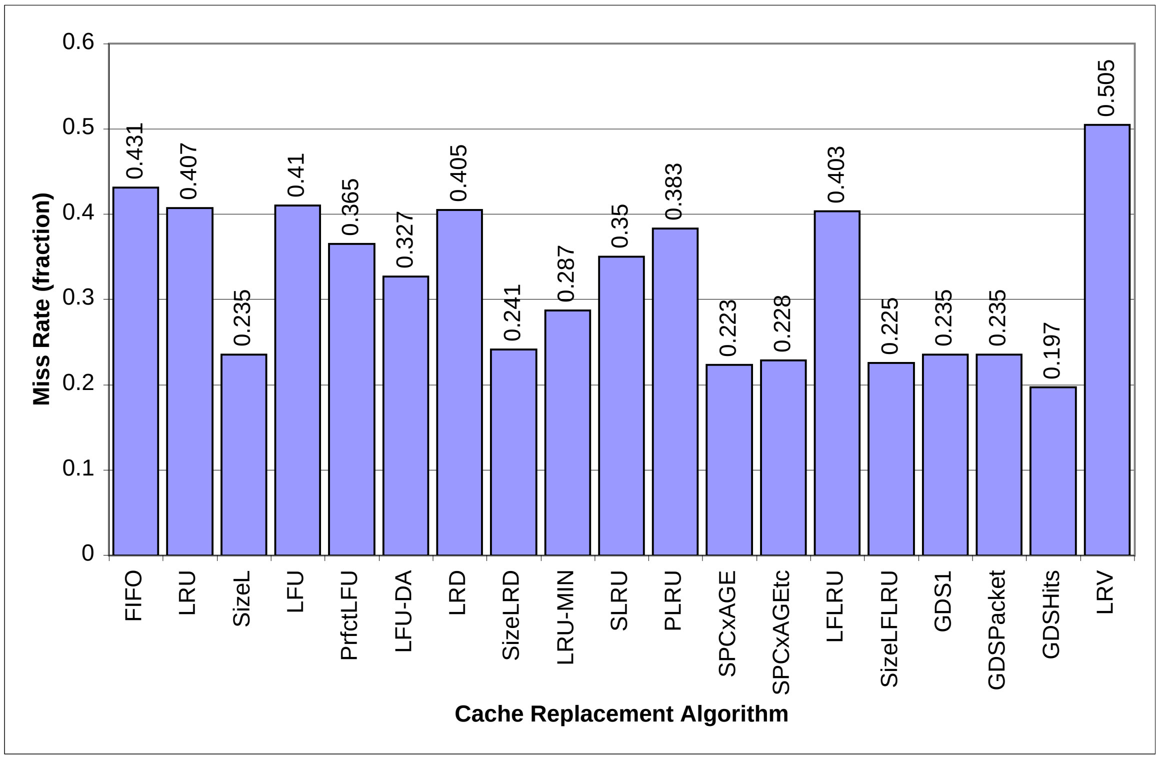


Fig. 6.20. Average Miss Rates on Virginia Tech BR Trace.

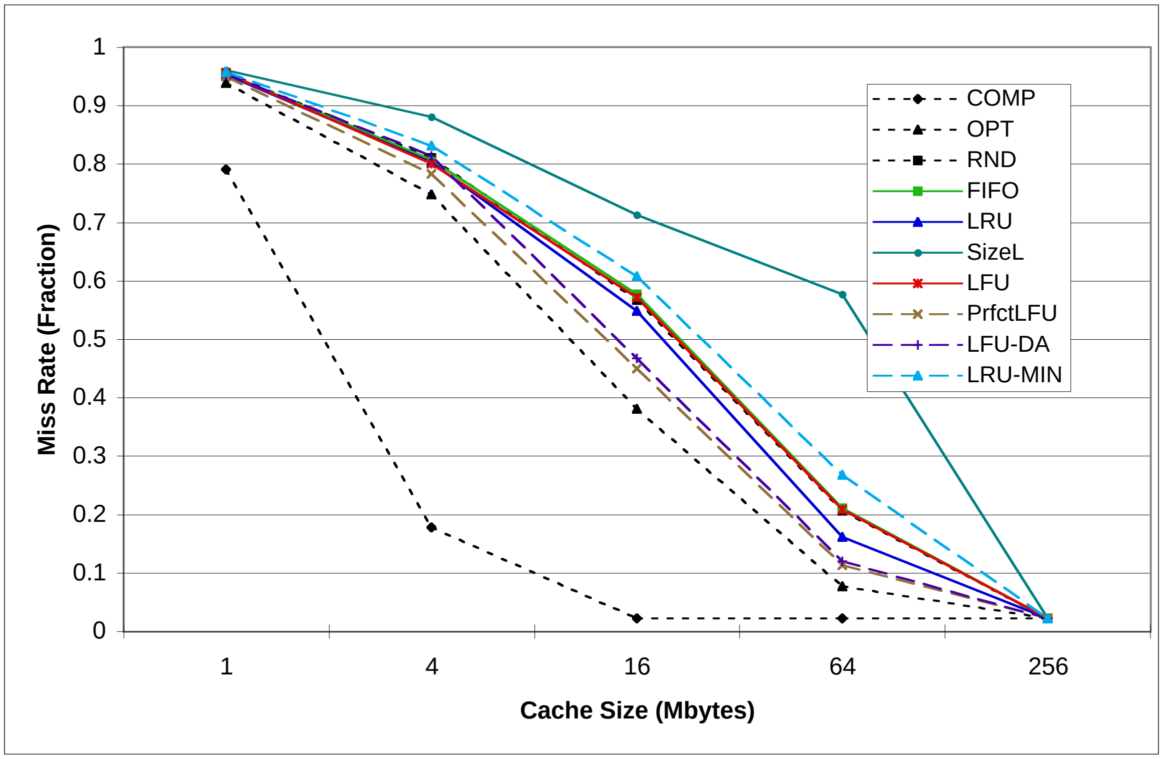


Fig. 6.21. Byte Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech BR Trace.

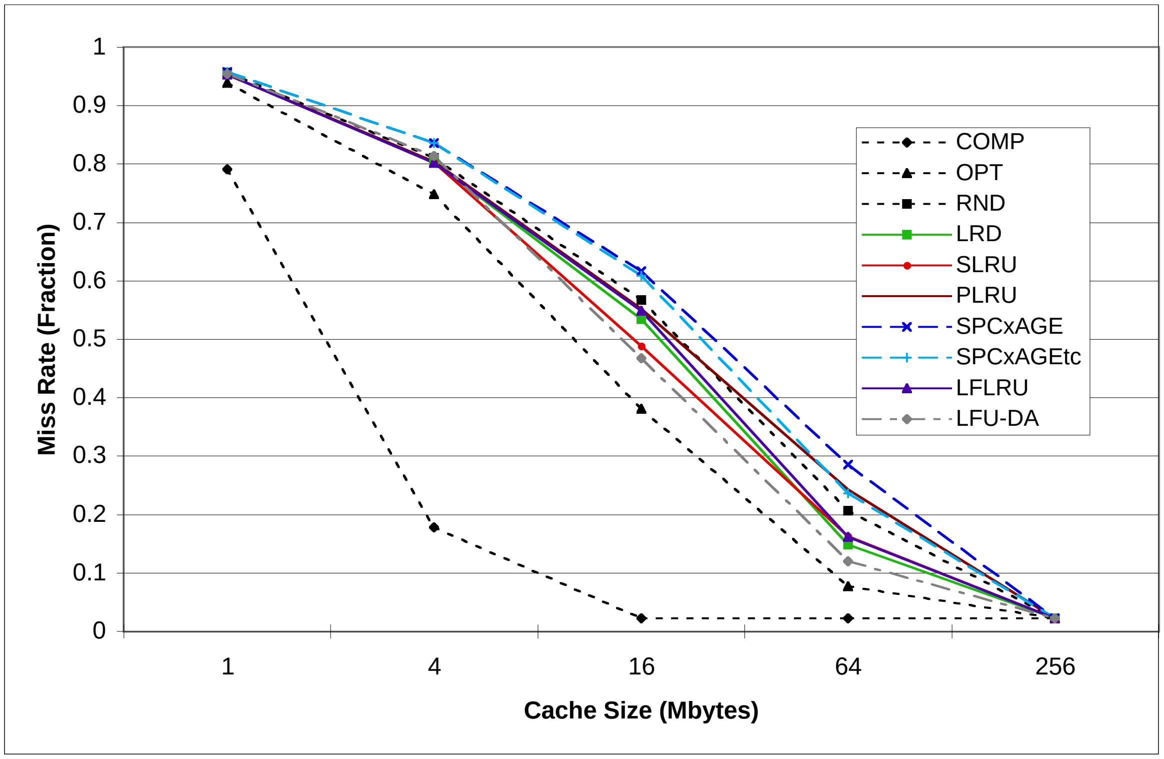


Fig. 6.22. Byte Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech BR Trace.

algorithms do not perform as well as the simpler ones on the Virginia Tech BR trace using byte miss rate as a measure. Figure 6.24 shows that LFU-DA, SLRU, and LRD are the three best algorithms on average for byte miss rate.

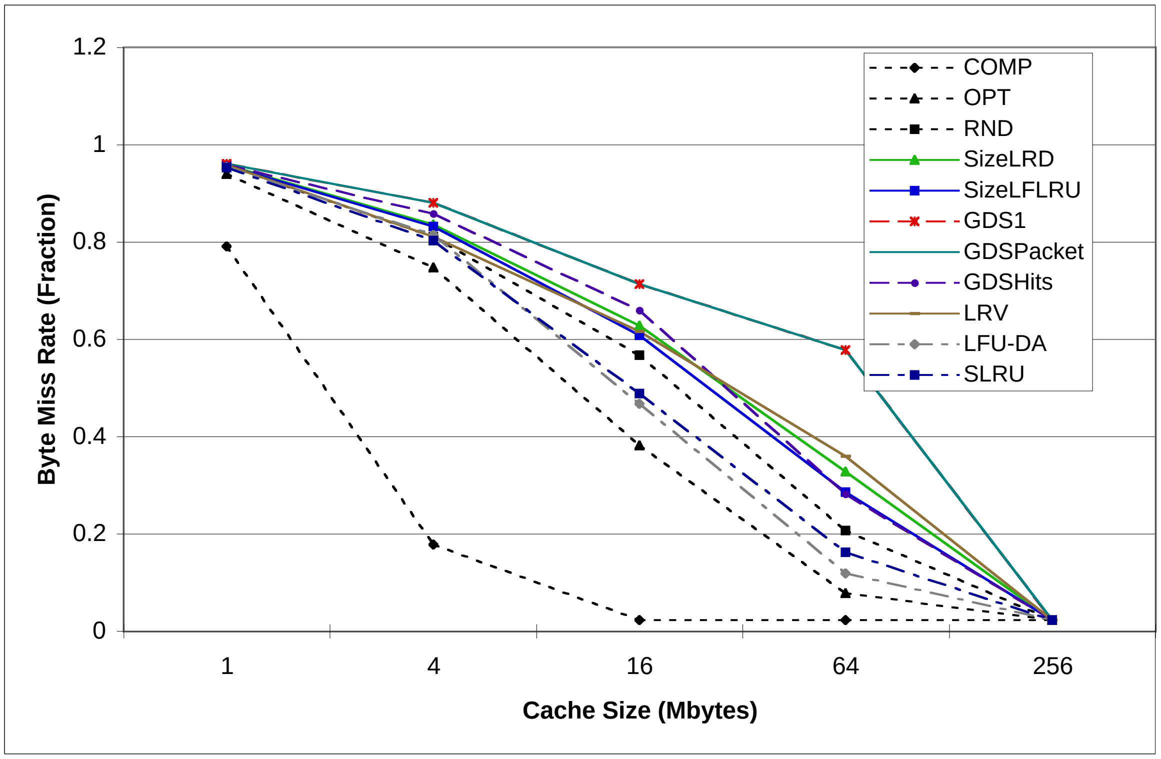


Fig. 6.23. Byte Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech BR Trace.

Origin Server Traces Summary

For this domain's summary, the table of top performing algorithms for each trace and both measures is presented in Table 6.2. In this table, one can see that the best algorithms for this domain are consistently GDSHits when optimizing for miss rate, and LRU-DA when optimizing for byte miss rate. Among the best algorithms for miss rate, SPACExAGE and SizeLFLRU are consistently the next best algorithms, while LRD and SLRU show some consistency among the top contenders for byte miss rate.

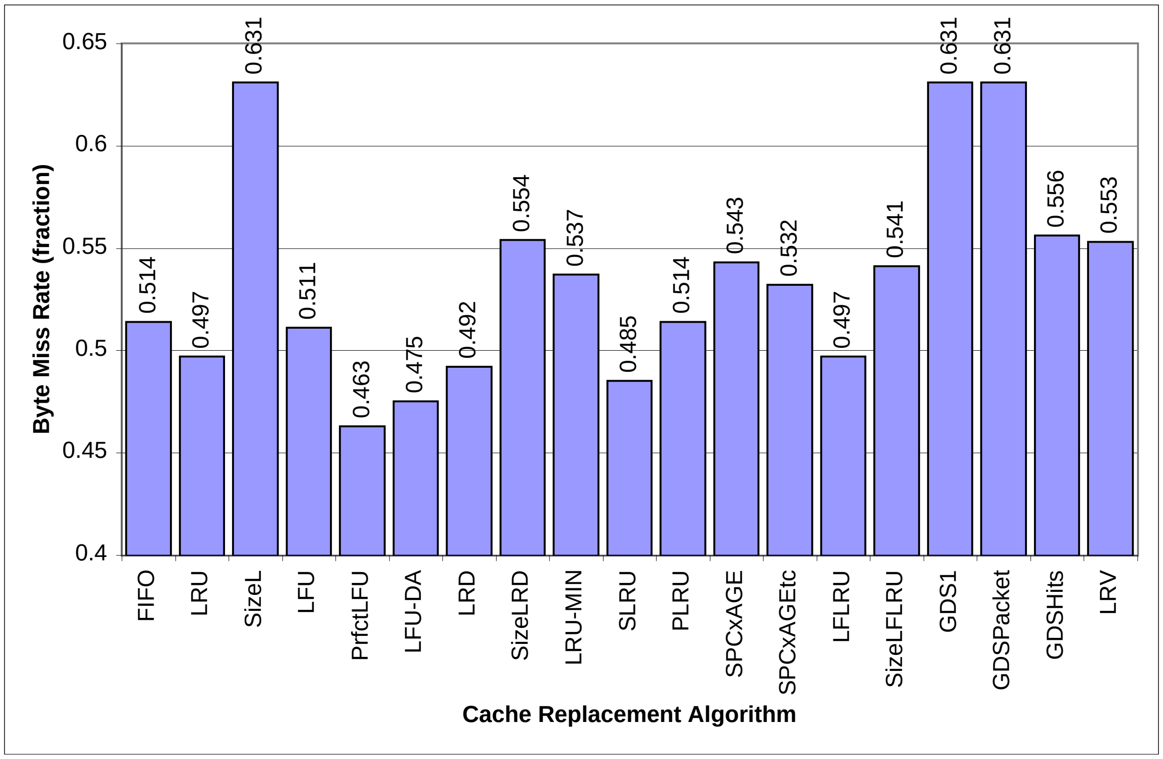


Fig. 6.24. Average Byte Miss Rates on Virginia Tech BR Trace.

Table 6.2
Summary of Top Cache Replacement Algorithms for Origin Proxy Server Traces

Trace	worldcup	dsml	VaTech BR
Miss	GDSHits (1st)	GDSHits (1st)	GDSHits (1st)
Rate	SPACExAGE (2nd-tie)	SPACExAGE (2nd-tie)	SPACExAGE (2nd)
	SPACExAGEtc (2nd-tie)	SizeLFLRU (2nd-tie)	SizeLFLRU (3rd)
	SizeLFLRU (2nd-tie)		
Byte	LFU-DA (1st)	LFU-DA (1st)	LFU-DA (1st)
Miss	SLRU (2nd)	LRD (2nd)	SLRU (2nd)
Rate	LRD (3rd)	LRU (3rd)	LRD (3rd)

Now some generalities will be drawn on the performance of the cache replacement policies for the traces of the origin server domain. The average miss rate graphs of Figures 6.4, 6.12, and 6.20 are shrunk and brought together in Figure 6.25, while Figures 6.8, 6.16, and 6.24 are shrunk and brought together in Figure 6.26. From these graphs, one can see that LRU is among the worst of the algorithms in miss rate performance, but it performs respectably for byte miss rate performance. Therefore, LRU is among the worst choices when implementing an origin server proxy cache when optimizing for miss rate, but is not such a bad choice for byte miss rate. When looking at the performance of size-based and access count based algorithms, the size-based ones are the better performers for miss rate. Among the origin server proxies, the access count for cached objects is not as good of a criteria for miss rate as shown by the relatively poorer performance of the access count based algorithms, LFU, LFU-DA, and PerfectLFU. So for these traces, a better indicator of being reaccessed is the size of the object. Among the access count (frequency) based algorithms, LFU-DA outperforms both LFU and PerfectLFU, which further suggests that the access count isn't a good replacement indicator for miss rate since LFU and PerfectLFU use only frequency while LFU-DA includes the inflation factor to age objects. Also, when looking at the performance of LRD versus SizeLRD and LFLRU versus SizeLFLRU, the two latter algorithms always outperform the former two by a large margin. For byte miss rate, the access count statistic seems to be a marginally better indicator for replacement.

FIFO is generally only marginally improved upon when it is augmented by the access count and size in the LRD and SizeLRD, and the average results of SizeLRD for byte miss rate is actually worse than for both FIFO and LRD. This implies that using the first-accessed object statistic is not beneficial. The fact that SPACExAGE and SPACExAGEtc perform within 1% of each other on average implies that the difference between measuring the number of other accesses since an object was requested and actual time that has elapsed since an object was accessed is almost negligible. When there was a difference between the performance of the two algorithms, though,

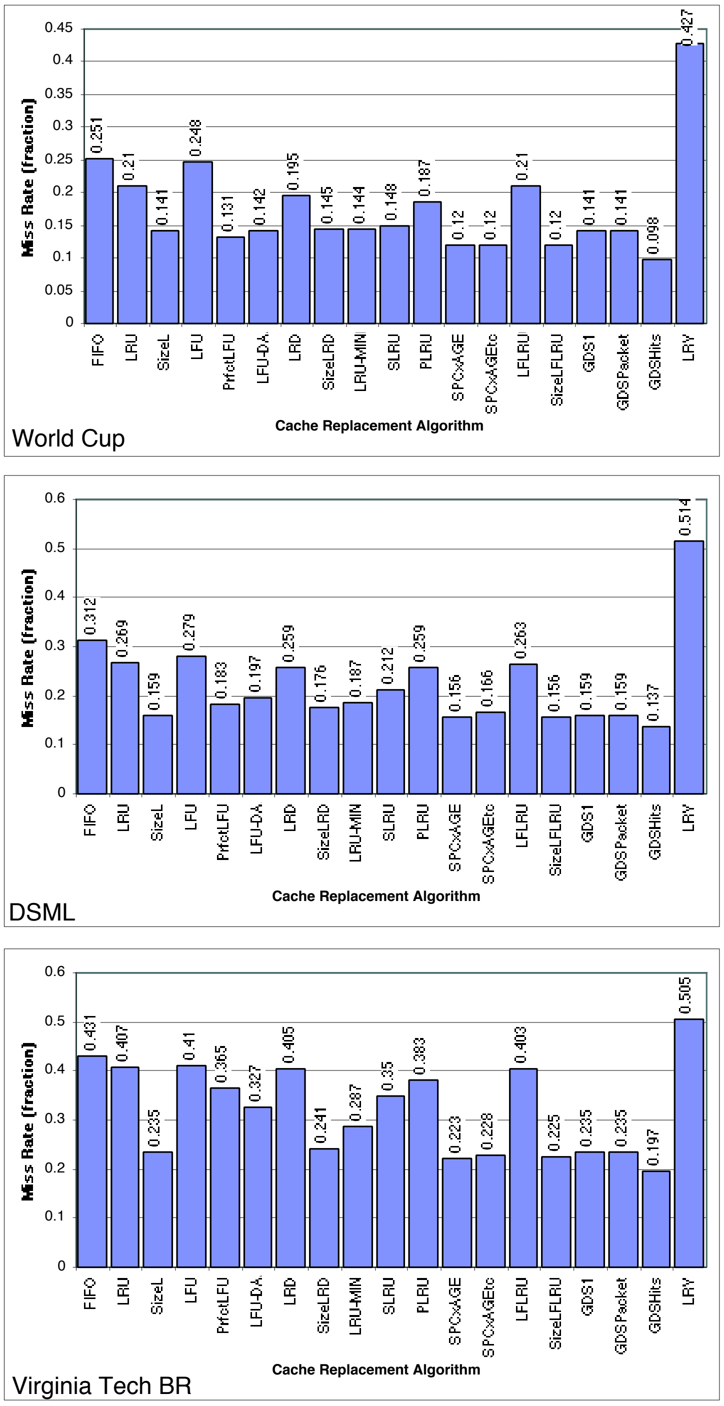


Fig. 6.25. Composite of Average Miss Rates for Origin Server Traces.

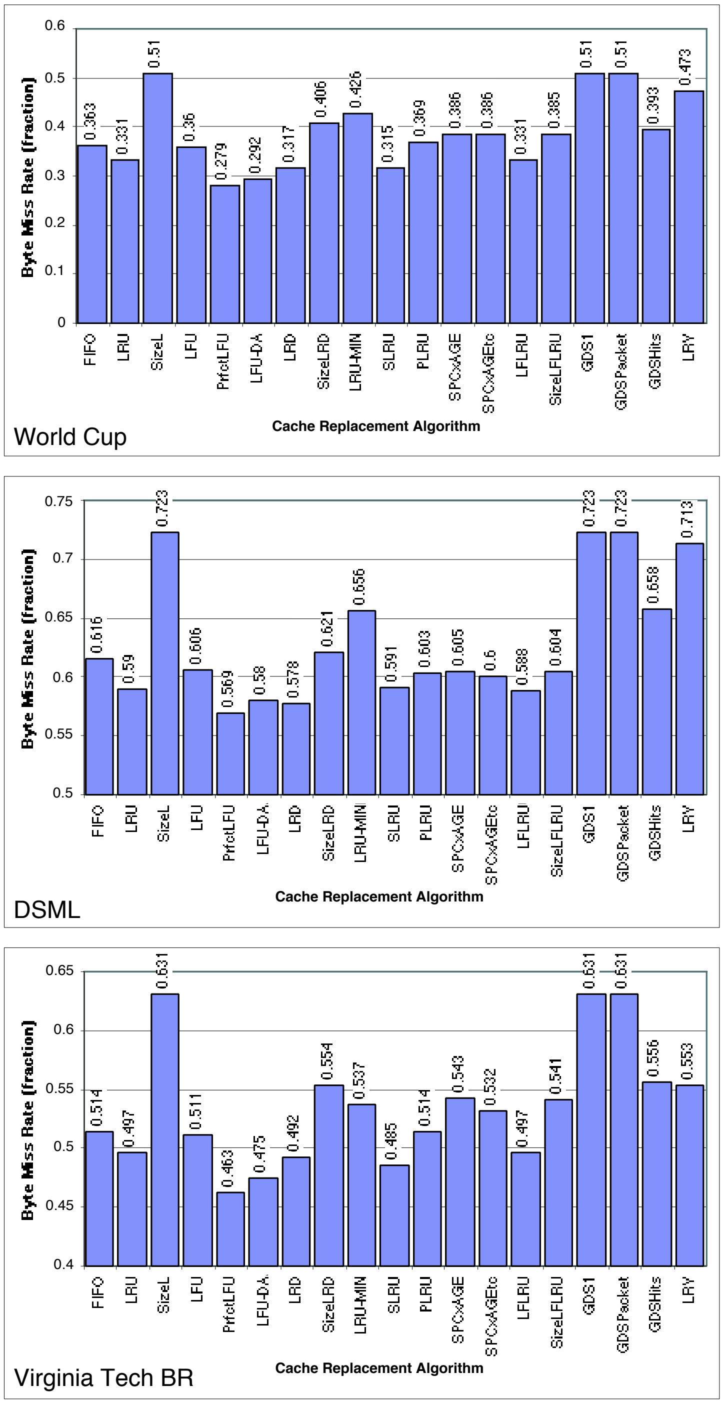


Fig. 6.26. Composite of Average Byte Miss Rates for Origin Server Traces.

SPACExAGE always performed better than SPACExAGEtc for miss rate, and the opposite for byte miss rate.

Finally, the inflation parameter algorithms of LFU-DA, GDS1, GDSPacket, and GDSHits perform quite well for miss rate with GDSHits being the top algorithm. On the other hand, the GreedyDual-Size algorithms perform poorly for byte miss rate, while LFU-DA is the top performing algorithm for byte miss rate. LRV consistently performed poorly for both miss rates and byte miss rates on all of the origin server traces.

6.1.2 Network Proxy Cache Server Traces

NLANR Boulder1 Proxy Server Trace

Before discussing the results of the Boulder1 Proxy Server trace, please note that on the graphs associated with the NLANR Boulder1 Proxy Server Trace the 256 megabyte simulation was not completed due to simulation system memory constraints. However, the results surely would scale up to the 256 megabyte system so all results and conclusions that are gleaned from these graphs apply for larger cache systems.

From the simple cache replacement policies in Figure 6.27, the SizeL cache replacement policy performed best, so its results are forwarded to Figure 6.28.

In Figure 6.28, the SPACExAGE and SPACExAGEtc algorithms performed equally well and were the best algorithms, so their performance results are forwarded to Figure 6.29.

From Figure 6.29, one can see that the GDSHits algorithm performs best, especially with the smaller cache sizes. And from Figures 6.29 and 6.30, there is a three way tie for second place between the SPACExAGE, SPACExAGEtc, and SizeLFLRU algorithms. This three way tie suggests that the access count portion of the SizeLFLRU algorithm does not have much influence on the overall performance of that algorithm. And this should be expected since the percentage of objects that are accessed more than once is fairly low as evidenced in Table 5.64 and to some extent in Table 5.69. (In the latter table, the significant evidence is comparing the

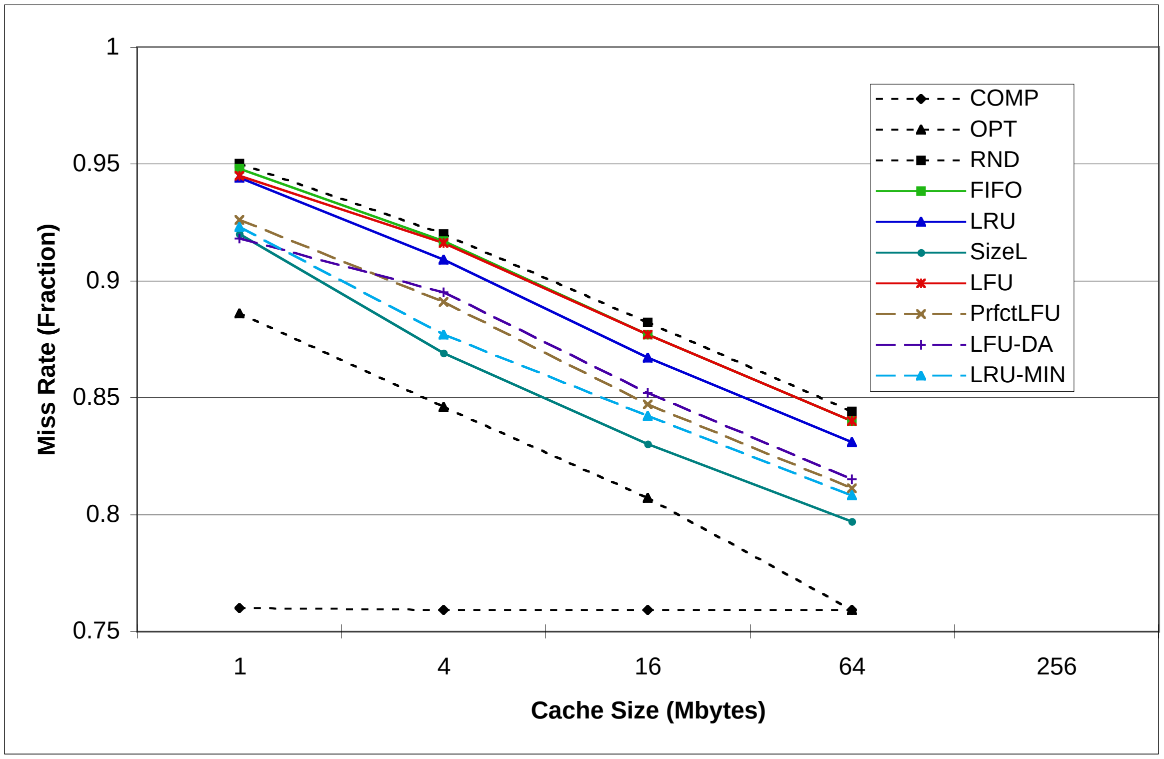


Fig. 6.27. Miss Rates of Simple Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

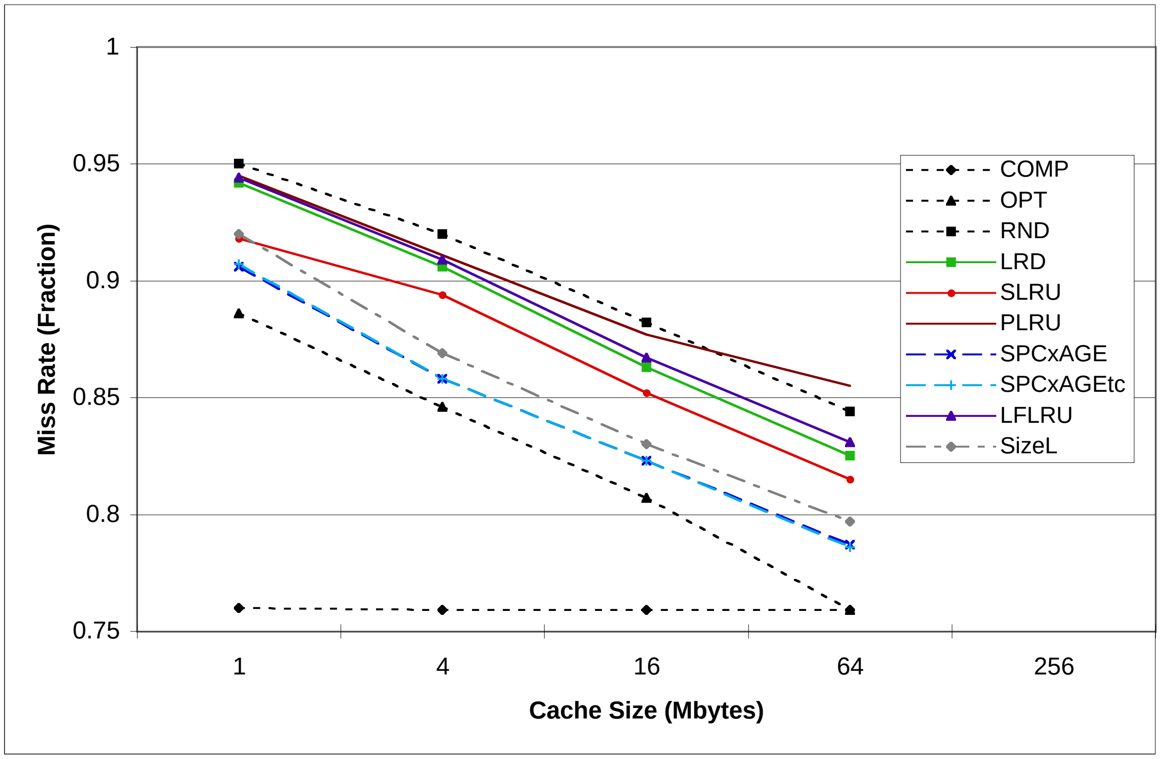


Fig. 6.28. Miss Rates of More Complex Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

First Access column to the Second Access column.)

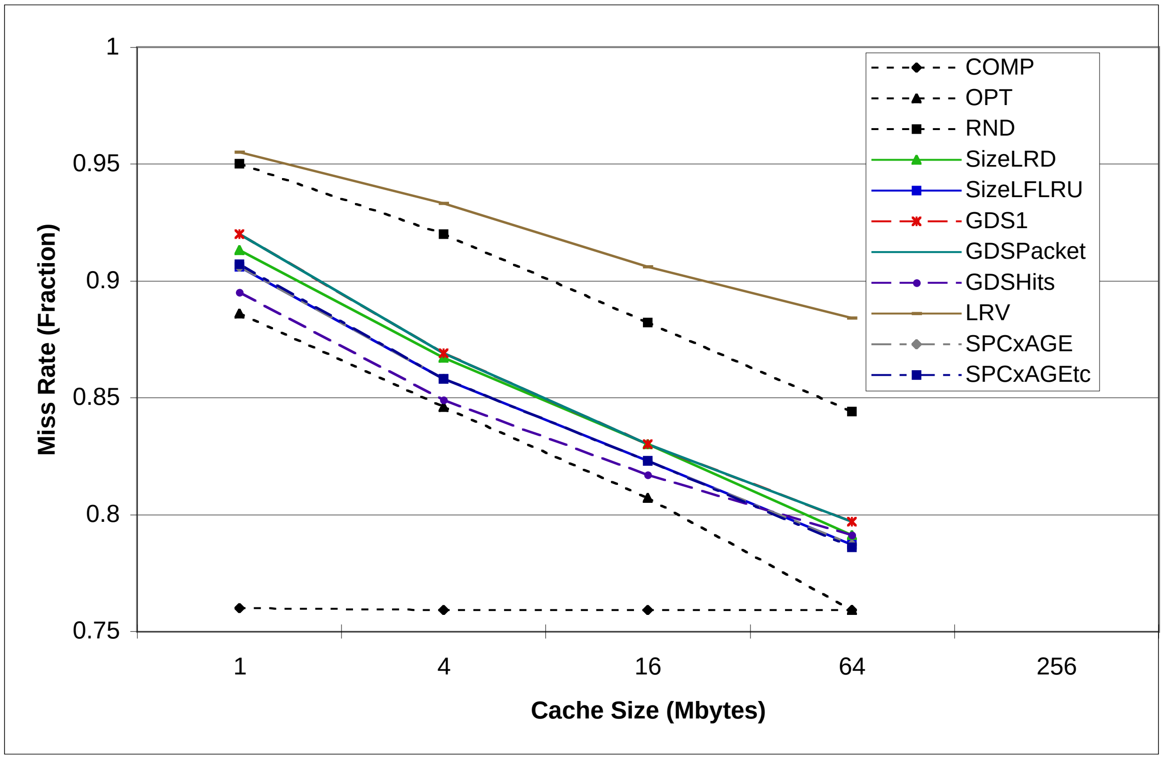


Fig. 6.29. Miss Rates of Complex Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

When using byte miss rate as the measure among the simple cache replacement algorithms in Figure 6.31, the LFU-DA algorithm performed best, so its results are forwarded to Figure 6.32.

In Figure 6.32, the LFU-DA algorithm is also the best while the SLRU algorithm is also very competitive. The results of these two algorithms are forwarded to Figure 6.33.

Now in Figure 6.33, it can be seen that the two forwarded algorithms, LFU-DA and SLRU, perform the best and second best, respectively. In Figure 6.34, the third best algorithm for byte miss rate is LFLRU. Surprisingly, LRV is actually somewhat competitive for this trace and the byte miss rate measure; its parameters must have been set to work well for this type of trace.

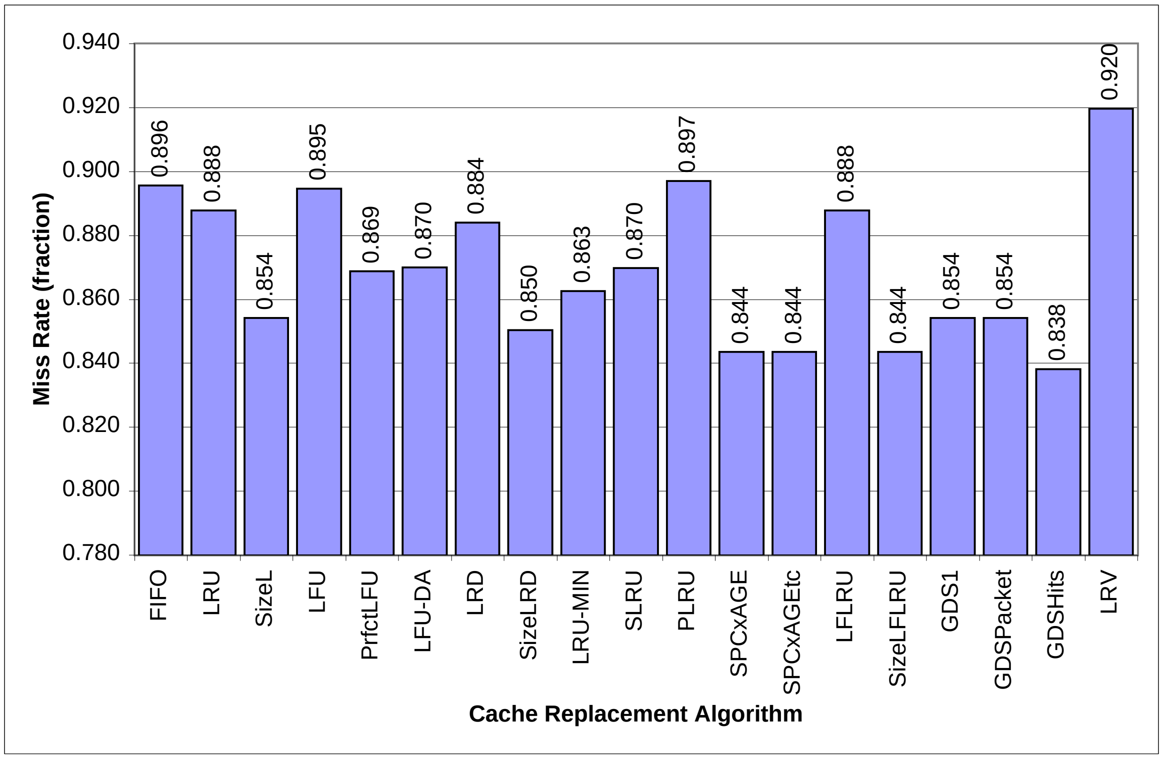


Fig. 6.30. Average Miss Rates on Boulder1 Proxy Server bo1_day1 Trace.

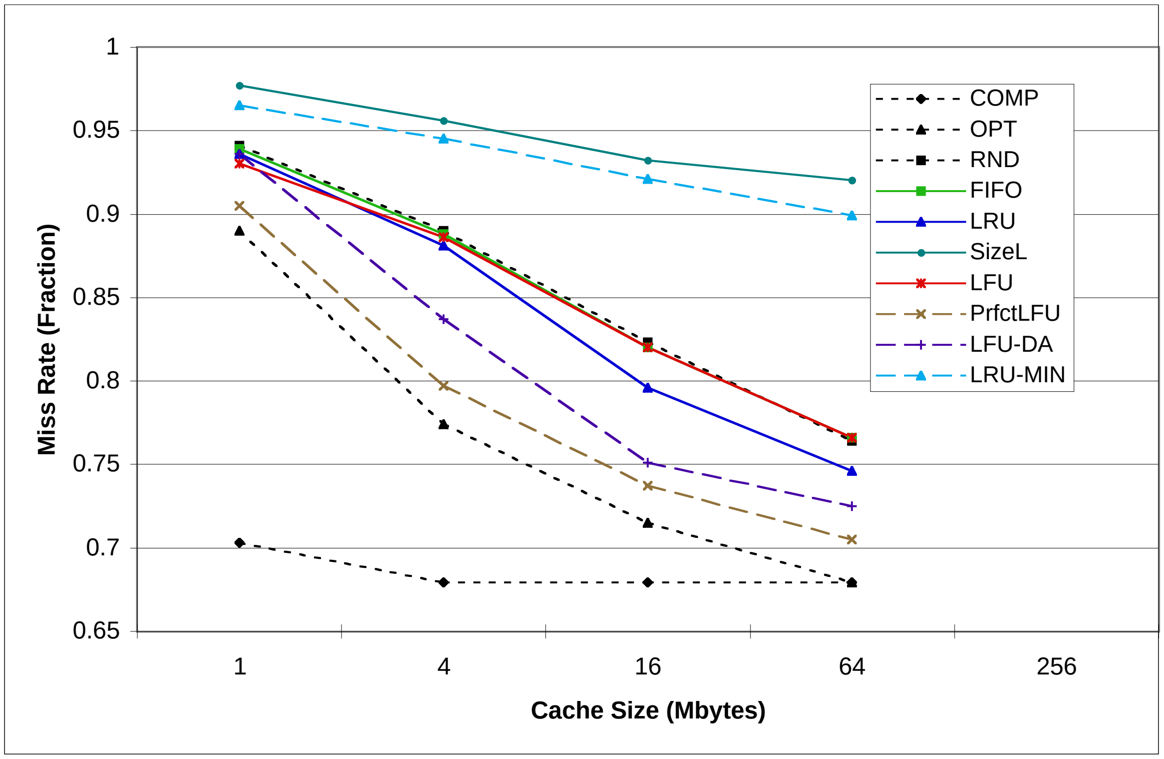


Fig. 6.31. Byte Miss Rates of Simple Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

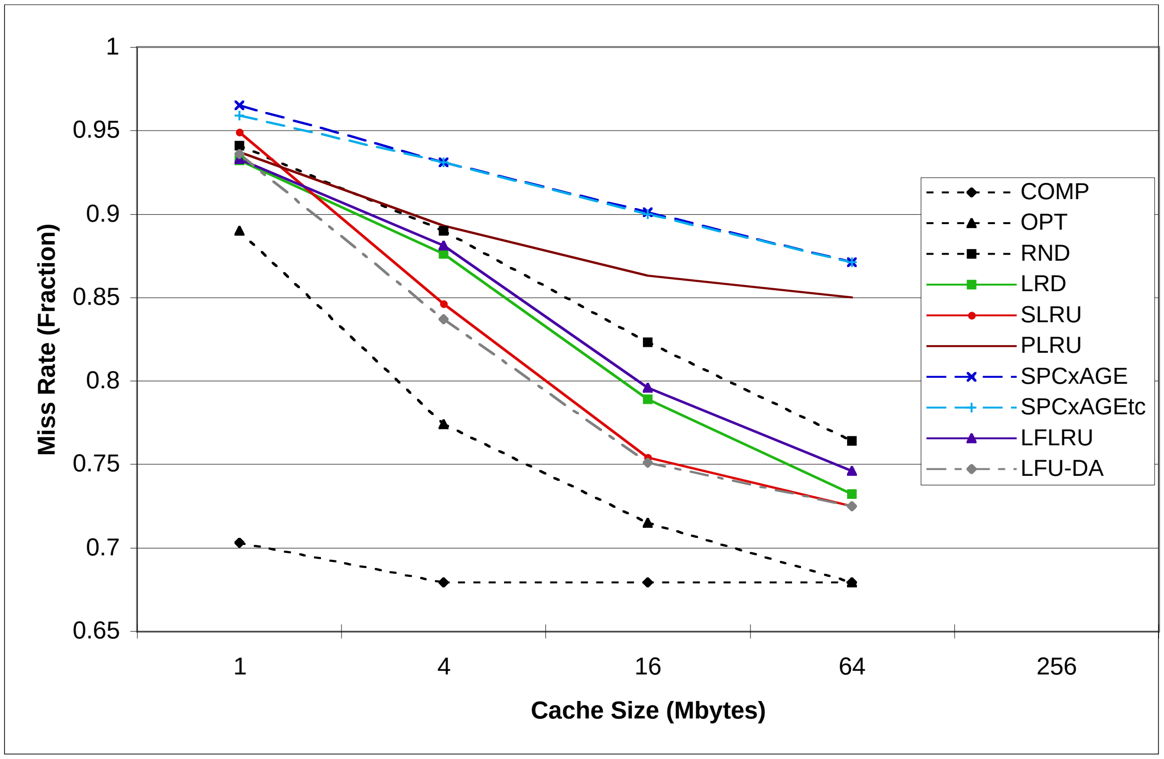


Fig. 6.32. Byte Miss Rates of More Complex Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

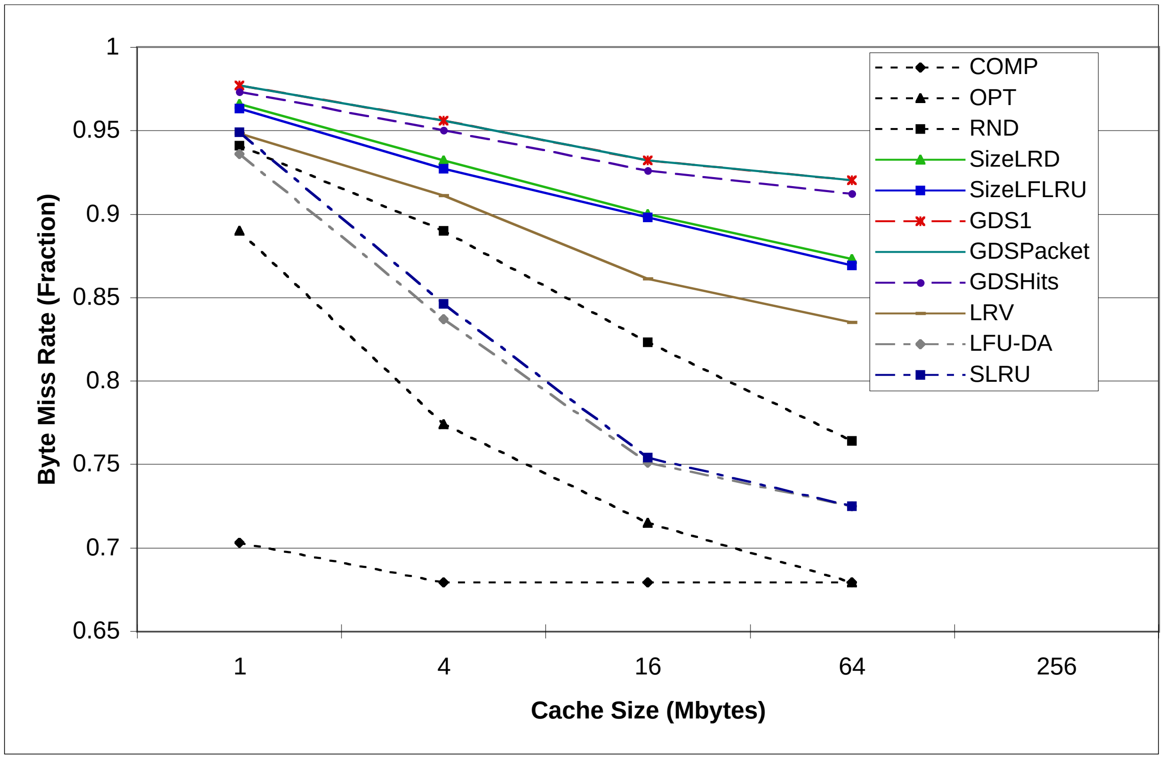


Fig. 6.33. Byte Miss Rates of Complex Cache Replacement Algorithms on Boulder1 Proxy Server bo1_day1 Trace.

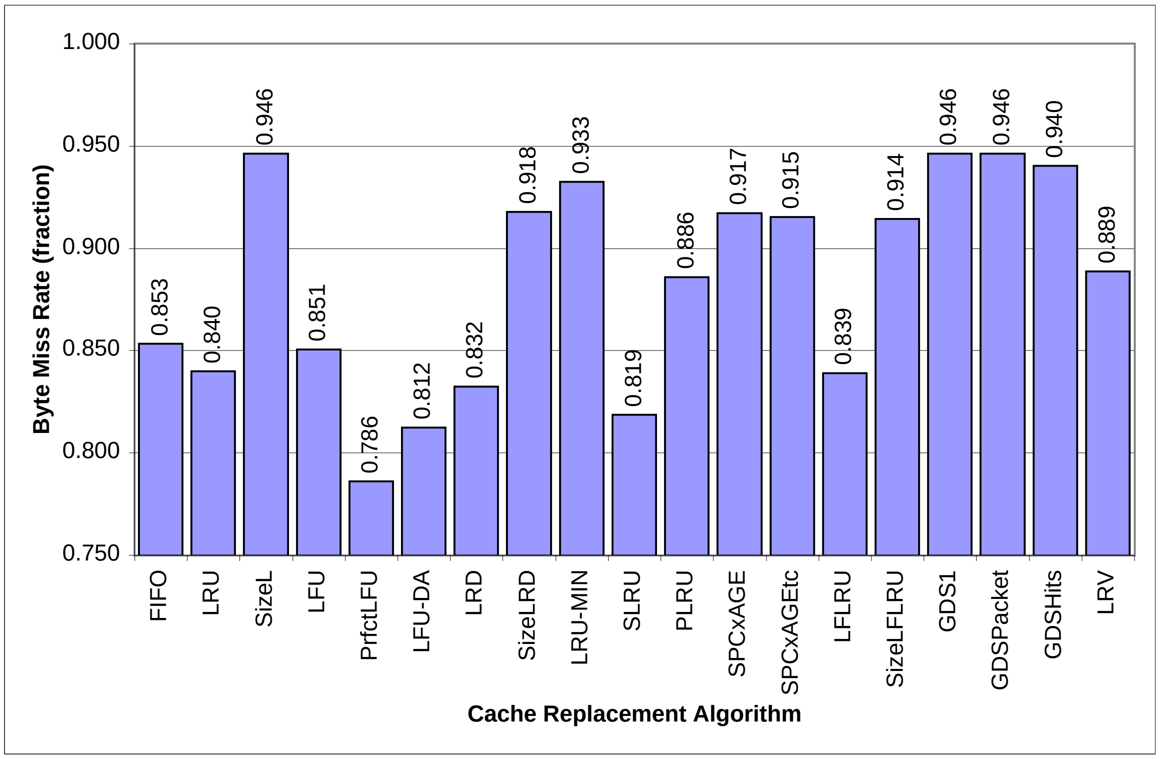


Fig. 6.34. Average Byte Miss Rates on Boulder1 Proxy Server bo1_day1 Trace.

NLANR San Jose Proxy Server Trace

For the San Jose Proxy Server trace among the simple cache replacement algorithms, the SizeL algorithm performs best as shown in Figure 6.35. The results from this algorithm are forwarded to Figure 6.36.

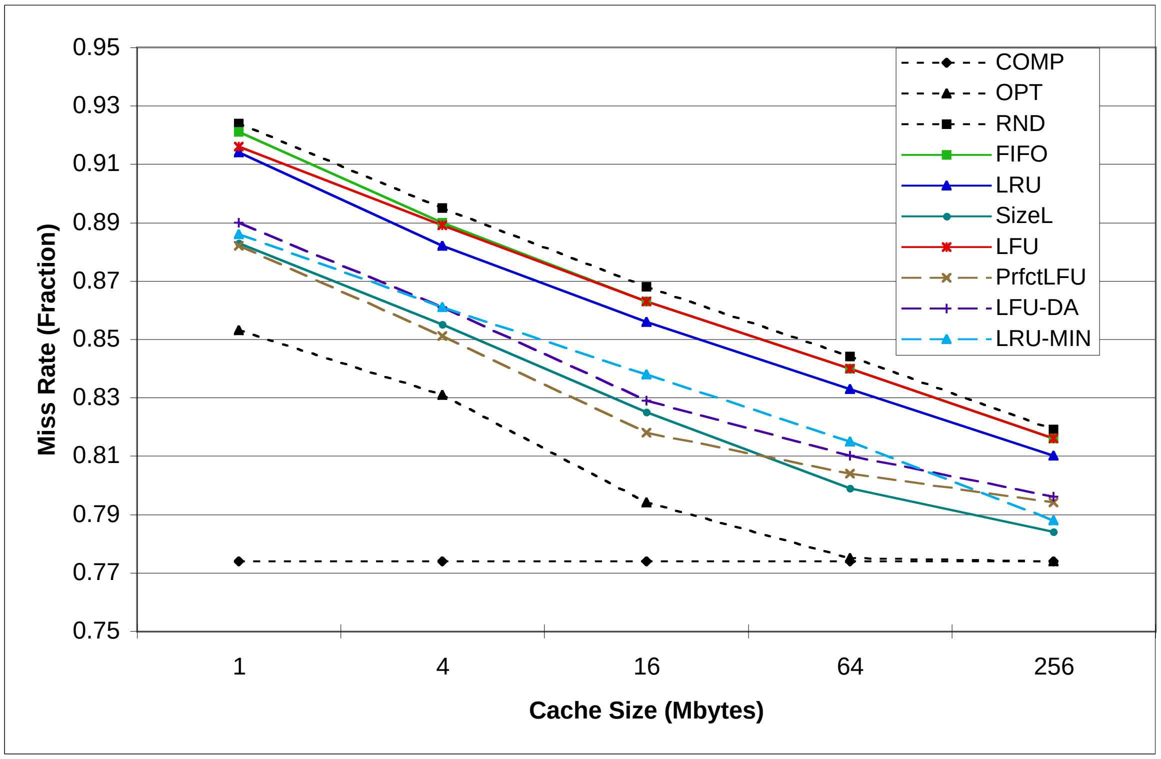


Fig. 6.35. Miss Rates of Simple Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

From Figure 6.36, the SPACExAGE and SPACExAGEtc algorithms perform best so their results are forwarded to Figure 6.37.

According to Figures 6.37 and 6.38, the top three performing algorithms for the San Jose trace with miss rate as the measure are GDSHits placing first, and SPACExAGE and SizeLFLRU tying for second.

When byte miss rate is the measure, Figure 6.39 shows that LFU-DA is the best performing algorithm among the simple cache replacement algorithms. The LFU-DA

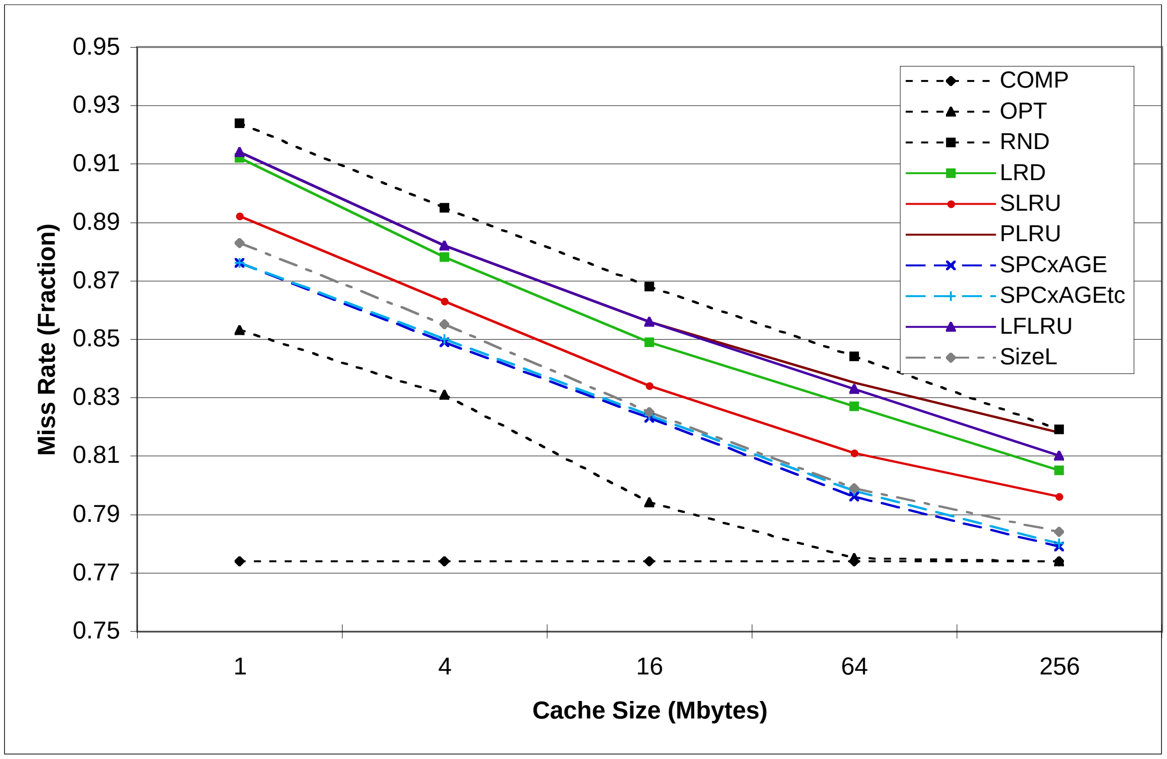


Fig. 6.36. Miss Rates of More Complex Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

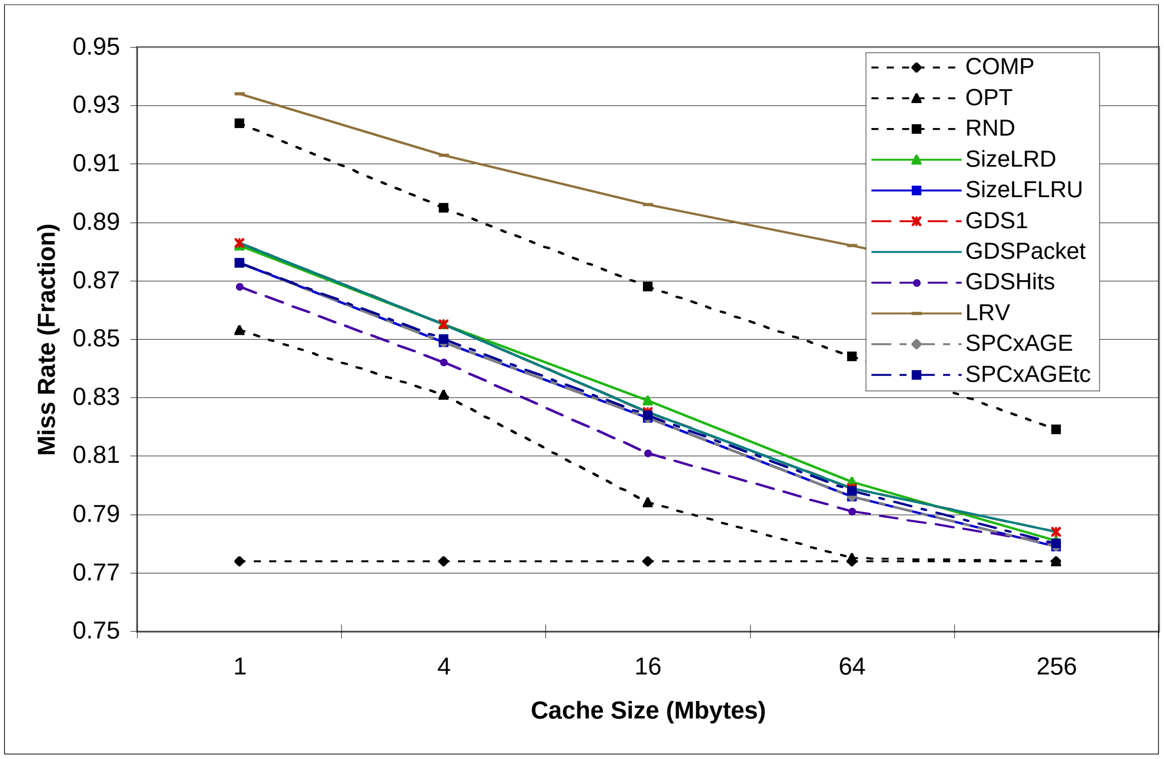


Fig. 6.37. Miss Rates of Complex Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

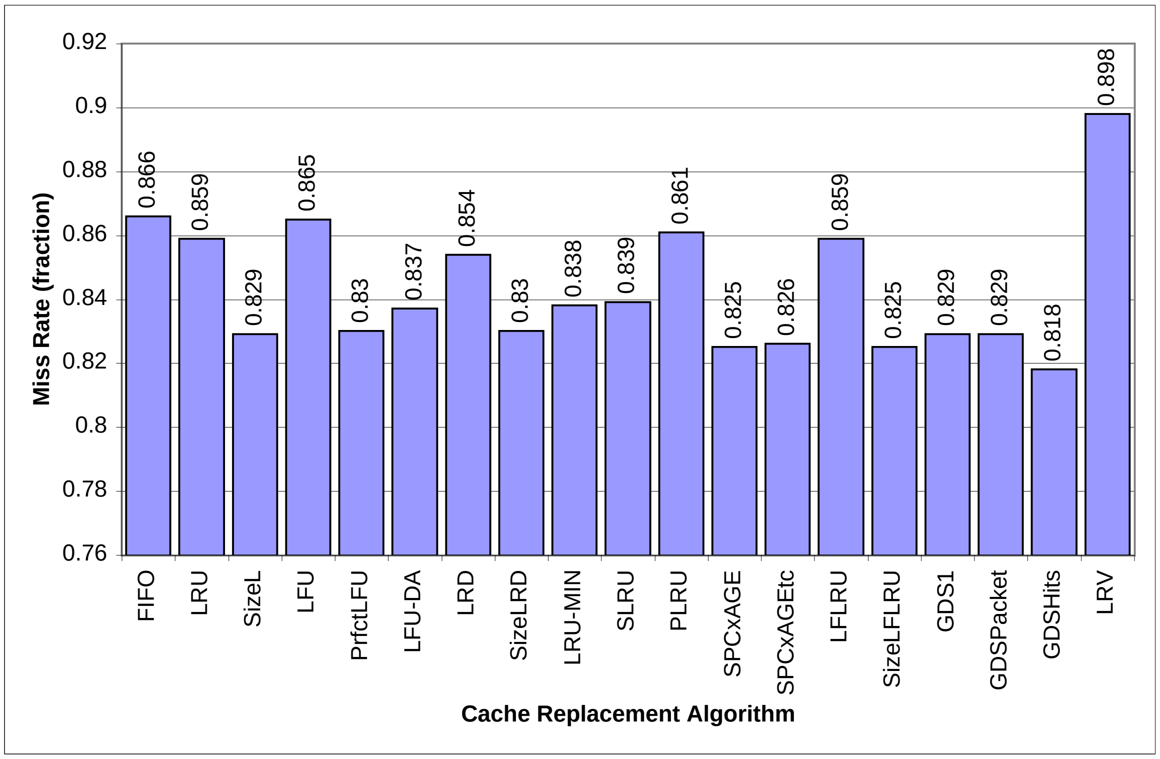


Fig. 6.38. Average Miss Rates on San Jose Proxy Server sj_novwk5 Trace.

results are then forwarded to Figure 6.40.

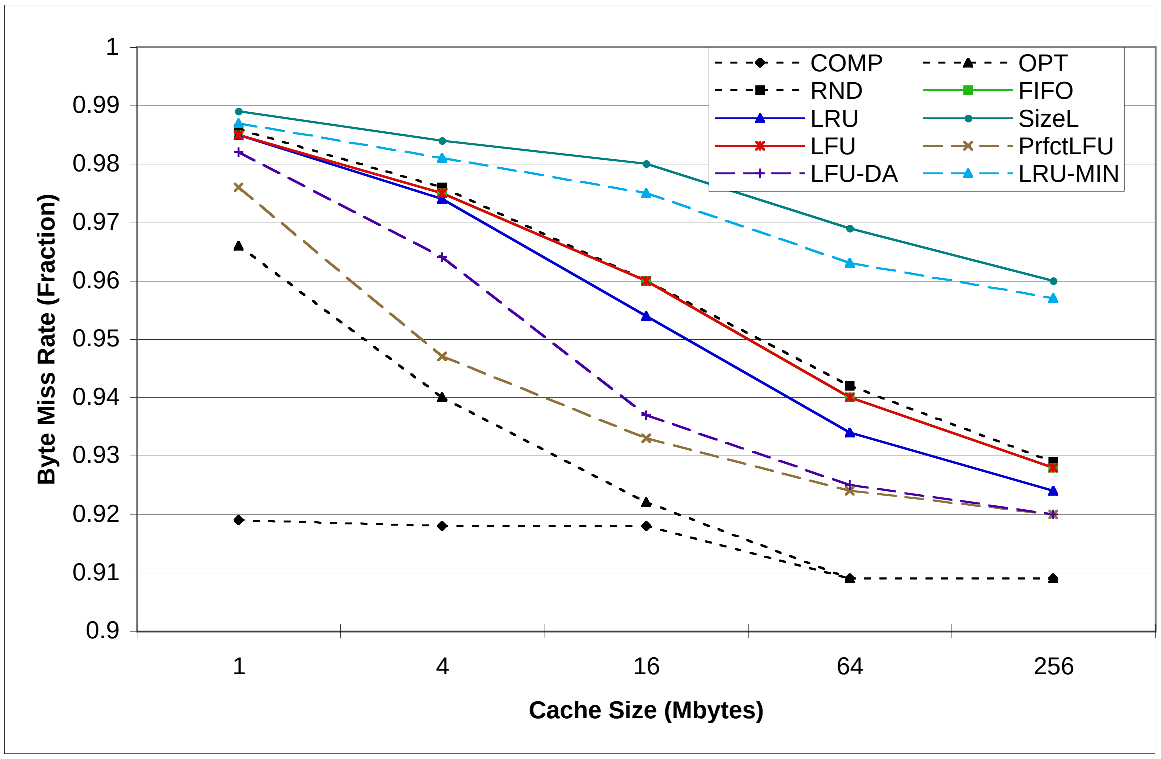


Fig. 6.39. Byte Miss Rates of Simple Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

In Figure 6.40, the top two algorithms are LFU-DA and SLRU whose results are subsequently forwarded to Figure 6.41.

Now from Figures 6.41 and 6.42, the top three cache replacement algorithms for the San Jose trace with byte miss rate as the measure are LFU-DA, SLRU, and LRD placing first, second, and third, respectively. In should be noted that LFLRU is only 0.1% worse in average byte miss rate than LRD.

NLANR Urbana-Champaign Proxy Server Trace

Among the simple cache replacement algorithms in Figure 6.43, the SizeL algorithm clearly performed the best, and its results are forwarded to Figure 6.44.

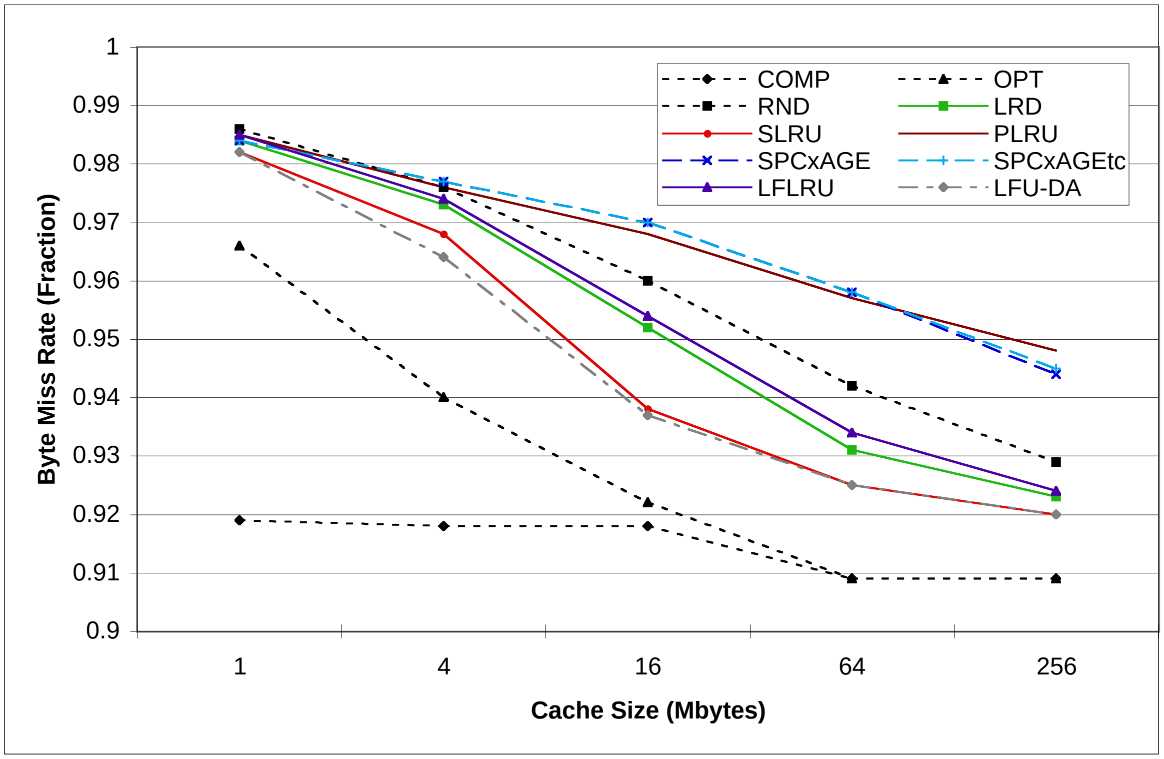


Fig. 6.40. Byte Miss Rates of More Complex Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

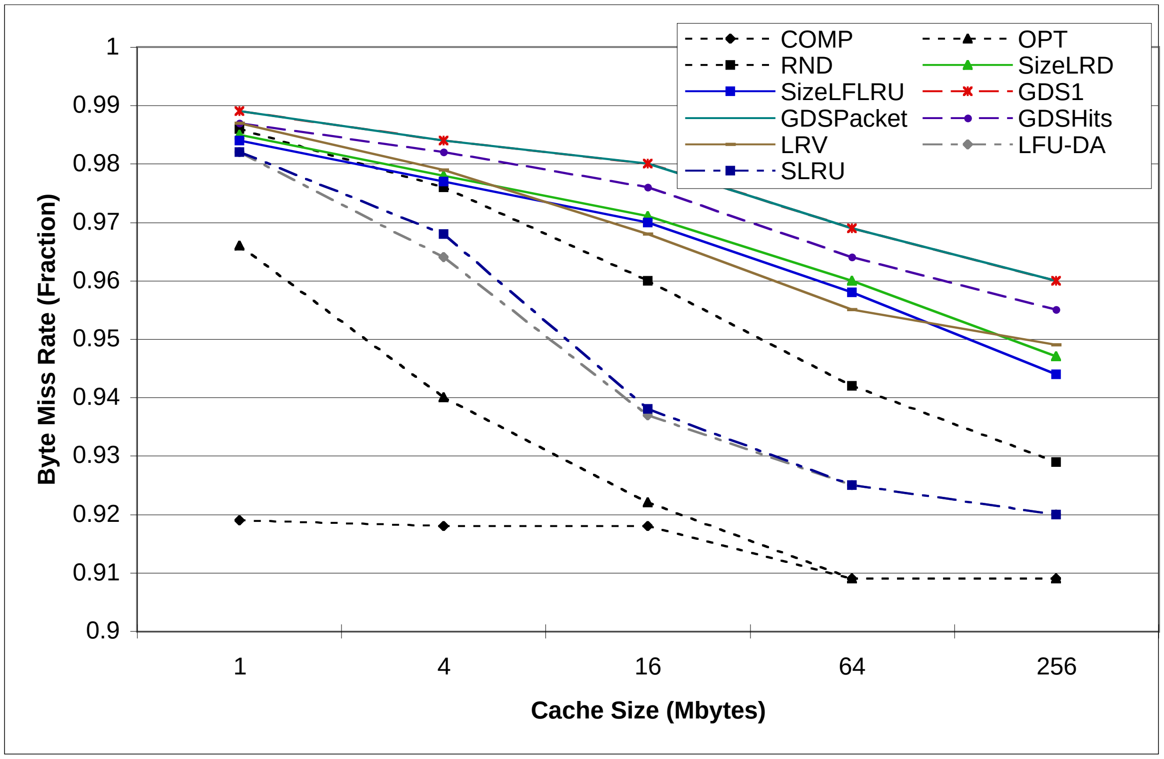


Fig. 6.41. Byte Miss Rates of Complex Cache Replacement Algorithms on San Jose Proxy Server sj_novwk5 Trace.

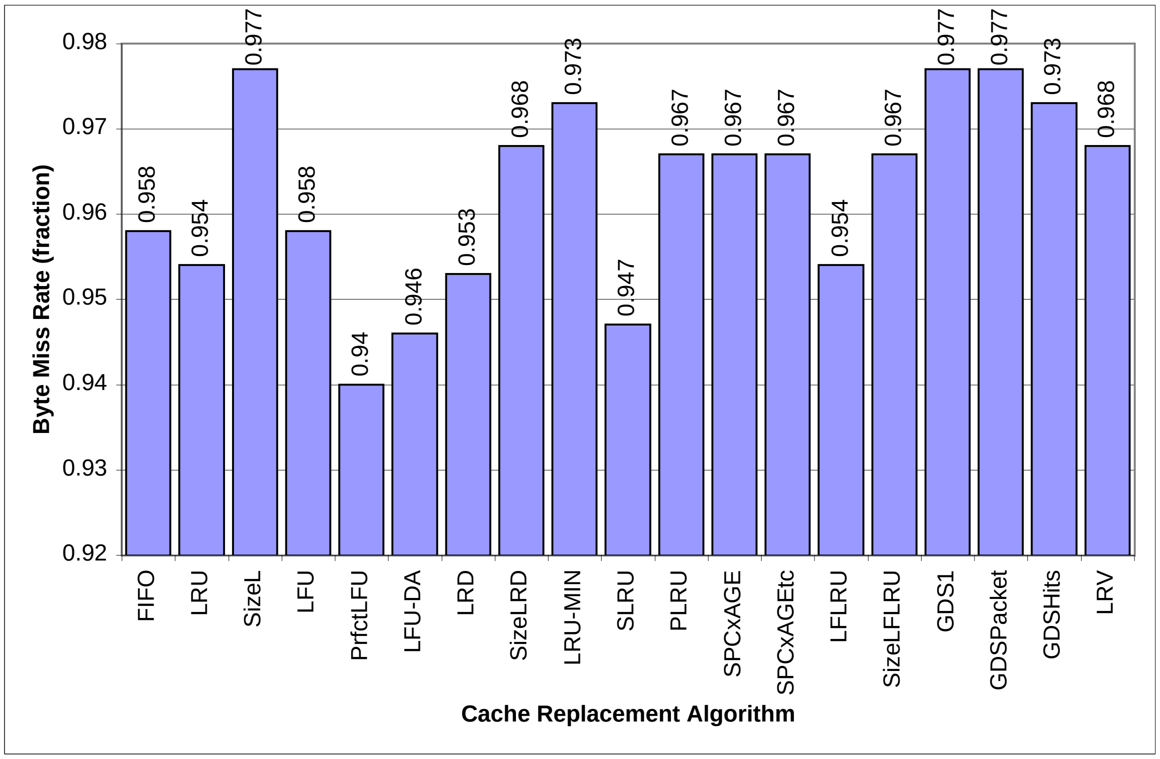


Fig. 6.42. Average Byte Miss Rates on San Jose Proxy Server sj_novwk5 Trace.

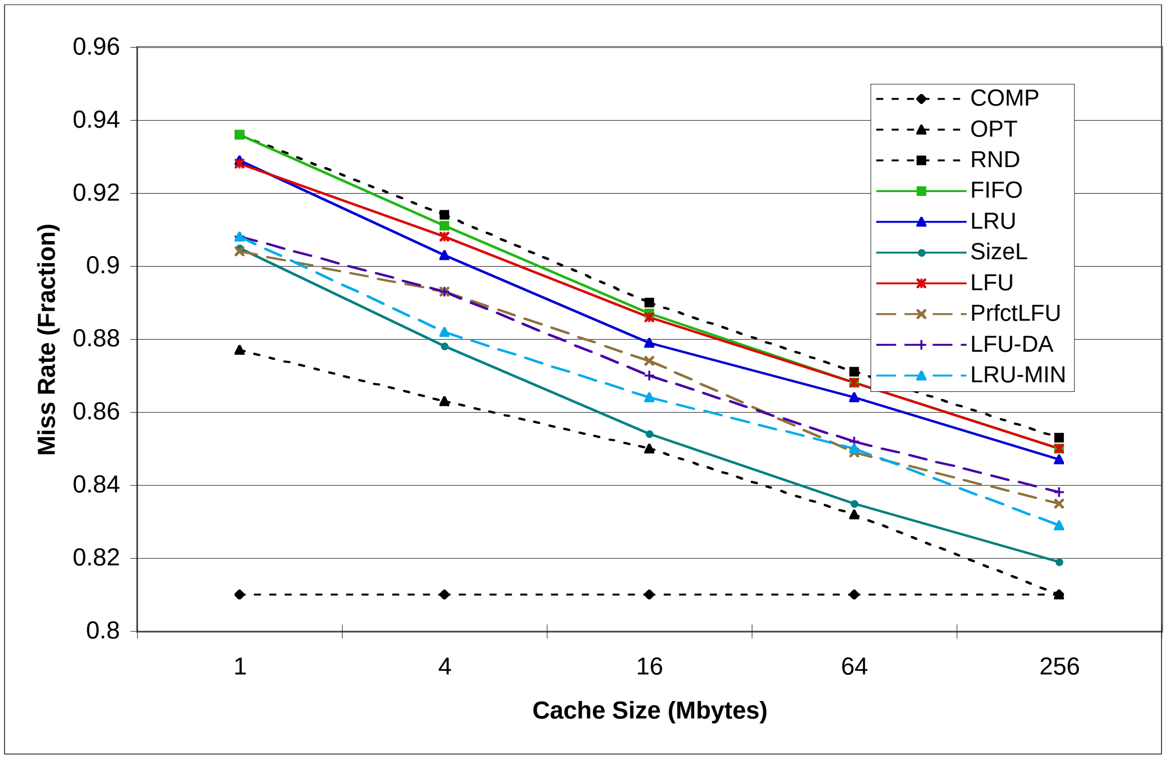


Fig. 6.43. Miss Rates of Simple Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

In Figure 6.44, the top performing algorithms are SPACExAGE and SPACExAGEtc. Their results are forwarded to Figure 6.45.

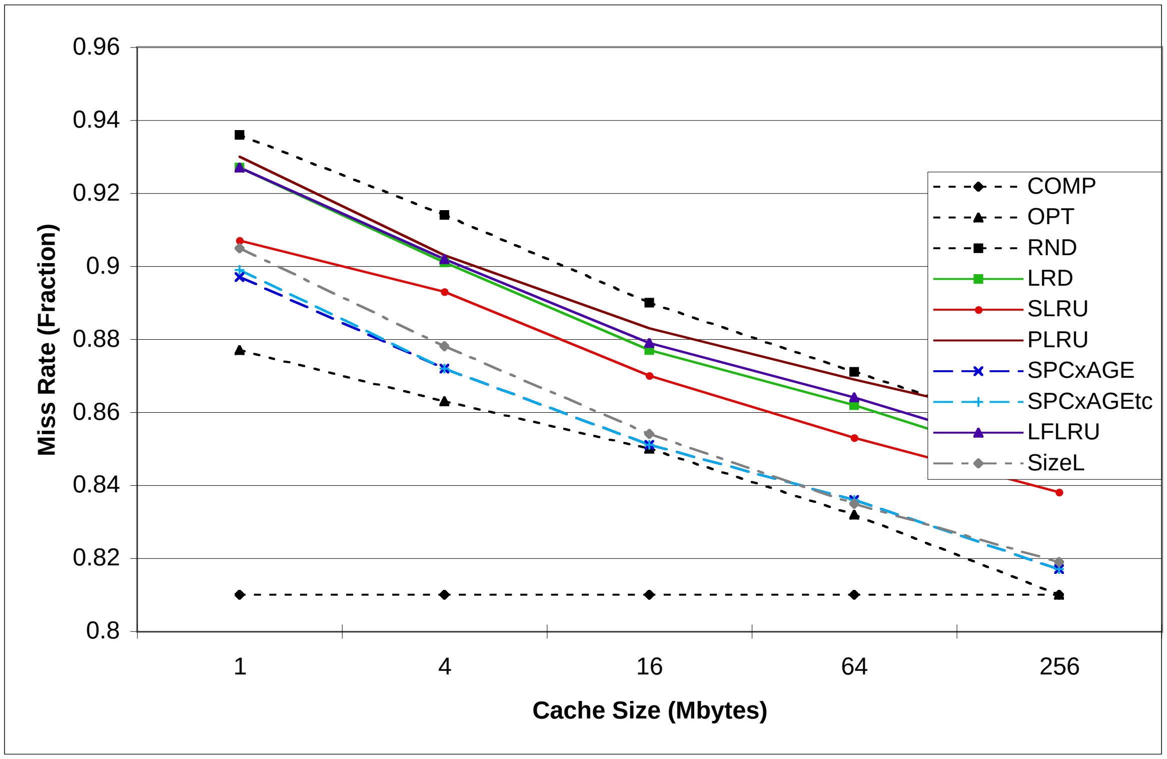


Fig. 6.44. Miss Rates of More Complex Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

In Figure 6.45, GDSHits performs best, and it performs especially well for the smaller cache sizes. It is more difficult to determine the other top performing algorithms from Figure 6.45, but from Figure 6.46, it can be seen that there is a tie for second place between SPACExAGE and SizeLFLRU.

For the byte miss rate measure in Figure 6.47, the LFU-DA algorithm performs best, and its results are forwarded to Figure 6.48.

In Figure 6.48, the LFU-DA and SLRU algorithms perform best, and their results are forwarded to Figure 6.49.

From Figures 6.49 and 6.50, one can see that the LFU-DA algorithm performs best, and it performs better than the second place algorithm, SLRU, for the smaller

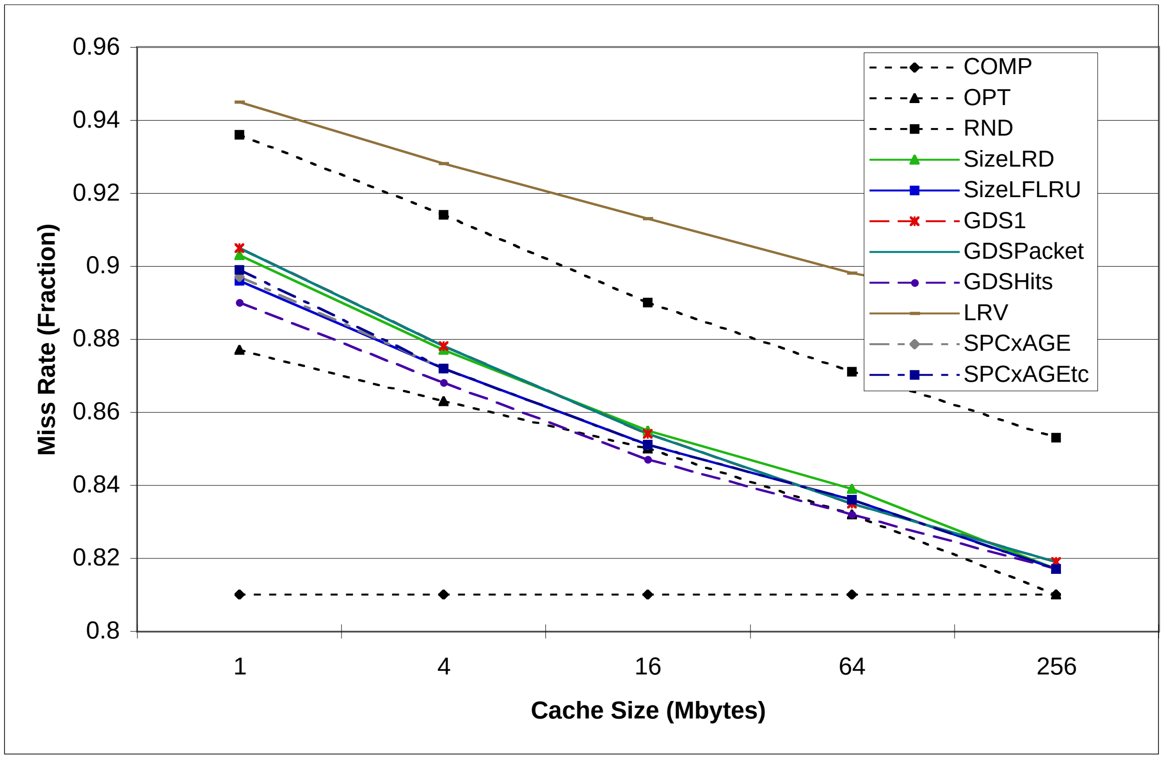


Fig. 6.45. Miss Rates of Complex Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

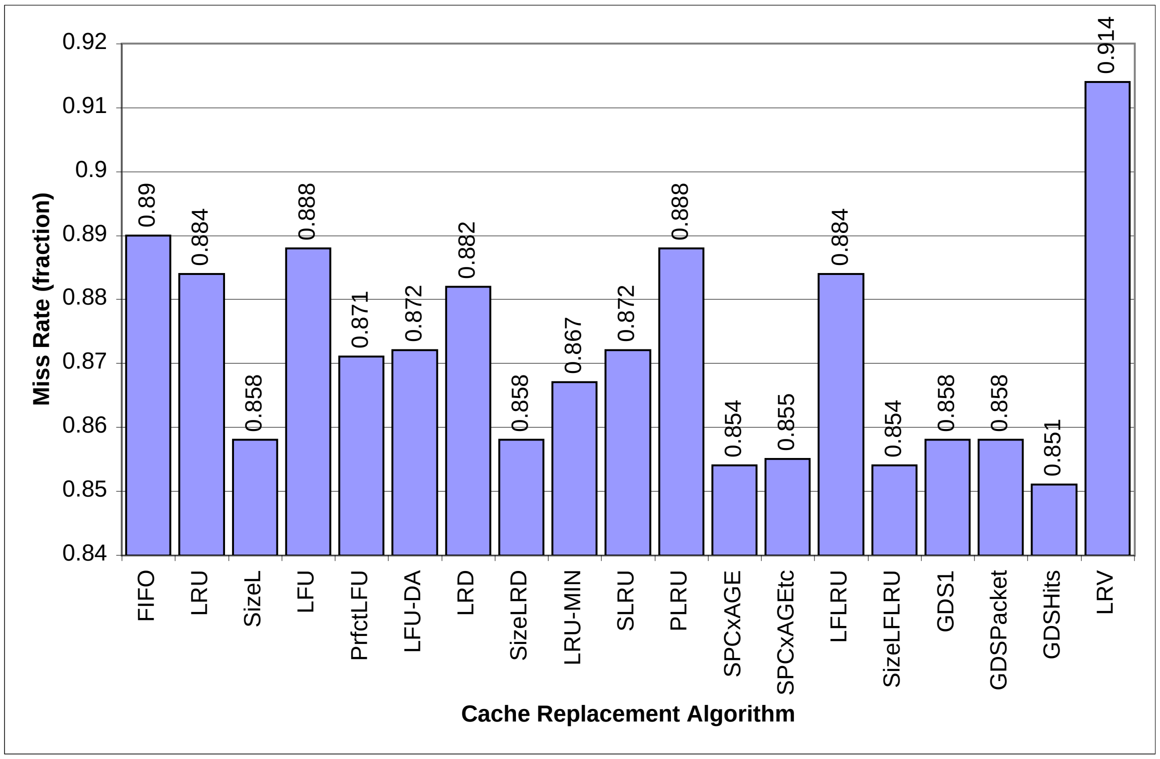


Fig. 6.46. Average Miss Rates on Urbana Champaign Proxy Server uc_day1 Trace.

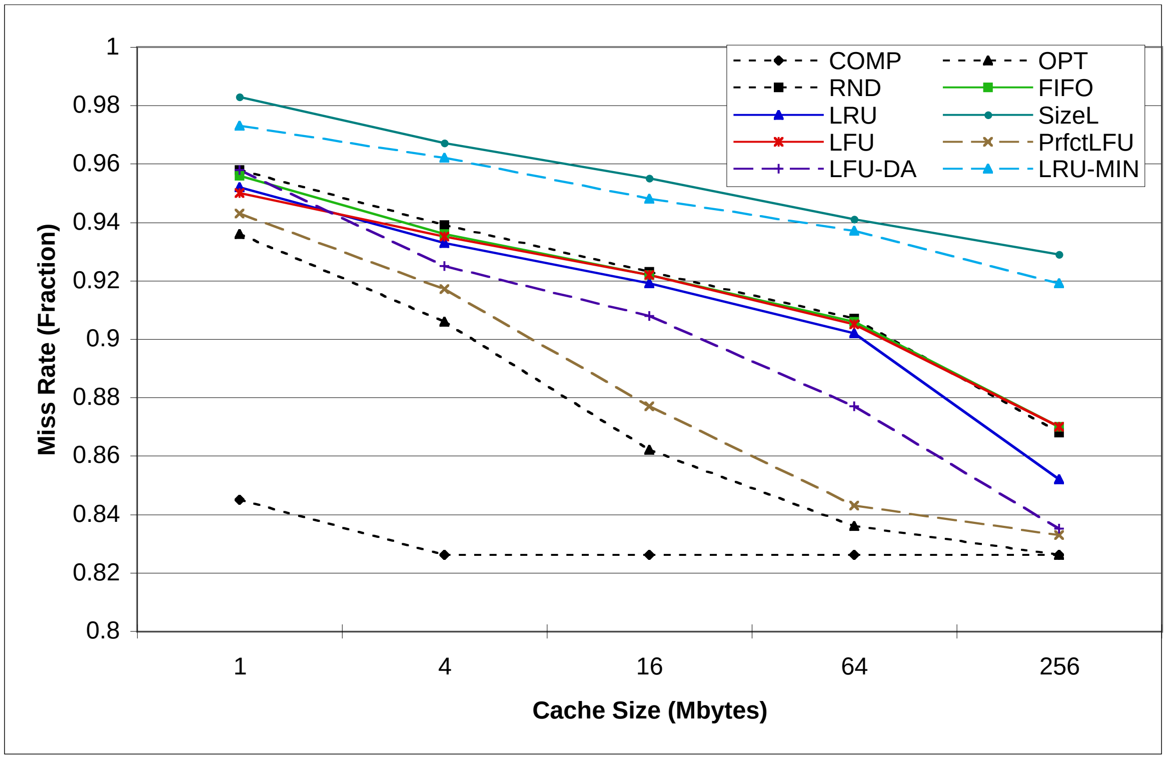


Fig. 6.47. Byte Miss Rates of Simple Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

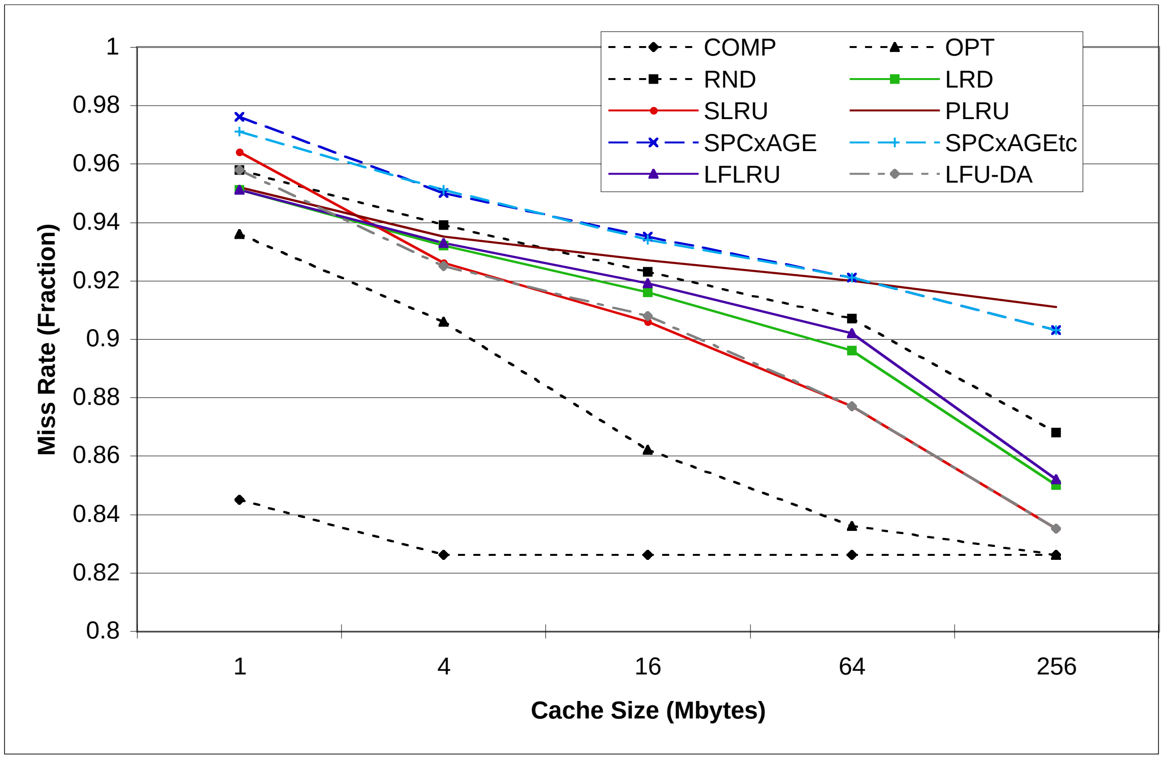


Fig. 6.48. Byte Miss Rates of More Complex Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

cache sizes. LRD places third, while LFLRU is close behind LRD.

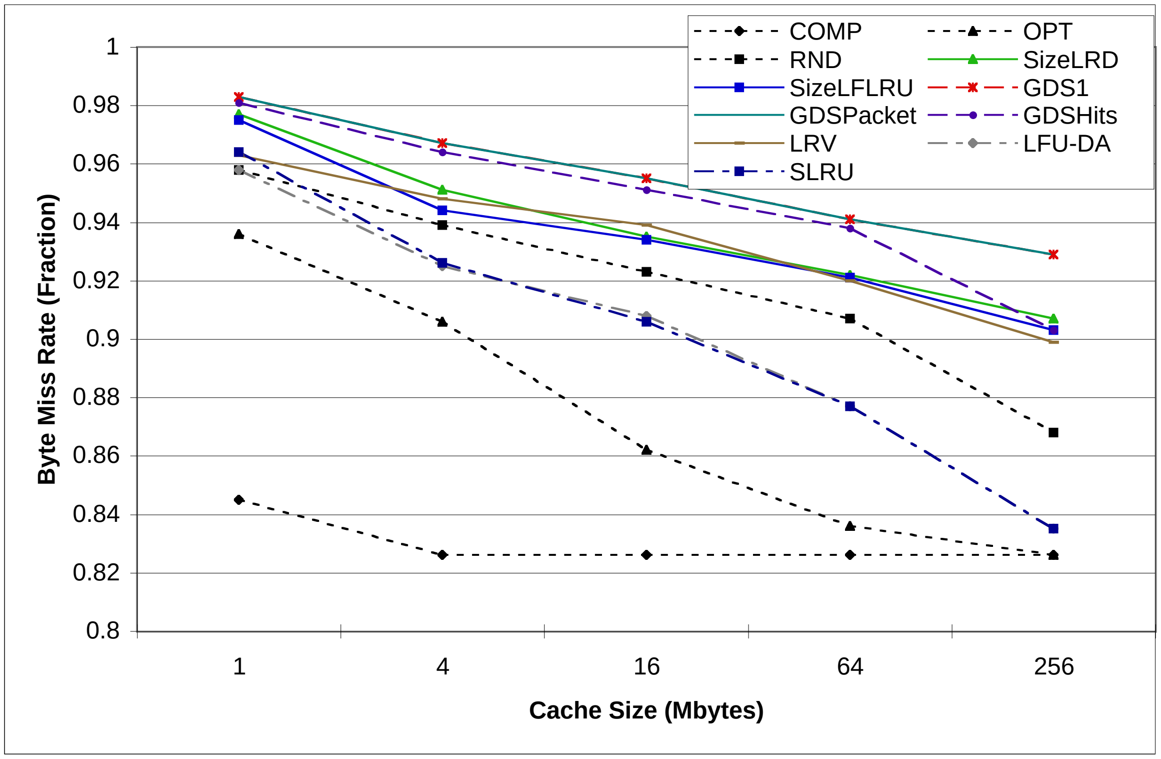


Fig. 6.49. Byte Miss Rates of Complex Cache Replacement Algorithms on Urbana Champaign Proxy Server uc_day1 Trace.

Network Proxy Server Traces Summary

For the network domain's summary, the table of top performing algorithms for each trace and both measures is presented in Table 6.3. On this table, one can see that the best algorithms for this domain are consistently GDSHits when optimizing for miss rate, and LRU-DA when optimizing for byte miss rate. Among the best algorithms for miss rate, SPACExAGE and SizeLFLRU are consistently the next best algorithms, while SLRU, LFLRU, and LRD show consistency among the top contenders for byte miss rate.

For the sake of the comparisons, the average miss rate graphs of Figures 6.30,

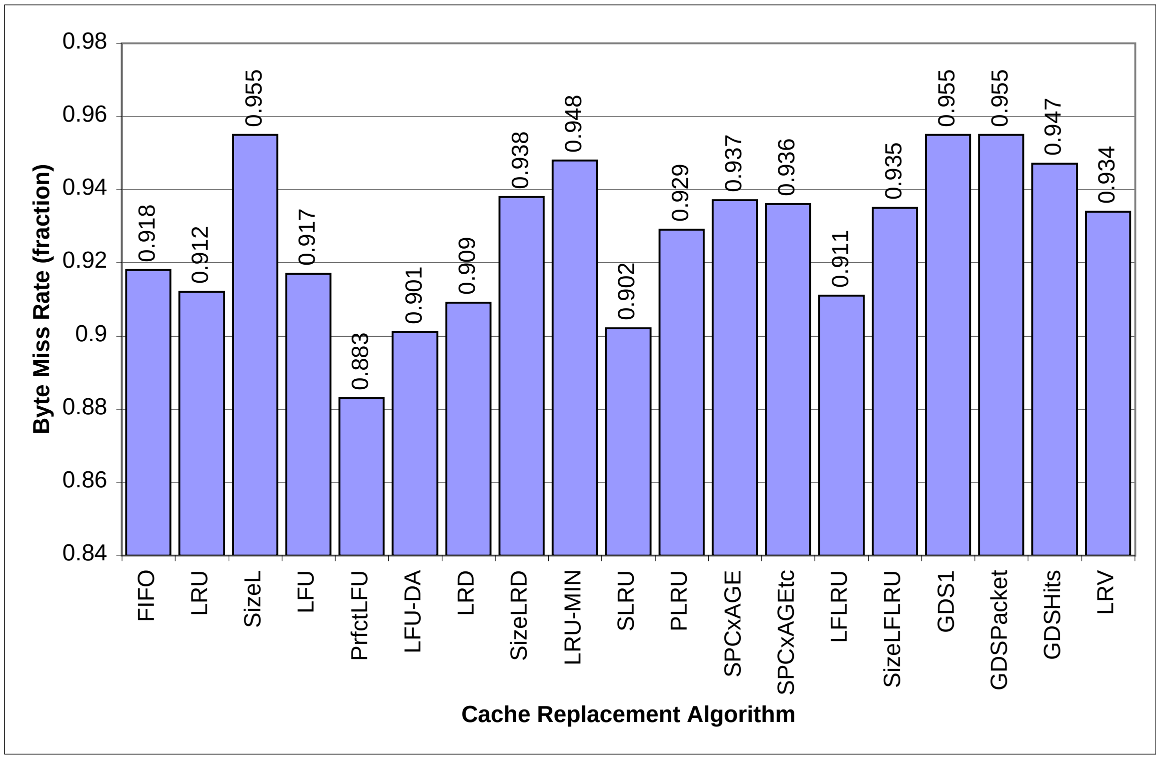


Fig. 6.50. Average Byte Miss Rates on Urbana Champaign Proxy Server uc_day1 Trace.

Table 6.3
Summary of Top Cache Replacement Algorithms for Network Proxy Server Traces

Trace	Boulder1	San Jose	Urbana Champaign
Miss Rate	GDSHits (1st) SPACExAGE (2nd-tie) SPACExAGEtc (2nd-tie) SizeLFLRU (2nd-tie)	GDSHits (1st) SPACExAGE (2nd-tie) SizeLFLRU (2nd-tie)	GDSHits (1st) SPACExAGE (2nd-tie) SizeLFLRU (2nd-tie)
Byte Miss Rate	LFU-DA (1st) SLRU (2nd) LFLRU (3rd)	LFU-DA (1st) LRD (2nd) LFLRU (3rd)	LFU-DA (1st) SLRU (2nd) LRD (3rd)

6.38, and 6.46 are shrunk and brought together in Figure 6.51. When looking at these average miss rate graphs, the results of these network proxy cache workload traces are very similar to each other; they only differ from each other by a relative shift of miss rate of up to 5%. The cache replacement algorithms that rely heavily on the access count of the objects like LFU, LFU-DA, and PerfectLFU do not perform well. Also, LFLRU and LRD, which combine access count with LRU and FIFO, respectively, do not improve much on the performance of LRU and FIFO. However, when object size is included in the cache score calculation, performance is improved significantly. SizeL also performs very well as do all three of the Greedy Dual Size algorithms which use size as one of their deciding criteria. But when combining object size, LRU, and access count, the best performance is gleaned as shown in Table 6.3. GDSHits is consistently the best and SizeLFLRU is consistently tied for second. One can argue that the access count portion of SizeLFLRU does not significantly factor into the cache replacement score because it tied with SPACExAGE, but GDSHits, which uses mostly size and access count along with an aging inflation factor, is the best performing algorithm. When taking object file types into account as in PLRU, the miss rate is increased. This finding should be expected when comparing the file types of first accesses and subsequent access; there is very little difference in these percentage breakdowns in Tables 5.67 and 5.68, Tables 5.76 and 5.77, and Tables 5.85 and 5.86.

For the average byte miss rates, the average byte miss rate graphs of Figures 6.34, 6.42, and 6.50 are shrunk and brought together in Figure 6.52. As with the miss rate graphs, the results of these network proxy cache workload traces for the byte miss rate measure are very similar to each other; they also only differ from each other by a relative shift which can be up to 15%. The size-based cache replacement algorithms usually performed poorly. For instance, SizeLRD and SizeLFLRU perform worse than FIFO and LRU respectively, while LRD and LFLRU do perform slightly better than FIFO and LRU. Also SizeL, LRU-MIN, SPACExAGE, and SPACExAGEtc perform rather poorly as do the GreedyDual-Size algorithms which all rely on object sizes for eviction determination. On the other hand, using access count helps in improving the performance of the algorithms over LRU. LFU-DA (which was the top performing algorithm for byte miss rate), SLRU, LRD, and LFLRU all performed as well or better than LRU on a consistent basis. For byte miss rate, using file type as with PLRU actually hindered the performance of the LRU algorithm.

6.1.3 Client Proxy Cache Server Traces

Among these client proxy cache server traces, the cache replacement algorithm performance results will be examined from the largest proxy cache server to the smallest. This ranking is in terms of a rough estimate of number of clients that the proxy cache server was serving.

Boeing Proxy Cache Server Trace

Among the simple cache replacement algorithms working on the Boeing proxy cache server trace, Figure 6.53 shows that the SizeL algorithm performs best, and its results are forwarded to Figure 6.54.

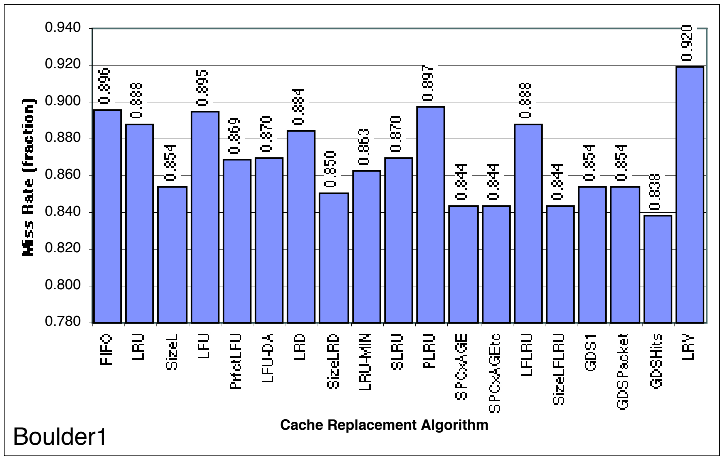
The SizeL and SPACExAGE results are forwarded from Figure 6.54 to Figure 6.55 for the best two performing algorithms among the more complex algorithms.

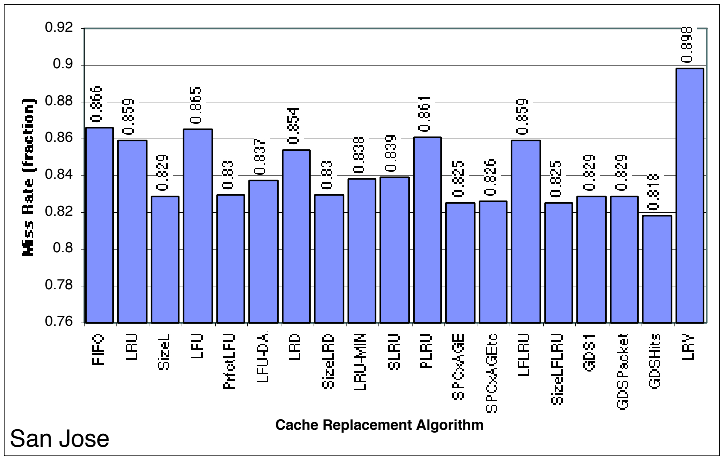
In Figures 6.55 and 6.56, the three GreedyDual-Size algorithms perform the best on the Boeing workload trace with miss rate as the measure. GDSHits performed best with GDS1 and GDSPacket tied for second.

Looking at the simple cache replacement algorithms using byte miss rate as the measure in Figure 6.57, LFU-DA performs the best so its results are forwarded to Figure 6.58.

On Figure 6.58, LFU-DA and SLRU perform best, and their results are forwarded to Figure 6.59.

From Figures 6.59 and 6.60, the best performing cache replacement algorithms for the Boeing workload trace when optimizing for byte hit rate are SLRU placing first, LFU-DA placing second, and LRD placing third. Having just 0.2% higher average byte miss rate than LRD is LFLRU.





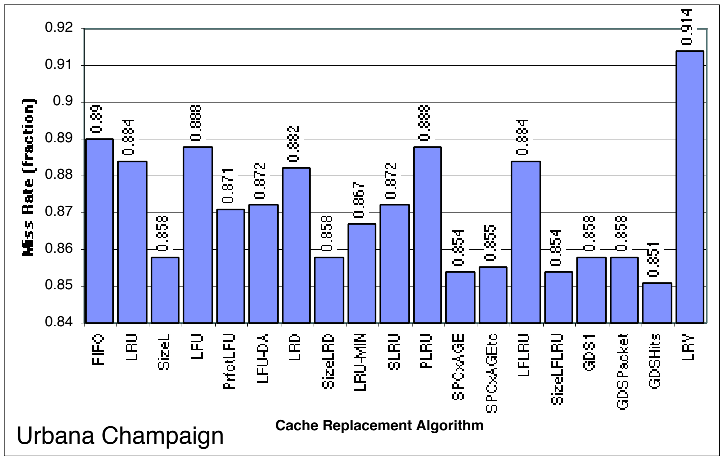


Fig. 6.51. Composite of Average Miss Rates for Network Proxy Server Traces.

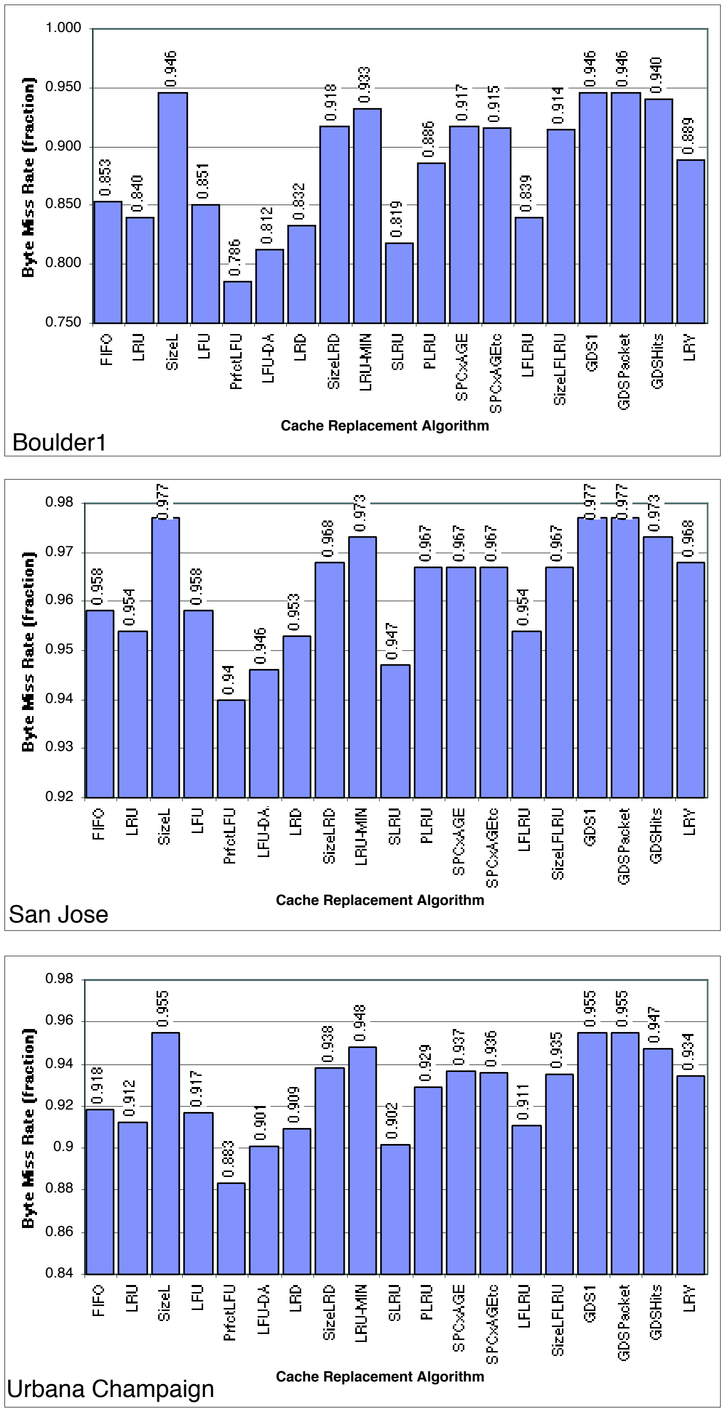


Fig. 6.52. Composite of Average Byte Miss Rates for Network Proxy Server Traces.

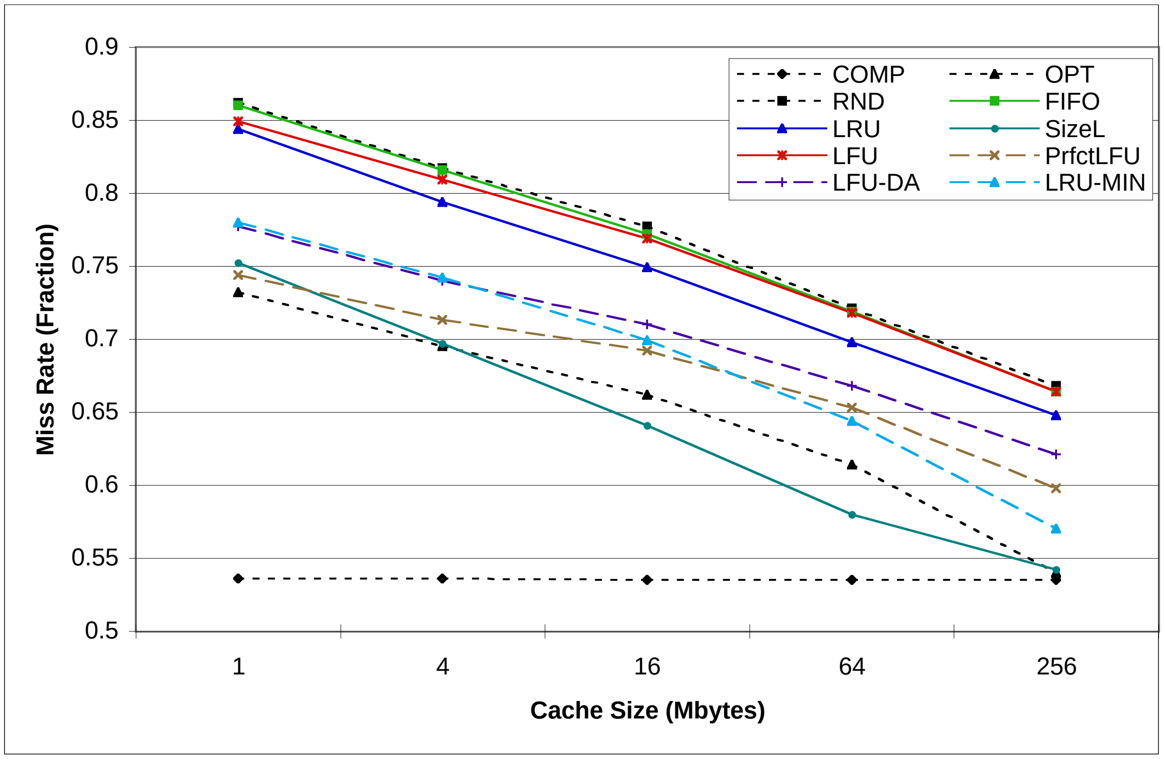


Fig. 6.53. Miss Rates of Simple Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

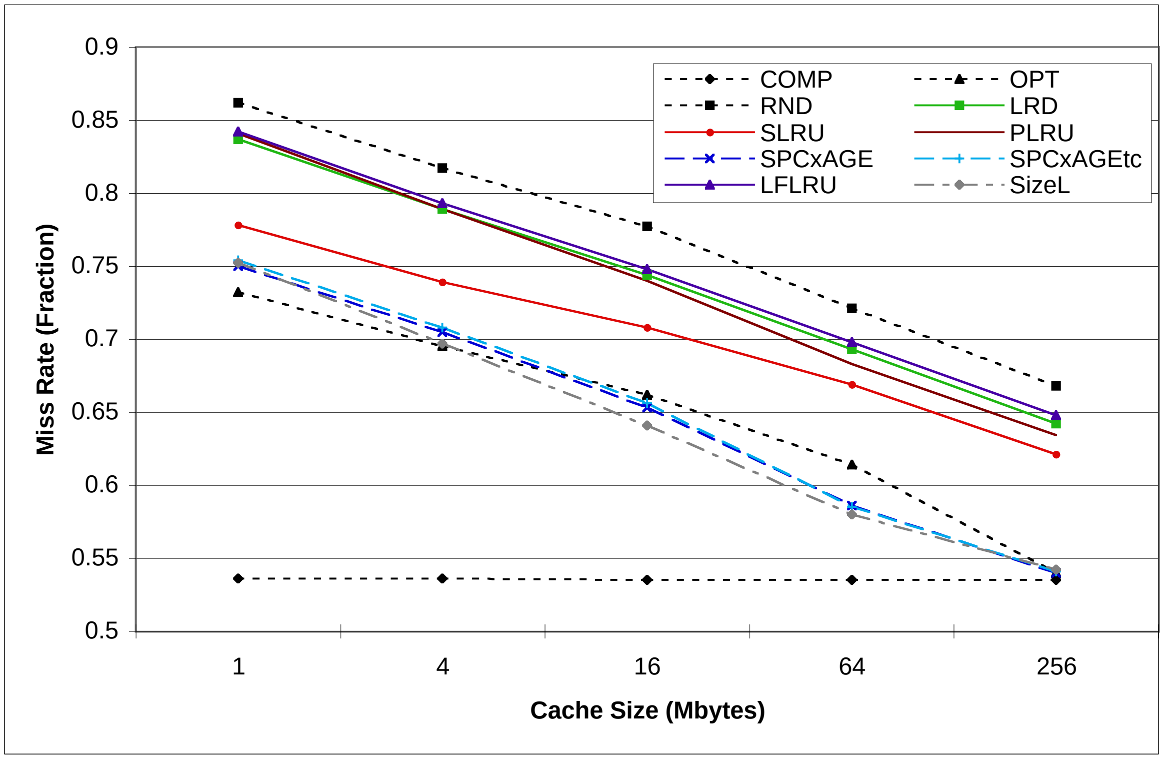


Fig. 6.54. Miss Rates of More Complex Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

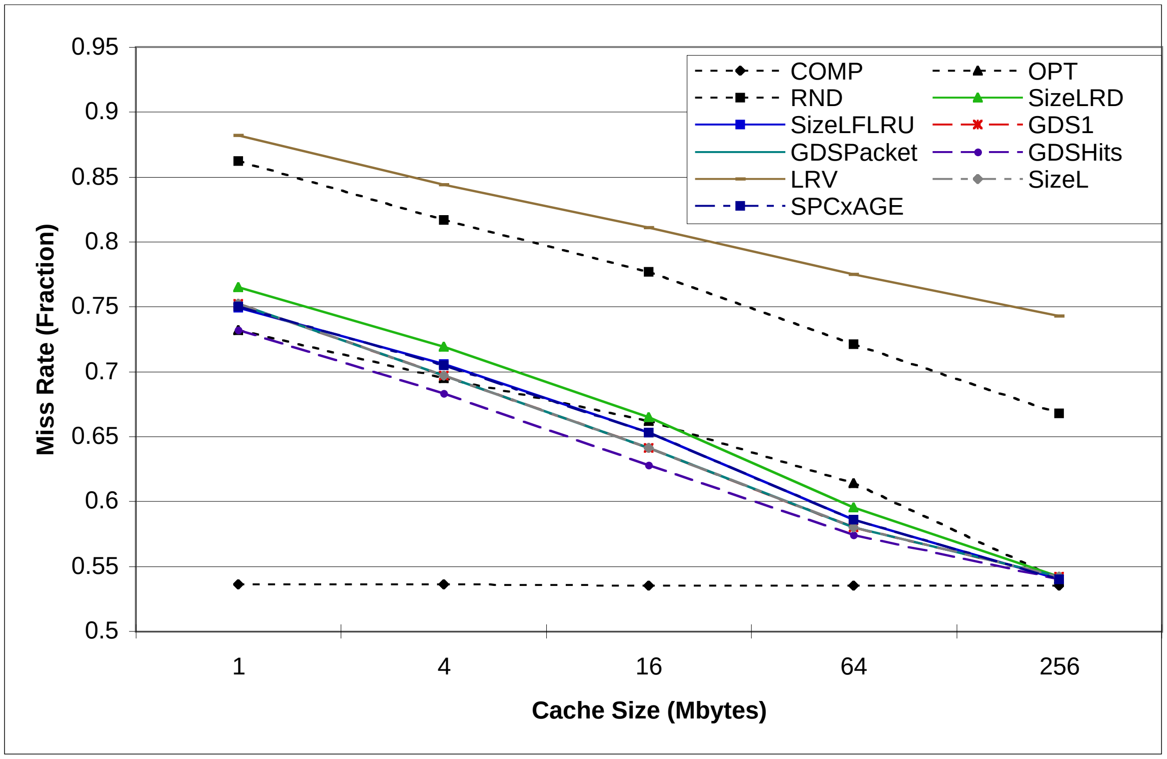


Fig. 6.55. Miss Rates of Complex Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

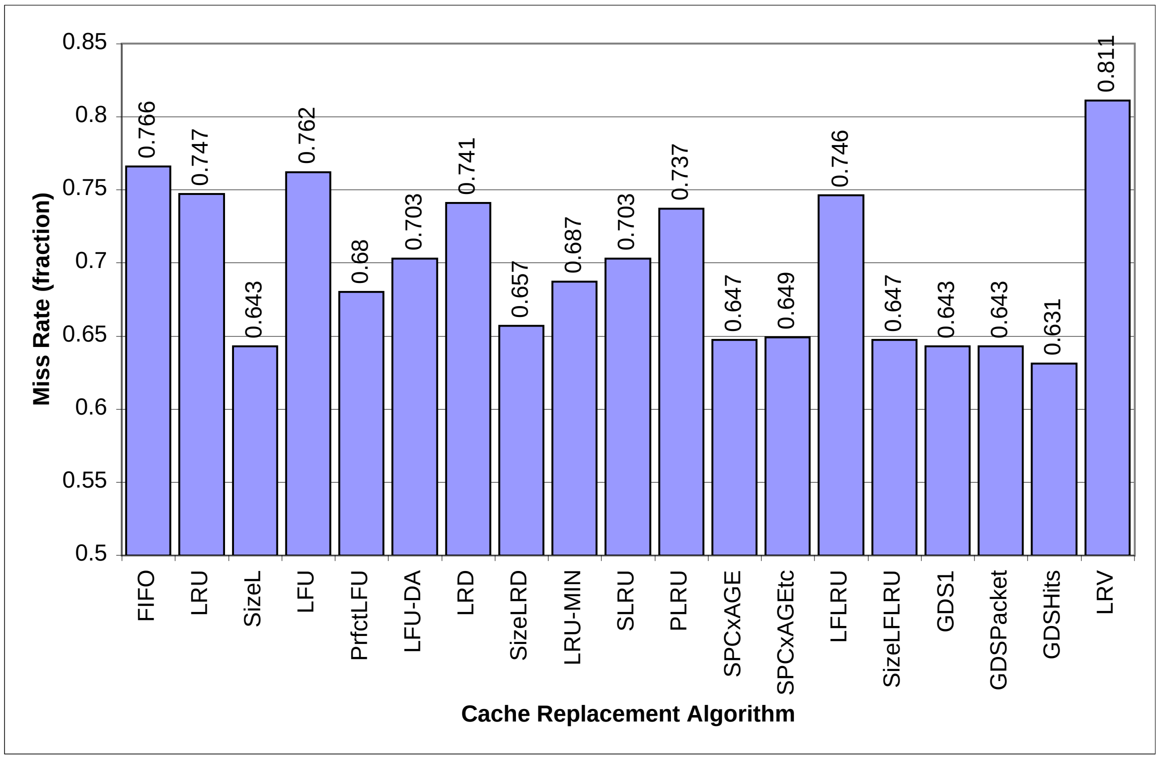


Fig. 6.56. Average Miss Rates on Boeing Proxy Cache Server boeing.990301 Trace.

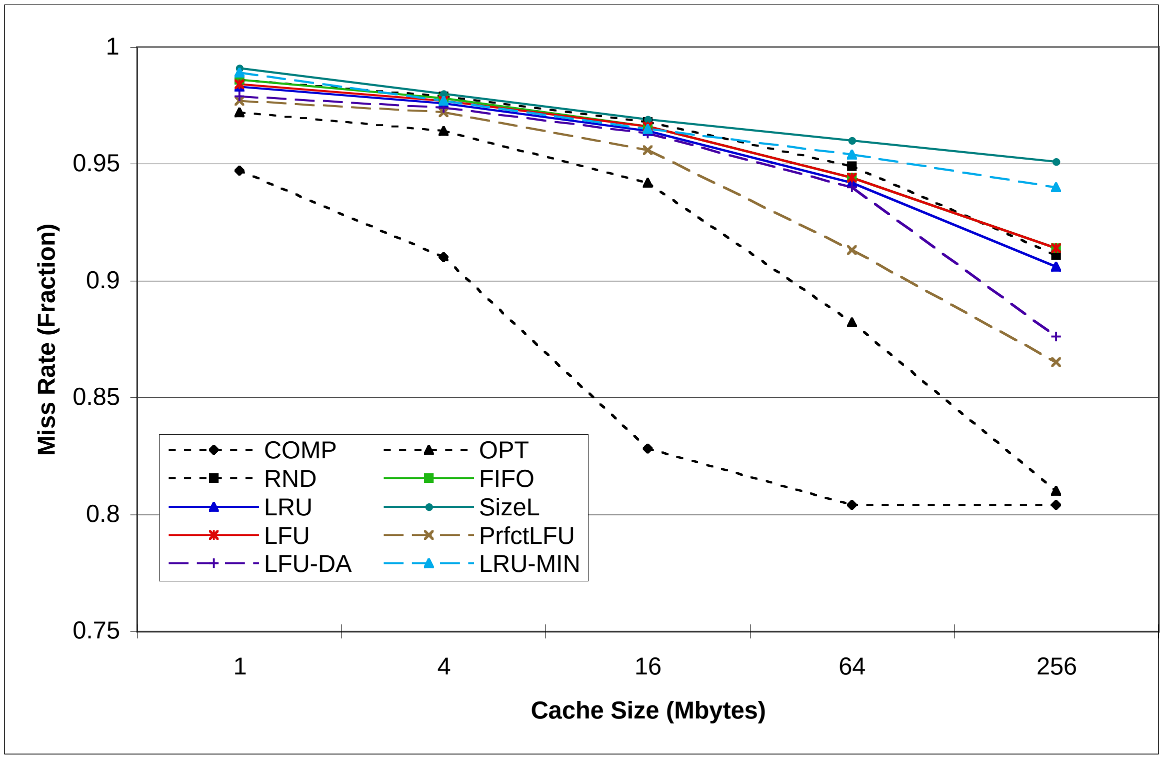


Fig. 6.57. Byte Miss Rates of Simple Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

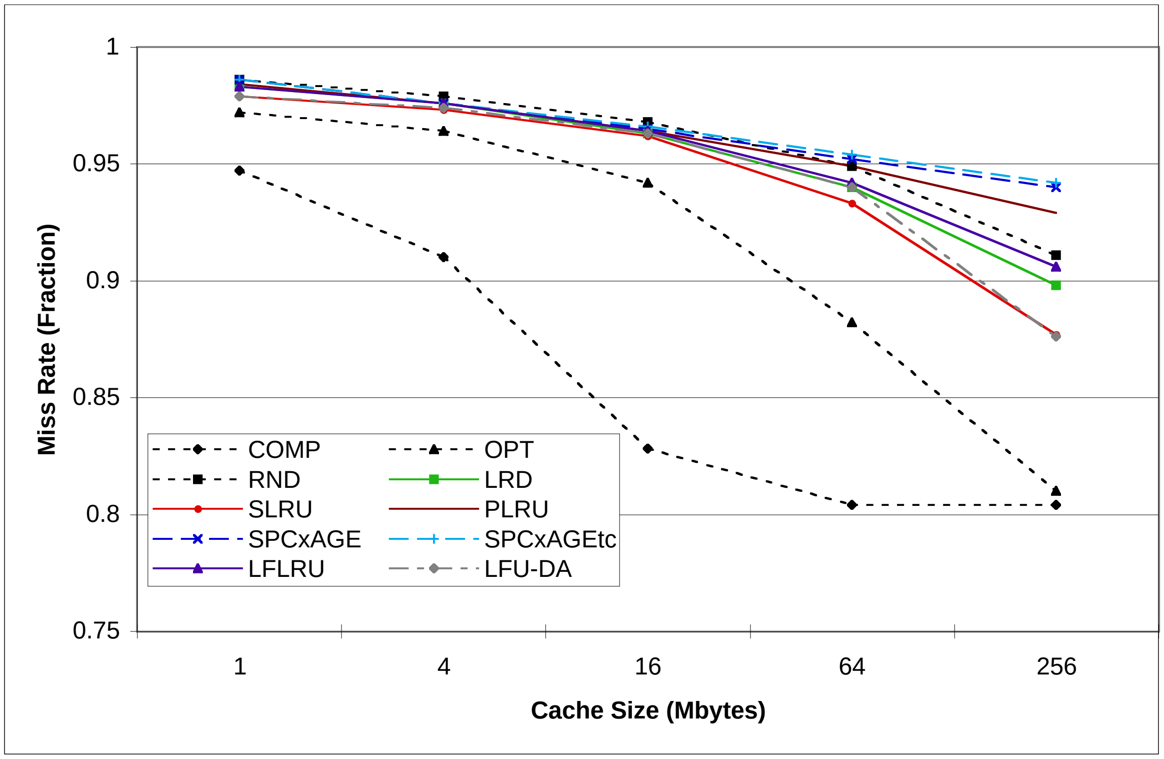


Fig. 6.58. Byte Miss Rates of More Complex Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

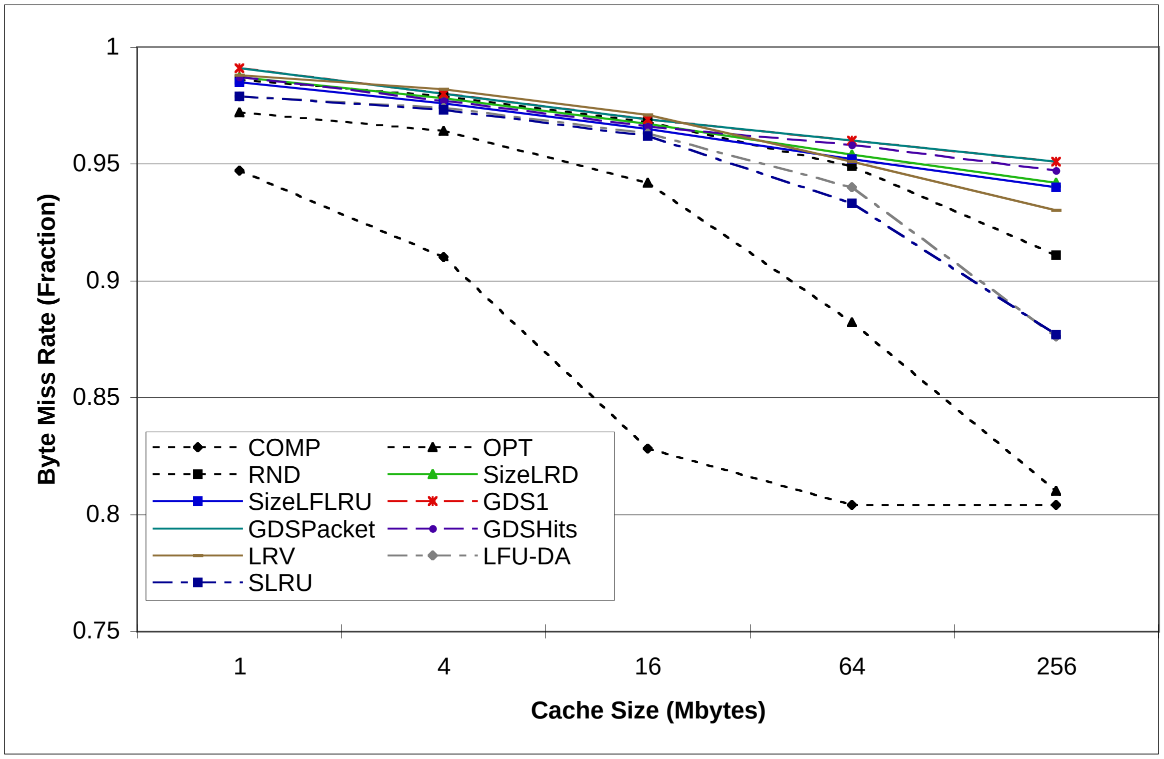


Fig. 6.59. Byte Miss Rates of Complex Cache Replacement Algorithms on Boeing Proxy Cache Server boeing.990301 Trace.

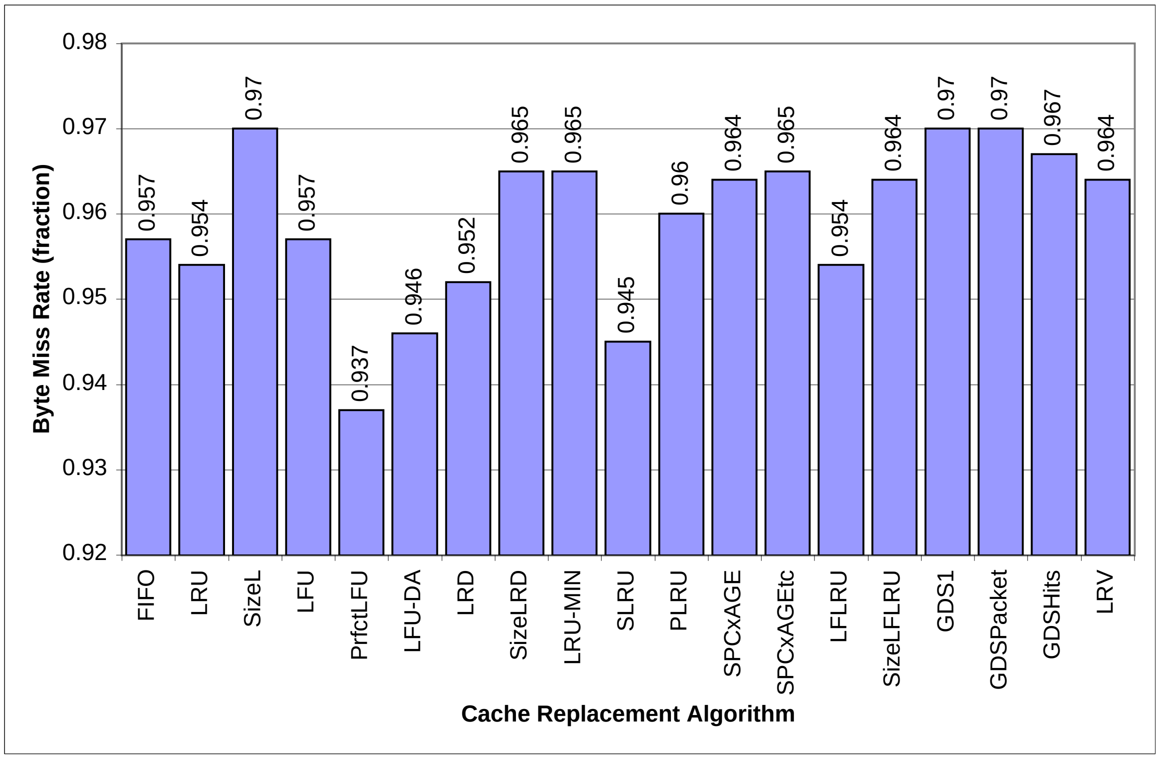


Fig. 6.60. Average Byte Miss Rates on Boeing Proxy Cache Server boeing.990301 Trace.

Purdue Stack (ECN) Proxy Server Trace

In Figure 6.61, the best performing simple cache replacement algorithm for the Stack Proxy Server workload trace is LRU-MIN for the miss rate measure. The LRU-MIN results are then forwarded to Figure 6.62.

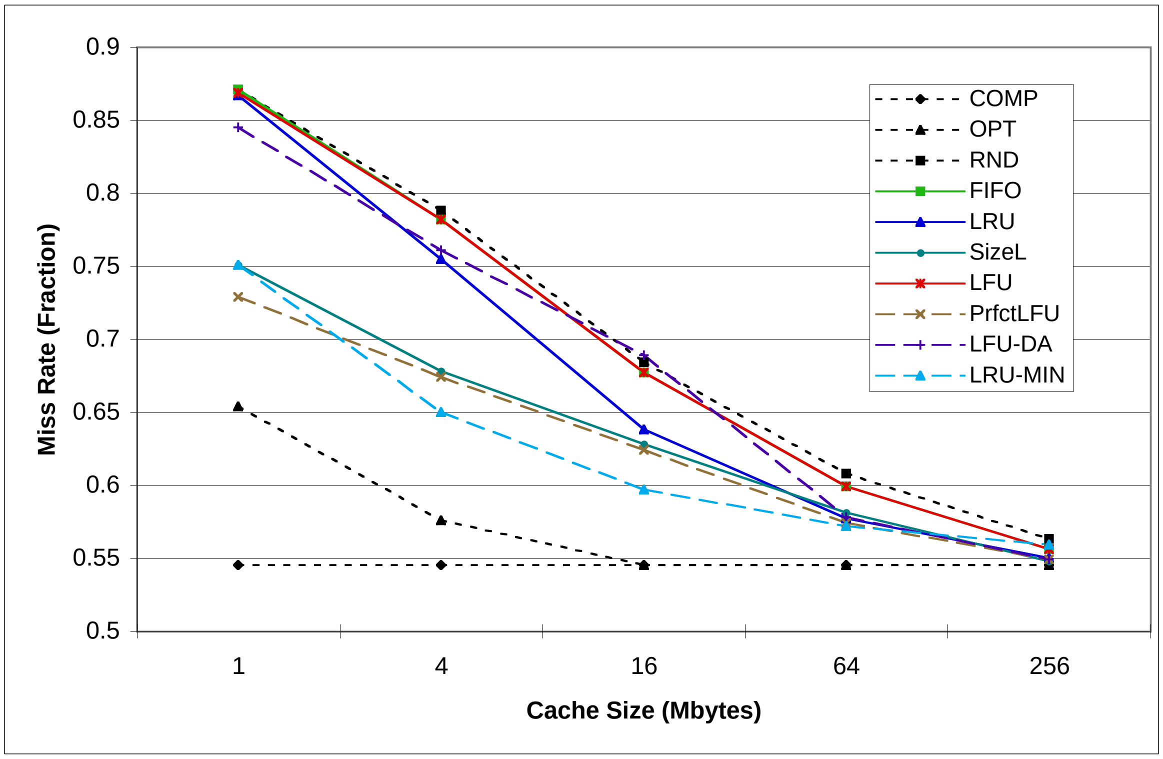


Fig. 6.61. Miss Rates of Simple Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

On the more complex miss rate graph of Figure 6.62, the SPACExAGE algorithm performs best. LRU-DA performs second best for small cache sizes while SPACEx-AGEtc performs second best for larger cache sizes, and they are tied on the average miss rate graph in Figure 6.64. Since better performance for smaller caches is more advantageous, the SPACExAGE and LRU-DA results are forwarded to Figure 6.63.

In Figure 6.63, GDSHits performs best for very the smallest cache size but does not maintain this superiority for larger cache sizes. GDSHits on the average miss rate

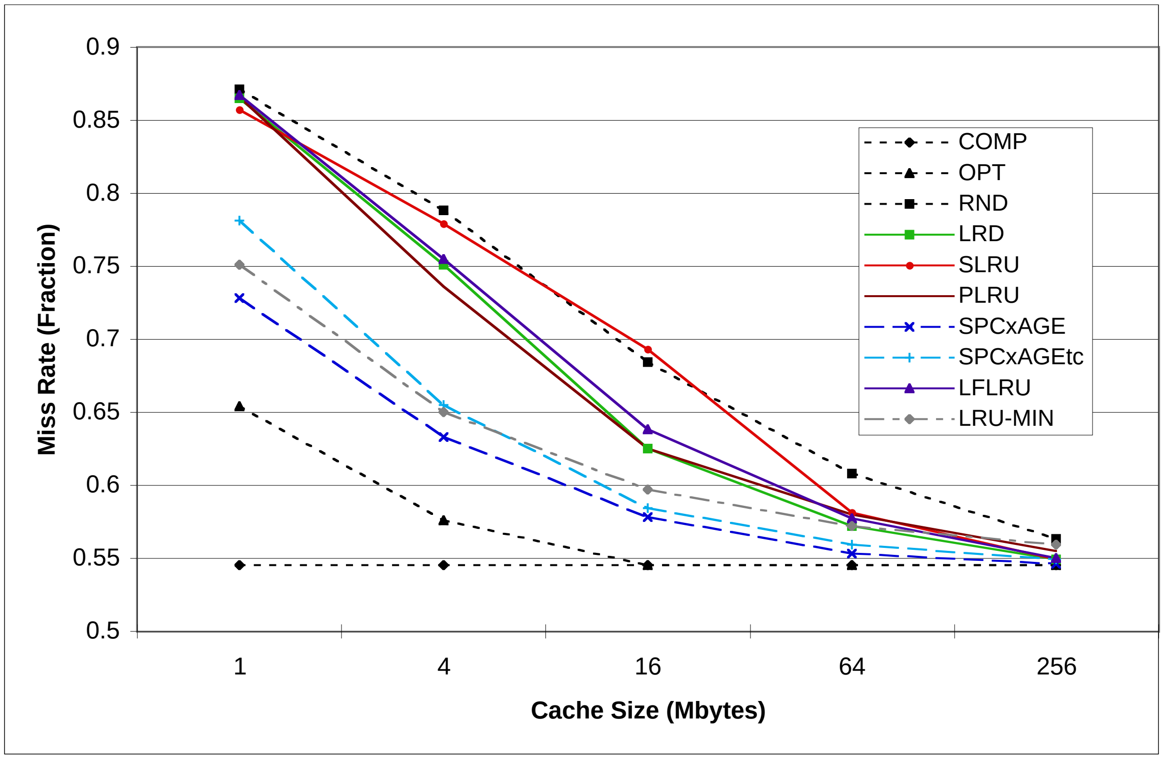


Fig. 6.62. Miss Rates of More Complex Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

graph still performed well enough for third place, while SPACExAGE and SizeLFLRU tied for first.

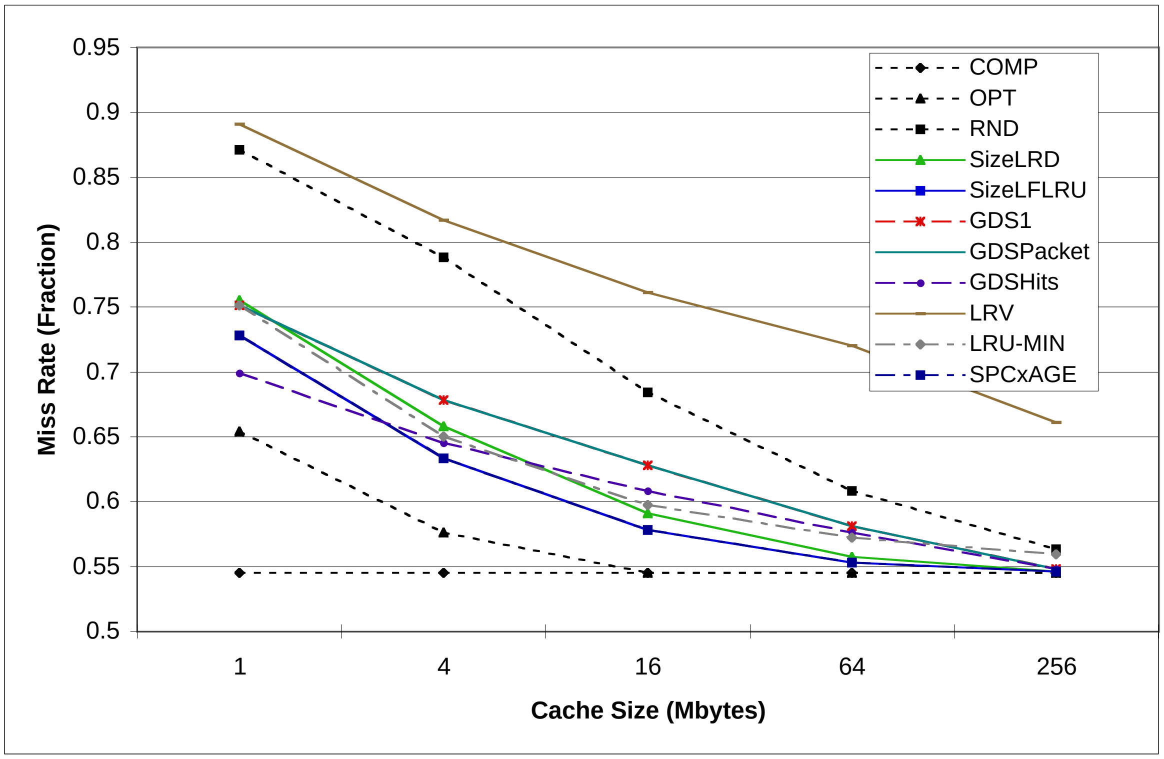


Fig. 6.63. Miss Rates of Complex Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

Among the simple algorithms when optimizing for byte miss rate in Figure 6.65, LRU performs best, and its results are forwarded to Figure 6.66.

Though it is difficult to see in Figure 6.66, LRD and SPACExAGEtc perform the best among the more complex cache replacement algorithms which can be seen more clearly in Figure 6.68. The results of the two algorithms are forwarded to Figure 6.67.

There is more separation among the curves in Figure 6.67, but it is still difficult to determine the best algorithms. Looking at the average byte miss rates in Figure 6.68, SPACExAGEtc and LRD tied for first place while SPACExAGE and SizeLFLRU tied for a close third. Also note that on average, PLRU and LFLRU perform have just 0.2% higher byte miss rate than these.

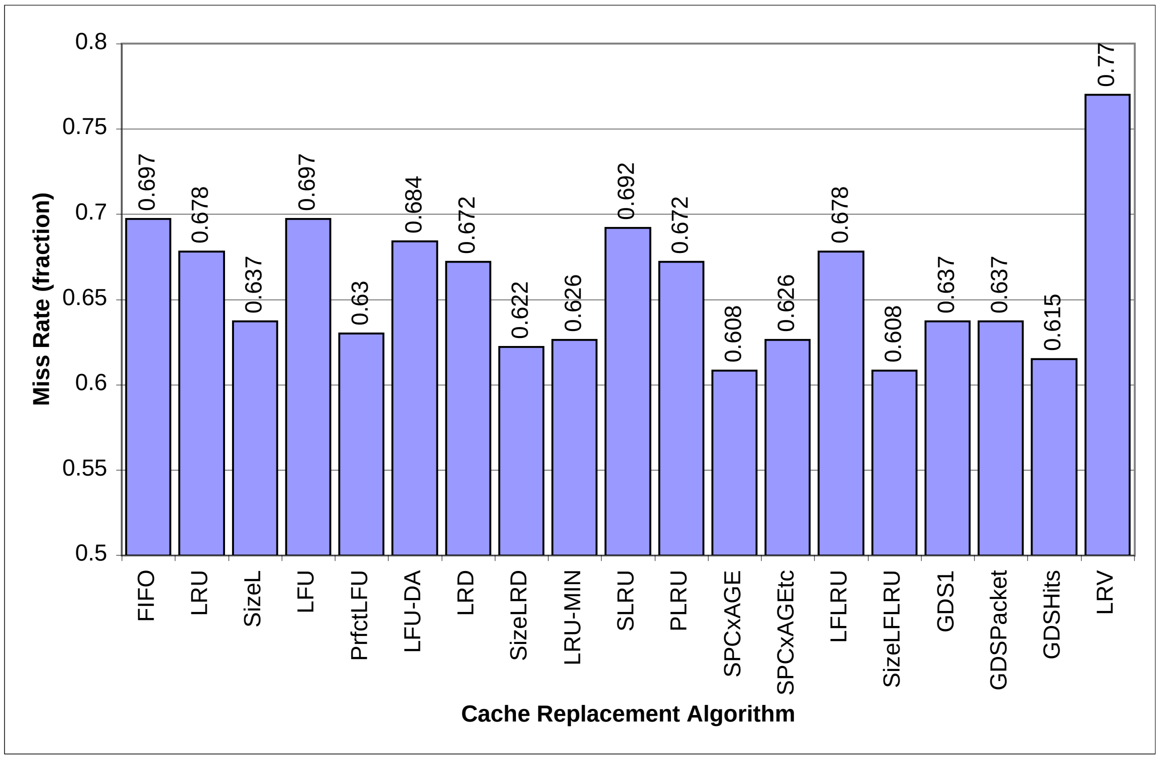


Fig. 6.64. Average Miss Rates on Purdue Stack Proxy Server stackproxy Trace.

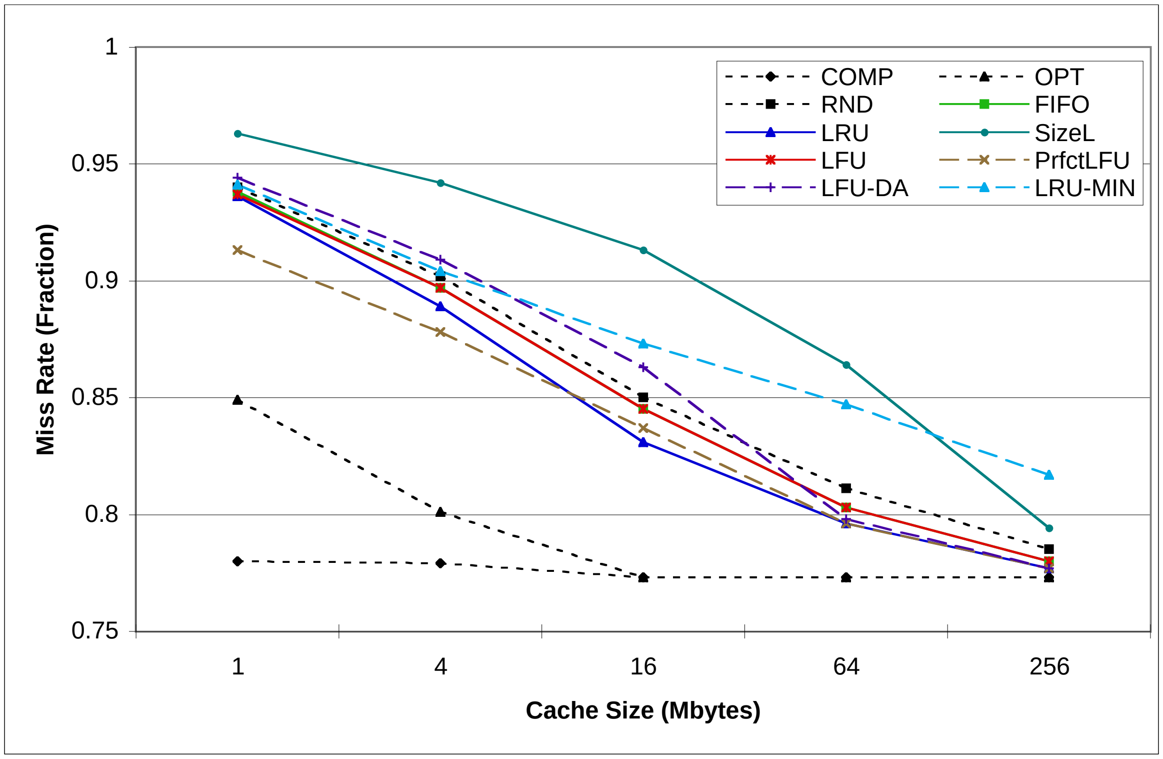


Fig. 6.65. Byte Miss Rates of Simple Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

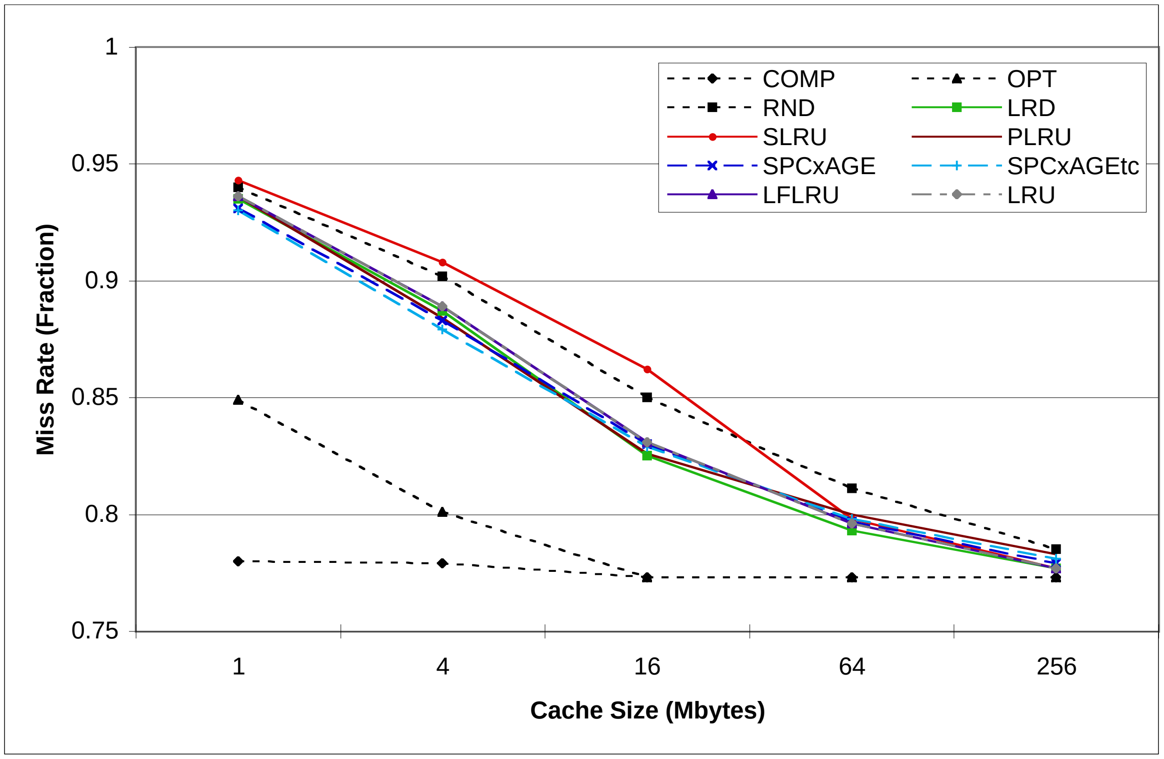


Fig. 6.66. Byte Miss Rates of More Complex Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

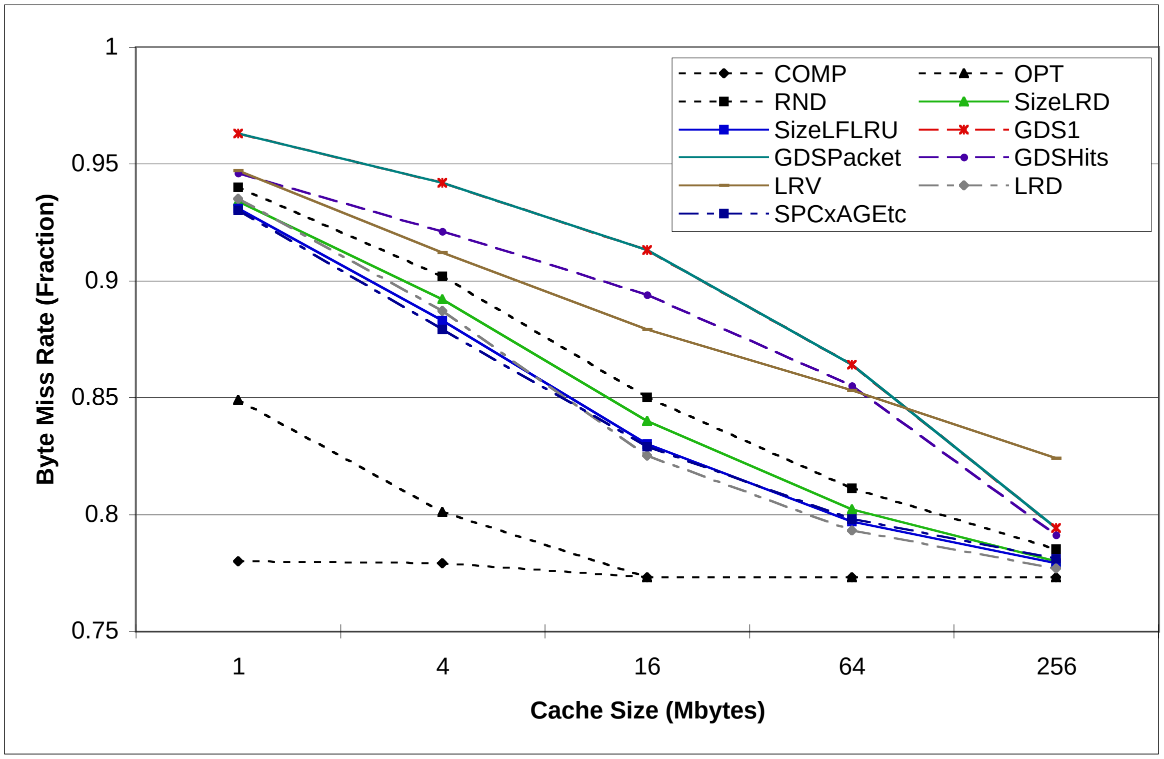


Fig. 6.67. Byte Miss Rates of Complex Cache Replacement Algorithms on Purdue Stack Proxy Server stackproxy Trace.

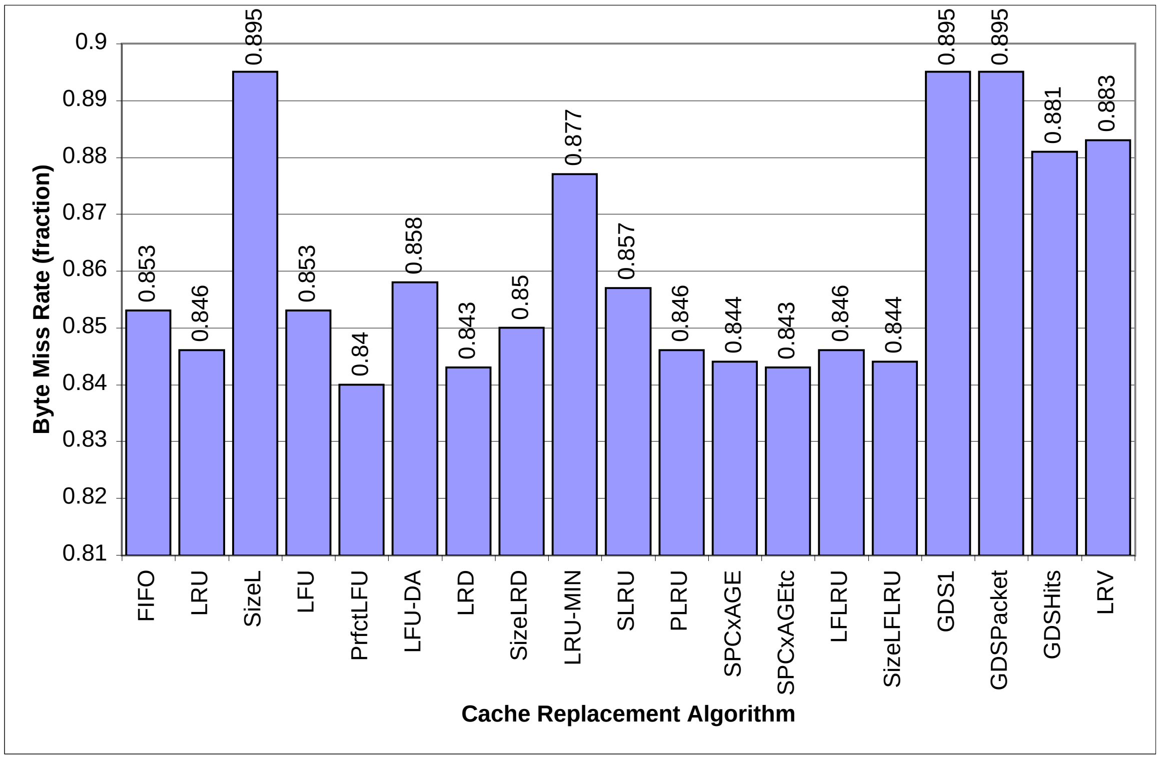


Fig. 6.68. Average Byte Miss Rates on Purdue Stack Proxy Server stackproxy Trace.

Virginia Tech BL Trace

For the Virginia Tech BL workload trace, the SizeL algorithm performs best among the simple cache replacement algorithms of Figure 6.69. The SizeL results are forwarded to Figure 6.70.

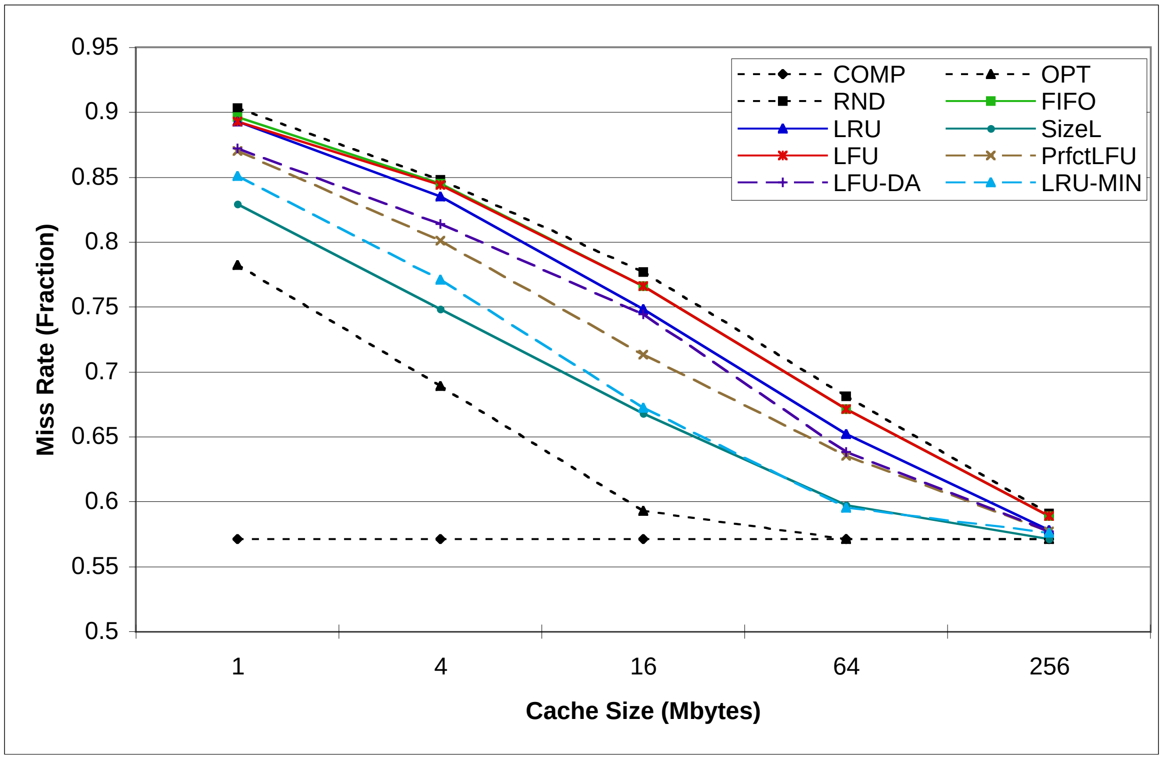


Fig. 6.69. Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech BL Trace.

On Figure 6.70, the SPACExAGE and SPACExAGEtc algorithms were best, and their results are forwarded to Figure 6.71.

From Figures 6.71 and 6.72, the GDSHits algorithm performed best with the Virginia Tech workload trace and the miss rate measure. The SPACExAGE and SizeLFLRU algorithms tied for second in average miss rate on Figure 6.72. When examining the graph in Figure 6.71, the two second place algorithms perform equally throughout the different cache sizes which implies that the access count does not

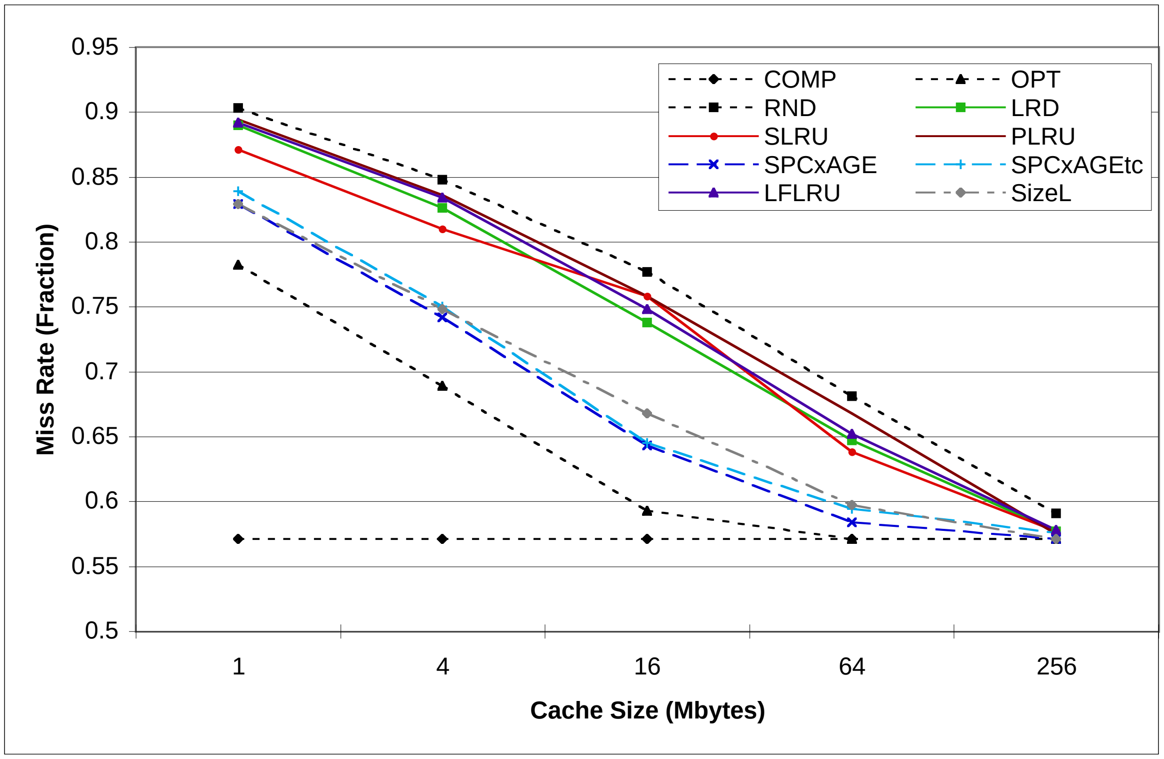


Fig. 6.70. Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech BL Trace.

contribute at all in the SizeLFLRU algorithm for this workload trace. Interestingly, GDSHits performs best for smaller cache sizes (which is most important), but SPACExAGE and SizeLFLRU perform better than GDSHits for larger cache sizes.

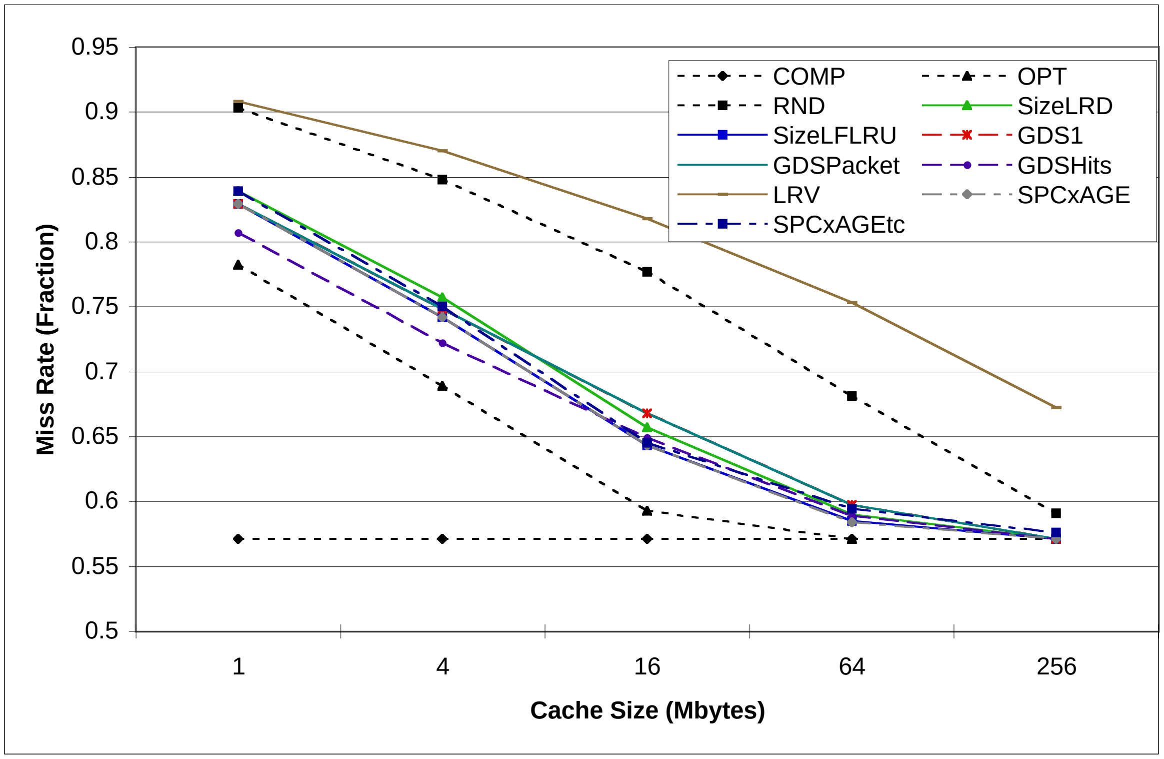


Fig. 6.71. Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech BL Trace.

Among the simple cache replacement algorithms for byte miss rate in Figure 6.73, the LFU-DA algorithm performs best, and its results forwarded to Figure 6.74.

In Figure 6.74, the LFU-DA and LRD algorithms are the best performers, especially in the middle cache sizes. Their results are then forwarded to Figure 6.75.

In Figure 6.75, one can see that LFU-DA and LRD perform the best overall. Figure 6.76 shows that the two algorithms have the same average byte miss rate, while Figure 6.75 shows that LFU-DA performs best for the smaller and larger cache sizes while LRD performs best for middle cache sizes. Since their average byte miss rate are equal, they will be considered tied for first place. LFLRU has the third best

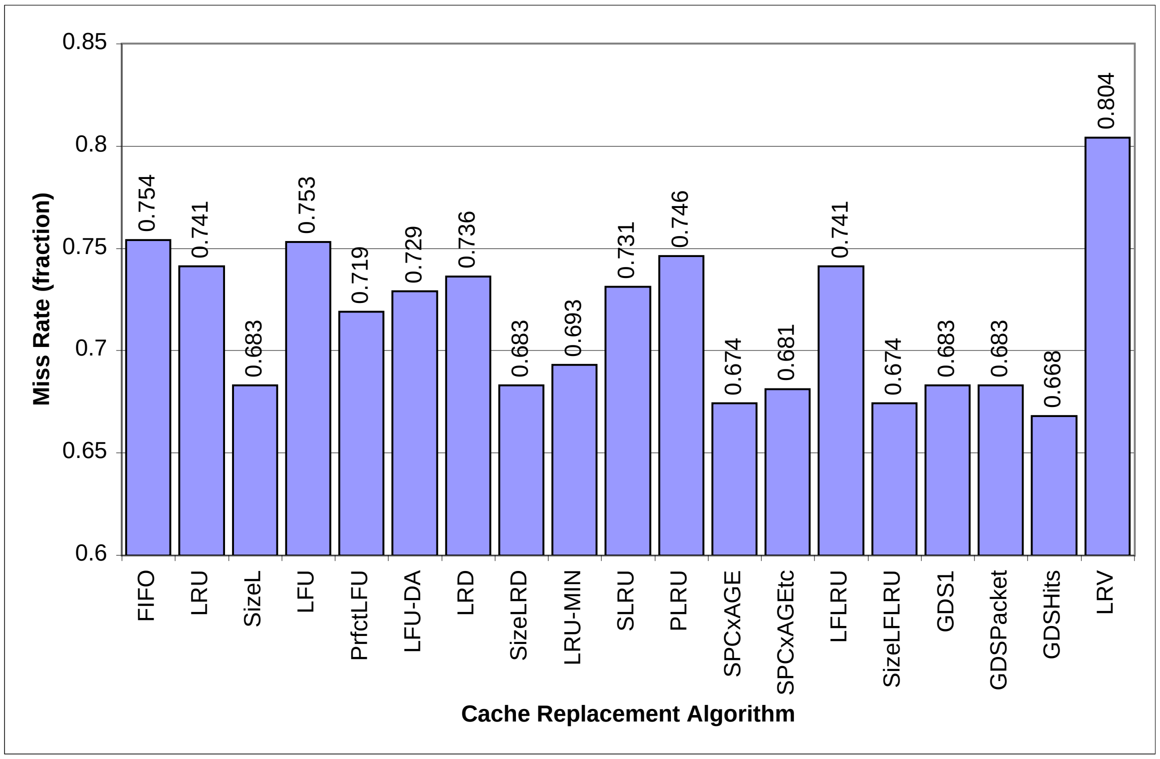


Fig. 6.72. Average Miss Rates on Virginia Tech BL Trace.

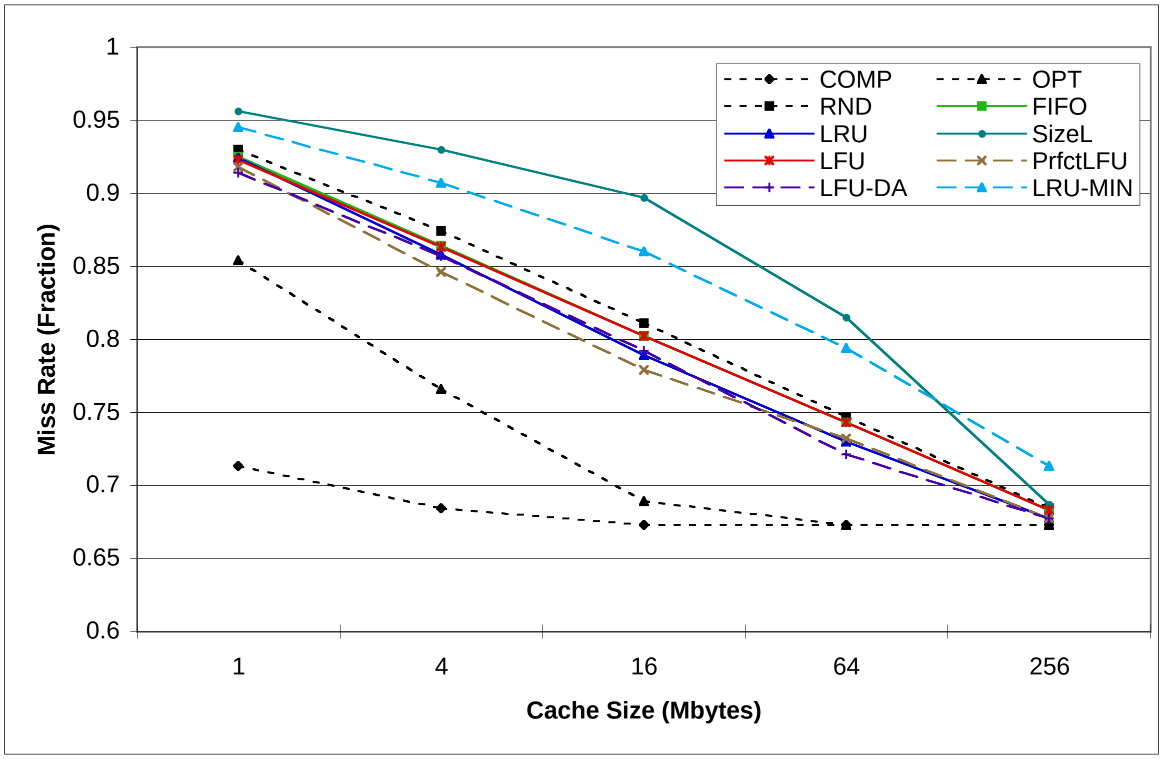


Fig. 6.73. Byte Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech BL Trace.

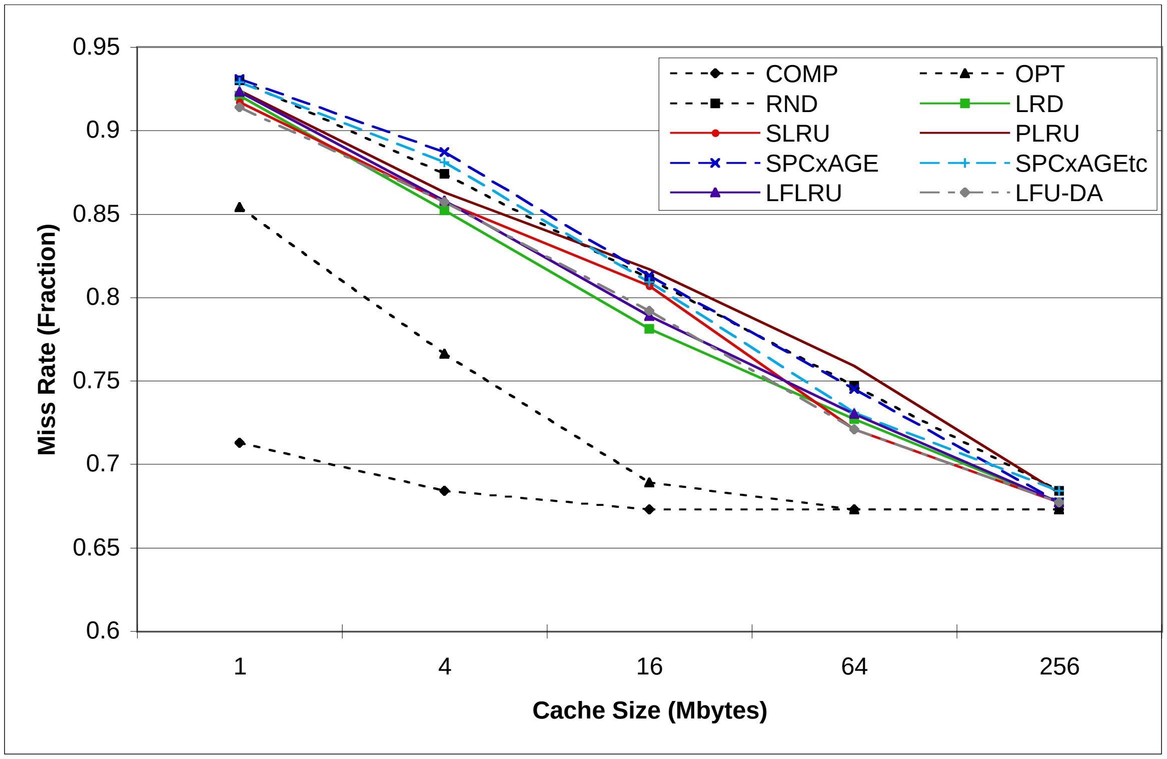


Fig. 6.74. Byte Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech BL Trace.

average, while LRU and SLRU are have an average byte miss rate that is only 0.1% worse than LFLRU.

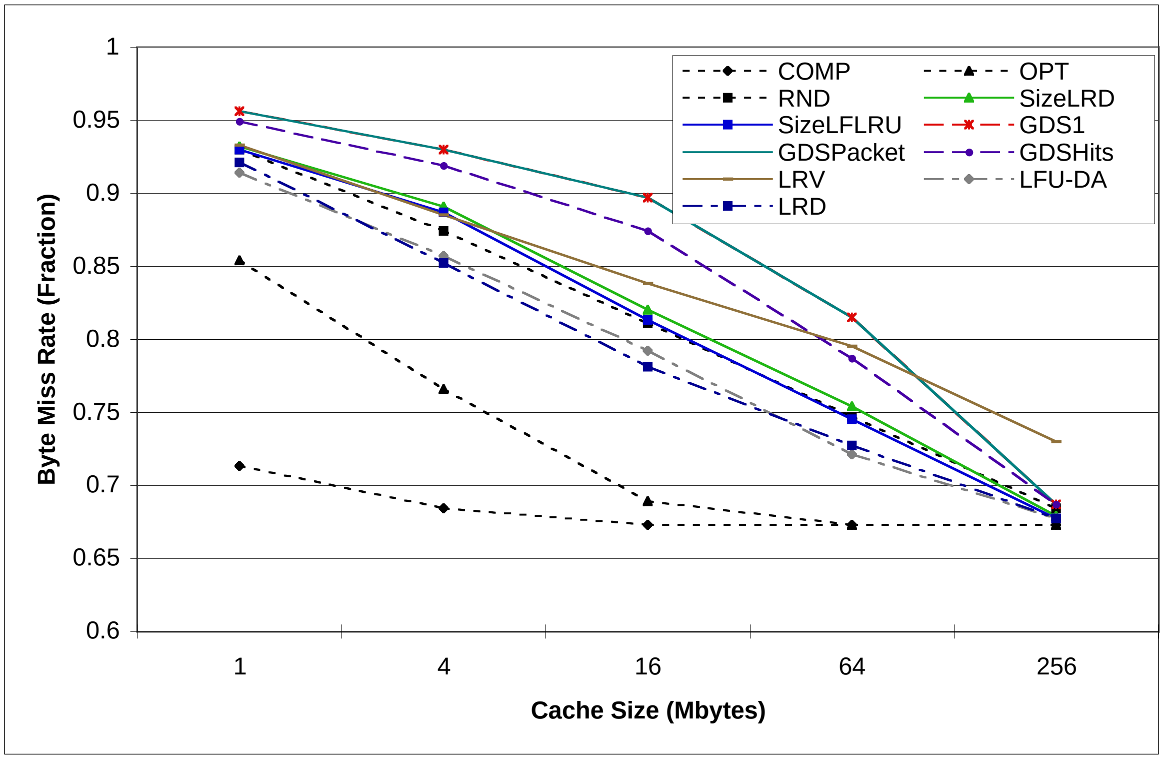


Fig. 6.75. Byte Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech BL Trace.

Virginia Tech G Trace

In Figure 6.77, the SizeL algorithm performs the best by a sizeable margin. Hence the SizeL algorithm results are forwarded to Figure 6.78.

The SizeL and SPACExAGE algorithm results from Figure 6.78 are forwarded to Figure 6.79 because they are the two top performing algorithms on the more complex algorithm miss rate graph. Interestingly, the SPACExAGE and SPACExAGEtc algorithms have a difference in miss rate of over 1%, so there can be a significant difference between their performance.

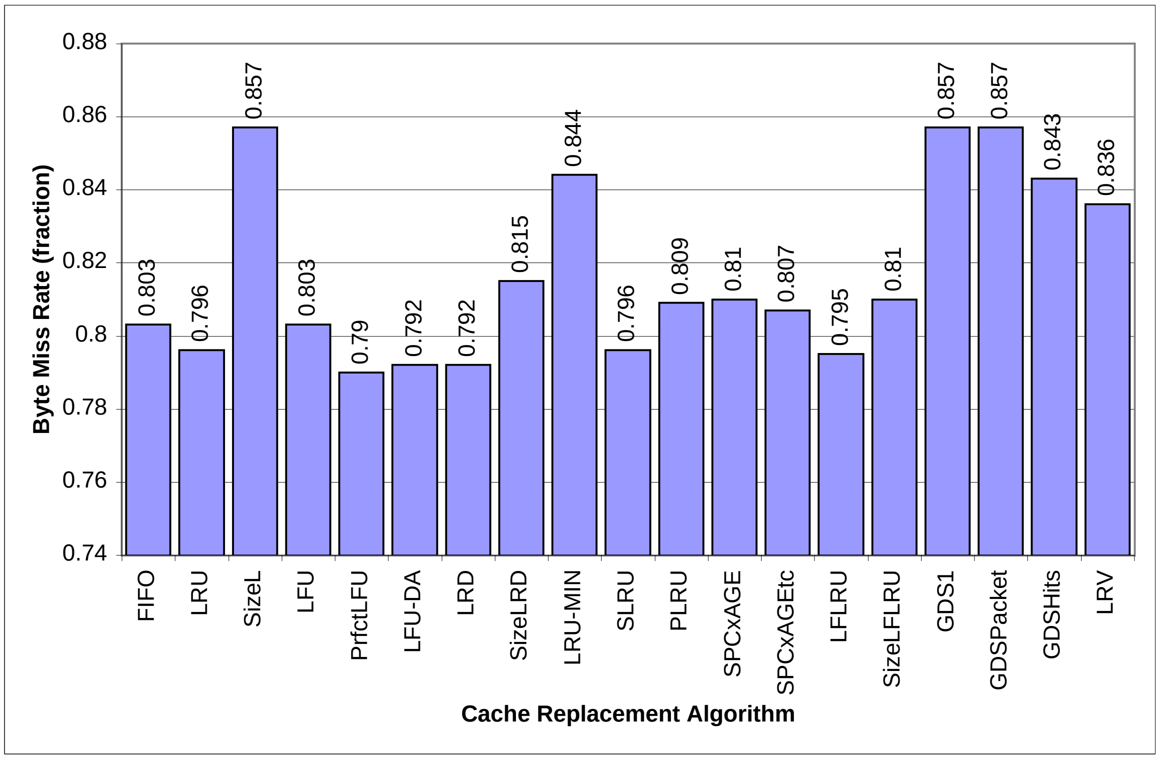


Fig. 6.76. Average Byte Miss Rates on Virginia Tech BL Trace.

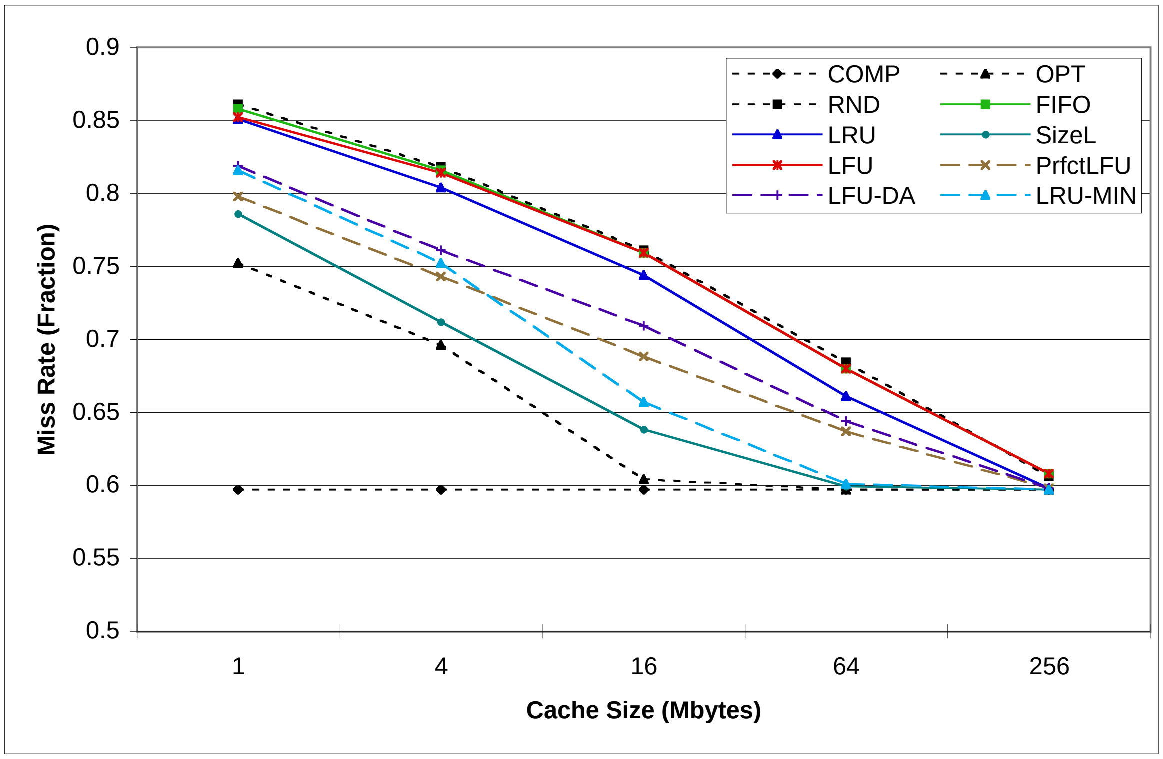


Fig. 6.77. Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech G Trace.

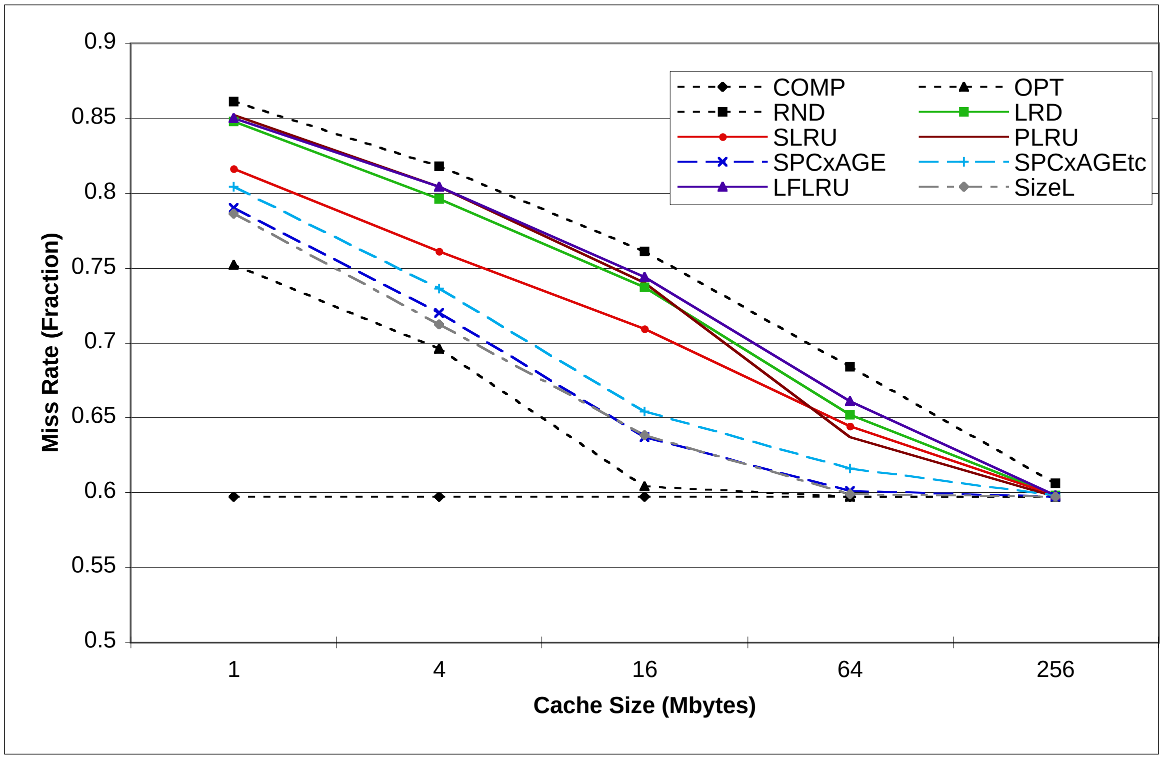


Fig. 6.78. Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech G Trace.

Figures 6.79 and 6.80 show the GDSHits algorithm performs the best of all of the algorithms. There is a three way tie for second and they all have the exact same miss rates for all five cache sizes. These three algorithms are SizeL, GDS1, and GDSPacket. On all three of these comparison graphs, Figures 6.77, 6.78, and 6.79, it is interesting to note that due to the small size of the total workload trace (just under 400 megabytes according to Table 5.27), almost all of the algorithms perform the best possible for the 256 megabyte cache size. But it is more interesting to note that some of the cache replacement algorithms manage the cache so well that they are able to achieve the same miss rate for only a 64 megabyte cache size.

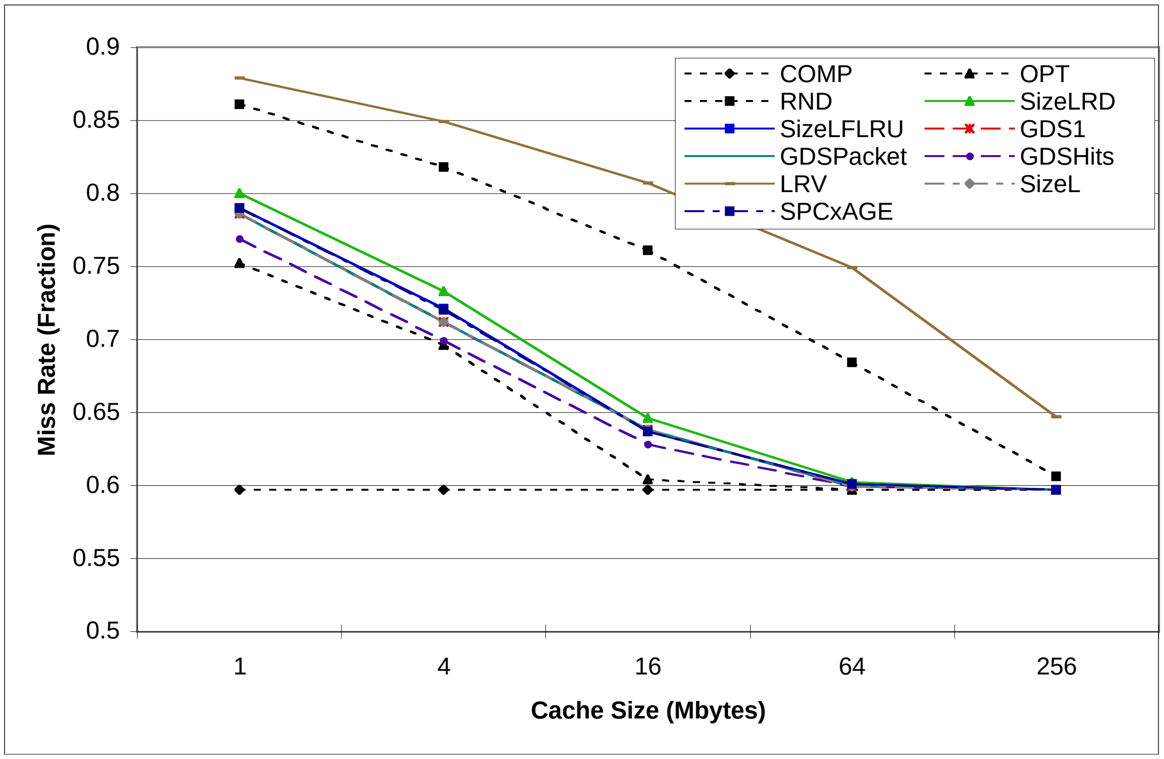


Fig. 6.79. Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech G Trace.

For the byte hit rate measure among the simple cache replacement algorithms of Figure 6.81, the LFU-DA algorithm performs best (after the PerfectLFU algorithm). So the LFU-DA results are forwarded to Figure 6.82.

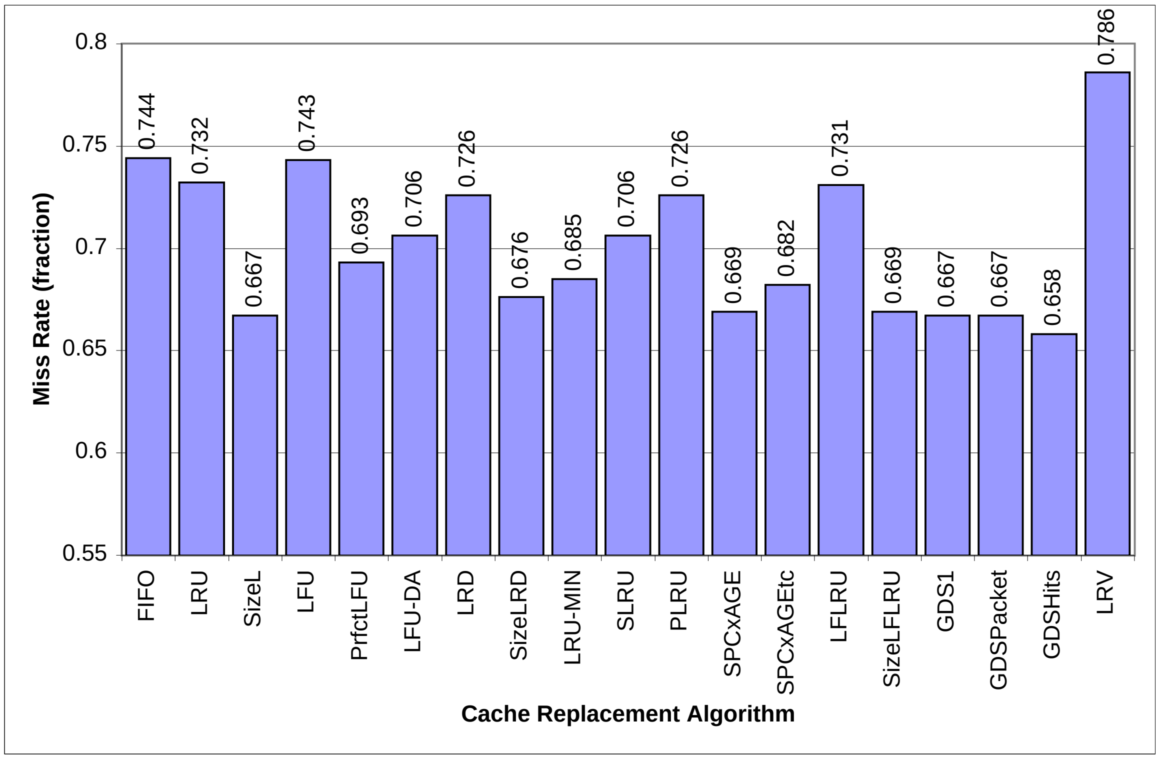


Fig. 6.80. Average Miss Rates on Virginia Tech G Trace.

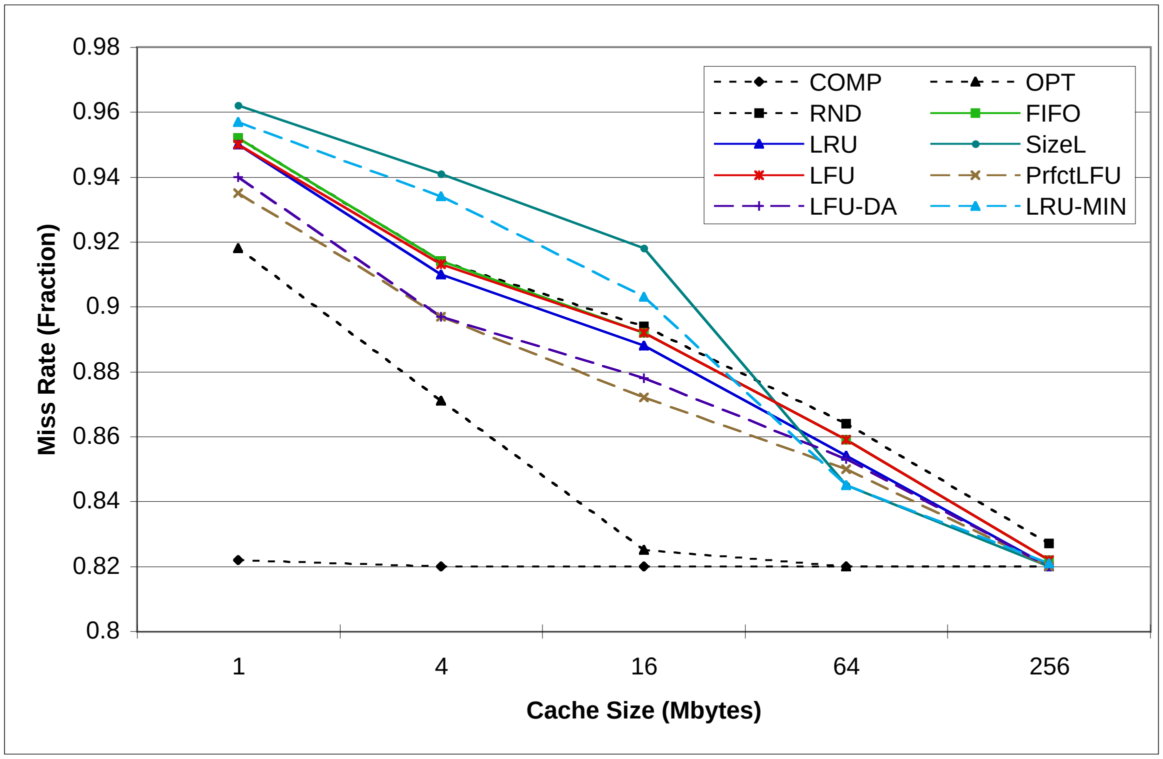


Fig. 6.81. Byte Miss Rates of Simple Cache Replacement Algorithms on Virginia Tech G Trace.

Figure 6.82 shows that LFU-DA performs best for smaller cache sizes while SPACExAGE performs best for larger cache sizes. Those two algorithm results are forwarded to Figure 6.83.

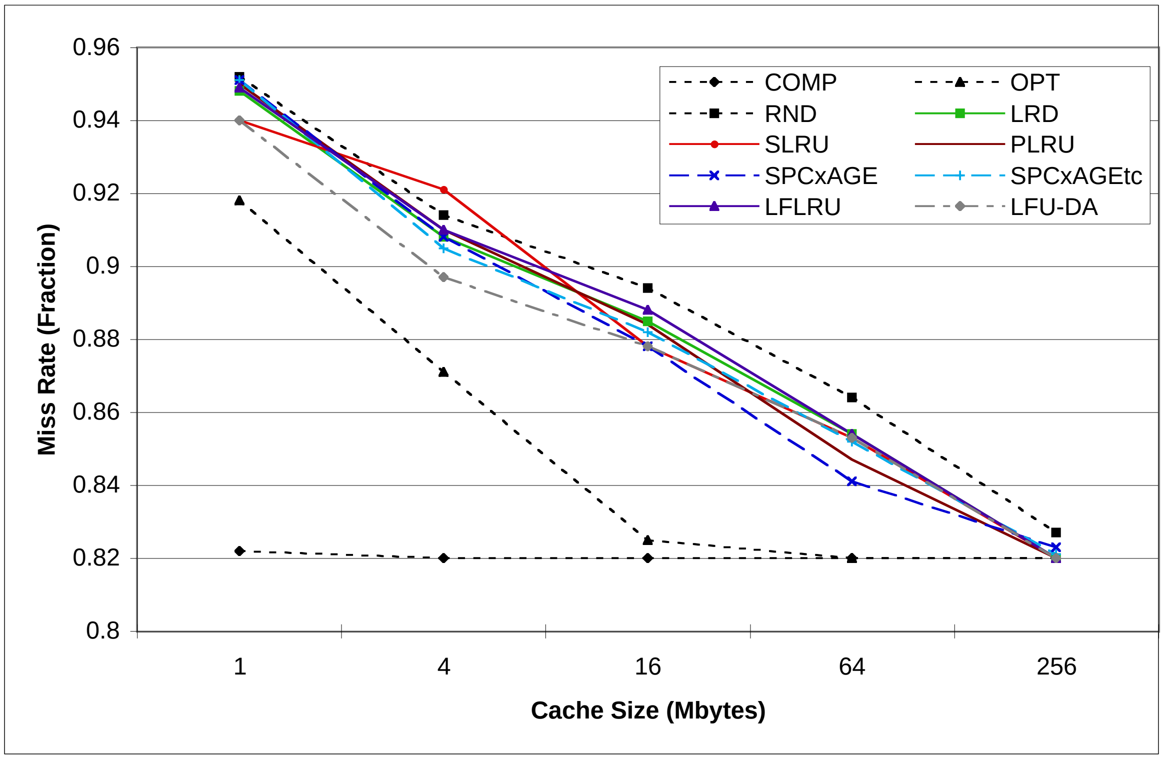


Fig. 6.82. Byte Miss Rates of More Complex Cache Replacement Algorithms on Virginia Tech G Trace.

For the Virginia Tech G workload trace for the byte miss rate measure, the top performing algorithm is LFU-DA as shown in Figure 6.83. It is especially good at smaller cache sizes, but it does have an abnormally high byte miss rate at the 64 megabyte cache size. The SizeLFLRU algorithm has an average byte miss rate that is just 0.1% worse than LFU-DA to take second place, and SPACExAGE has an average byte miss rate that is just 0.1% worse than SizeLFLRU to take third place. The SizeLFLRU and SPACExAGE algorithm results are more consistent over all of the cache sizes than LFU-DA, but LFU-DA still has a lower average byte miss rate. It is significant to note that the most differentiation between the performances of these algorithms is in the smaller cache sizes.

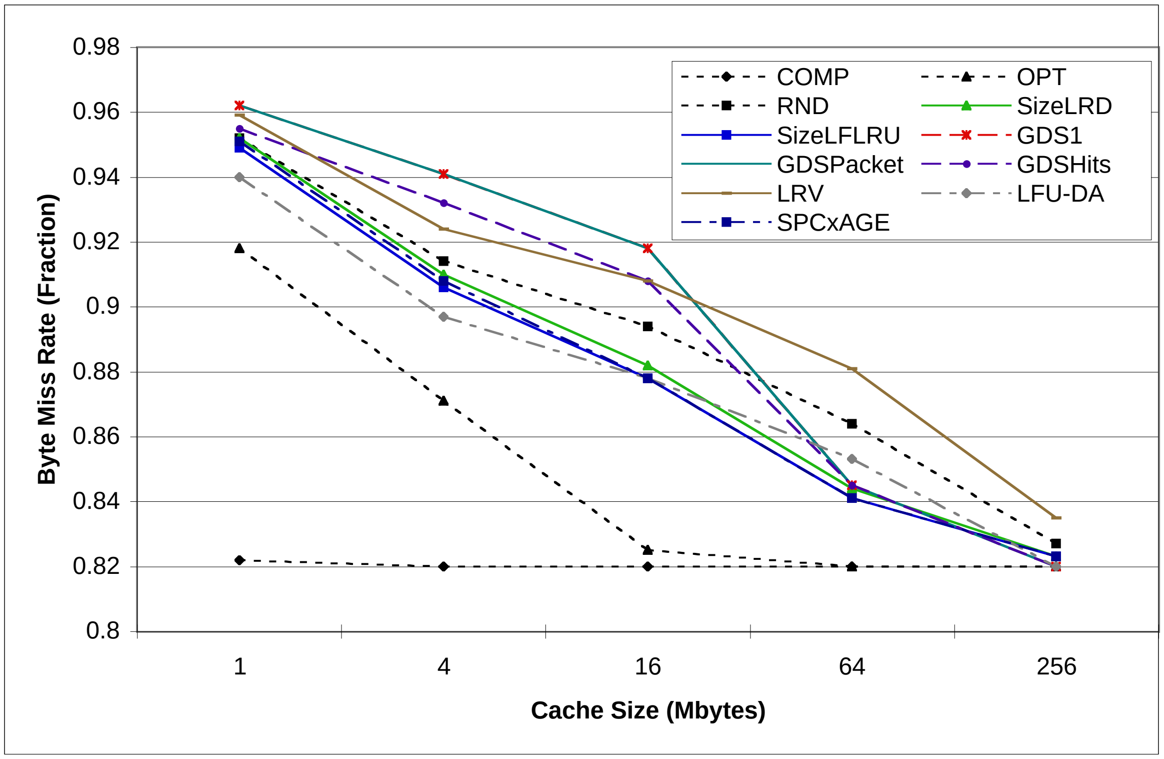


Fig. 6.83. Byte Miss Rates of Complex Cache Replacement Algorithms on Virginia Tech G Trace.

Client Proxy Server Traces Summary

For this domain's summary, the table of top performing algorithms for each trace and both measures is presented in Table 6.4. In this table, one can see that the best algorithms for this domain are GDSHits in most cases when optimizing for miss rate, and LRU-DA when optimizing for byte miss rate. Among the best algorithms for miss rate, SPACExAGE and SizeLFLRU are also fairly consistently among the top algorithms, while LRD and SLRU show some consistency among the top contenders for byte miss rate. These results are similar, though not as consistent, to the findings of the origin server workload traces and the network proxy server workload traces.

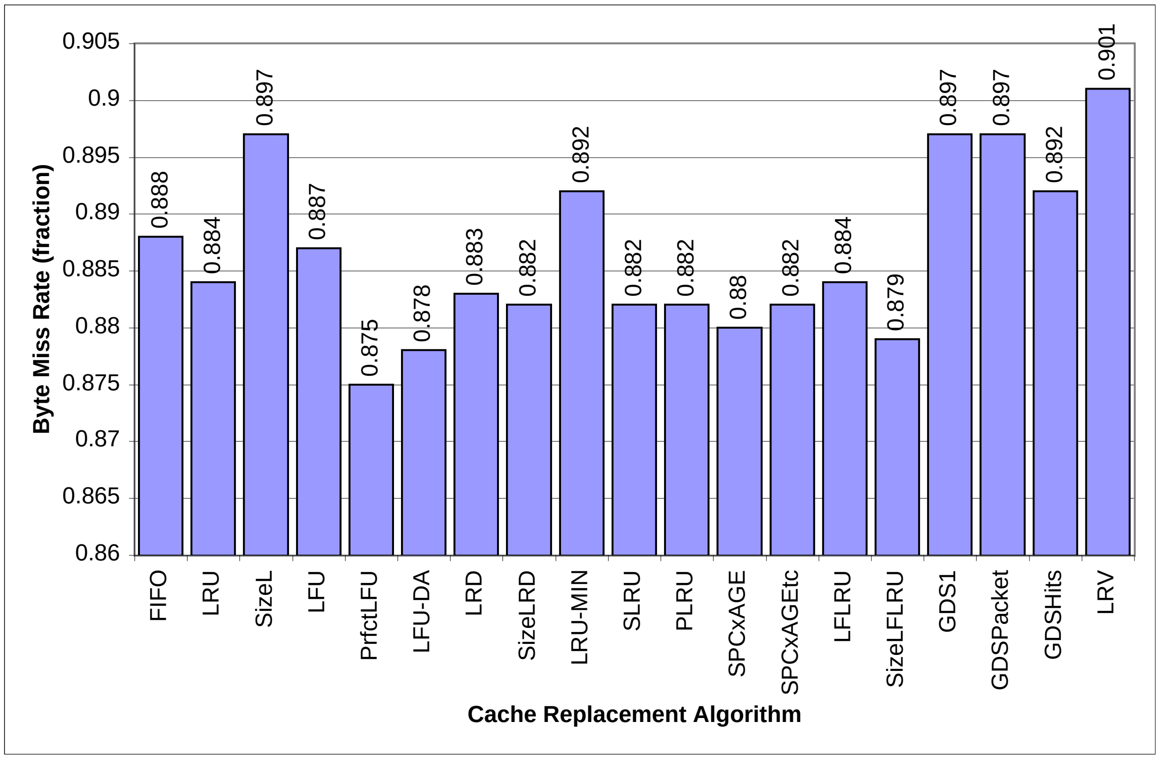


Fig. 6.84. Average Byte Miss Rates on Virginia Tech G Trace.

Table 6.4
Summary of Top Cache Replacement Algorithms for Client Proxy Server Traces

Trace	boeing	stack	VaTech BL	VaTech G
Miss Rate	GDSHits (1st) GDS1 (2nd) GDSPacket (3rd)	SizeLFLRU (1st-tie) SPACExAGE (1st-tie) GDSHits (3rd)	GDSHits (1st) SPACExAGE (2nd-tie) SizeLFLRU (2nd-tie)	GDSHits (1st) SizeL (2nd-tie) GDS1 (2nd-tie) GDSPacket (2nd-tie)
Byte Miss Rate	SLRU (1st) LFU-DA (2nd) LRD (3rd)	SPACExAGEtc (1st-tie) LRD (1st-tie) SizeLFLRU (3rd-tie) SPACExAGE (3rd-tie)	LFU-DA (1st-tie) LRD (1st-tie) LFLRU (3rd)	LFU-DA (1st) SizeLFLRU (2nd) SPACExAGE (3rd)

For these comparisons, the average miss rate graphs of Figures 6.56, 6.64, 6.72, and 6.80 are shrunk and brought together in Figure 6.85. Comparing all of the average miss rate graphs shows that there are definitely algorithms that consistently perform relatively poorly in terms of miss rates. These are FIFO, LRU, LFU, LFU-DA, LRD, SLRU, PLRU, LFLRU, and LRV. FIFO, LRU, and LFU probably do not have adequate information about objects to make good replacement decisions. SLRU and PLRU only perform marginally better than LRU and probably suffer from the same lack of information; also in the case of PLRU, file type does not help make many decisions. Since LFU and LFU-DA are on this list, it indicates that access count isn't that helpful with making replacement decisions for the client traces. This indication is reinforced by the fact that LRD and LFLRU are also on the list; these two algorithms just combine access count with FIFO and LRU, respectively, and LRD and LFLRU don't perform much better than FIFO and LRU. Conversely, the algorithms that take object size into account consistently perform well for the miss rate measure. This trend starts with SizeL and includes SizeLRD, SizeLFLRU, SPACExAGE, SPACExAGEtc, GDS1, GDSPacket, and GDSHits.

Switching to the average byte miss rate graphs of Figures 6.60, 6.68, 6.76, and 6.84, they also show several algorithms that consistently perform poorly. These graphs of Figures 6.60, 6.68, 6.76, and 6.84 are shrunk and brought together in Figure 6.86. SizeL, LRU-MIN, GDS1, GDSPacket, GDSHits, and LRV are on this poorly performing list. What they all have in common is a reliance on the object size statistic. However, those algorithms that combine access count, a time measure, and size do not perform poorly. For instance, LFLRU performs better than LRU, and SizeLFLRU improved upon the LFLRU performance. But while LRD improves on the FIFO performance, SizeLRD generally performs worse than LRD (except on the Virginia Tech G trace). SLRU and PLRU are not consistent in improving the byte miss rate of LRU; for the Boeing workload trace, SLRU is the top algorithm while for the Stack workload trace, it does not perform that well. PLRU performs poorly for the Boeing workload trace while it performs respectfully for the Stack and Virginia Tech G workload traces. For each of the workload traces, there is not a sizeable performance difference between SPACExAGE and SPACExAGEtc; they are all within a few tenths of a percent of byte miss rates. So for the client proxy servers the way time is stored does not matter that much. Finally, LRV is consistently a poor performer for both miss rate and byte miss rate.

6.1.4 Video Server Traces

OnCommand On-Demand Movie Server Trace

With the OnCommand system, the best performing simple cache replacement algorithm was LFU as can be seen to some extent in Figure 6.87.

In Figure 6.88, one can see (though it is rather difficult) that LFU and LFLRU perform the best.

Then in Figure 6.89, LFU, LFLRU, SizeLFLRU, and GDSHits all perform fairly similarly with LFU, LFU-DA, and GDSHits first, second, and third, respectively, according to Figure 6.90.

Figures 6.87, 6.88, 6.89, and 6.90 show the three comparative byte miss rate line graphs and the average byte miss rate graph for the OnCommand trace. Since the movie sizes were all assumed to be the same (7200 seconds), there is no difference between these graph results and those for the miss rates above. Therefore, there is no reason to go through the cascading process for these results; the graphs, however, are included for completeness.

DVJ2 Educational Multimedia Video Server Trace

On the DVJ2 system, students watch videos of lectures, recitations, and demonstrations. For the simple cache replacement algorithms, Figure 6.95 shows that LFU and LRU perform the best with LFU being slightly better. The LFU results are then forwarded to Figure 6.96.

In Figure 6.96 for the more complex cache replacement algorithms, LFLRU and and LFU perform the best on average so their results are forwarded to Figure 6.97.

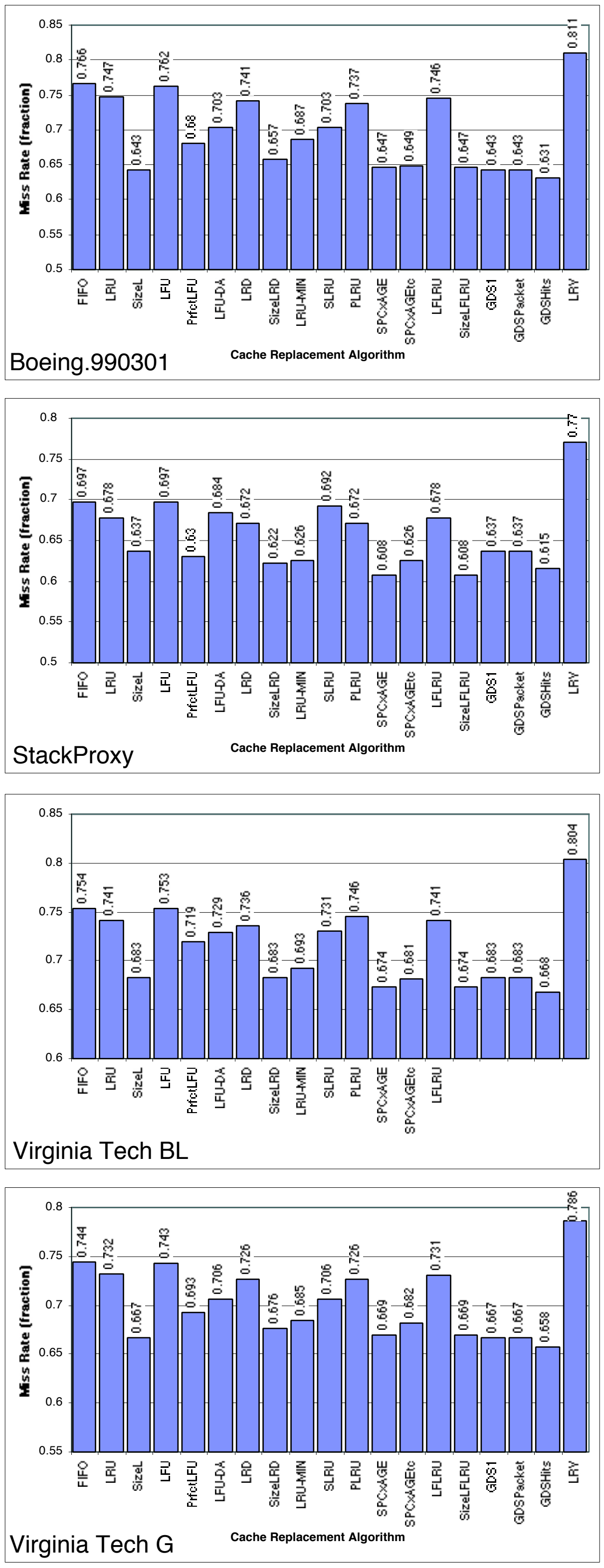


Fig. 6.85. Composite of Average Miss Rates for Client Proxy Server Traces.

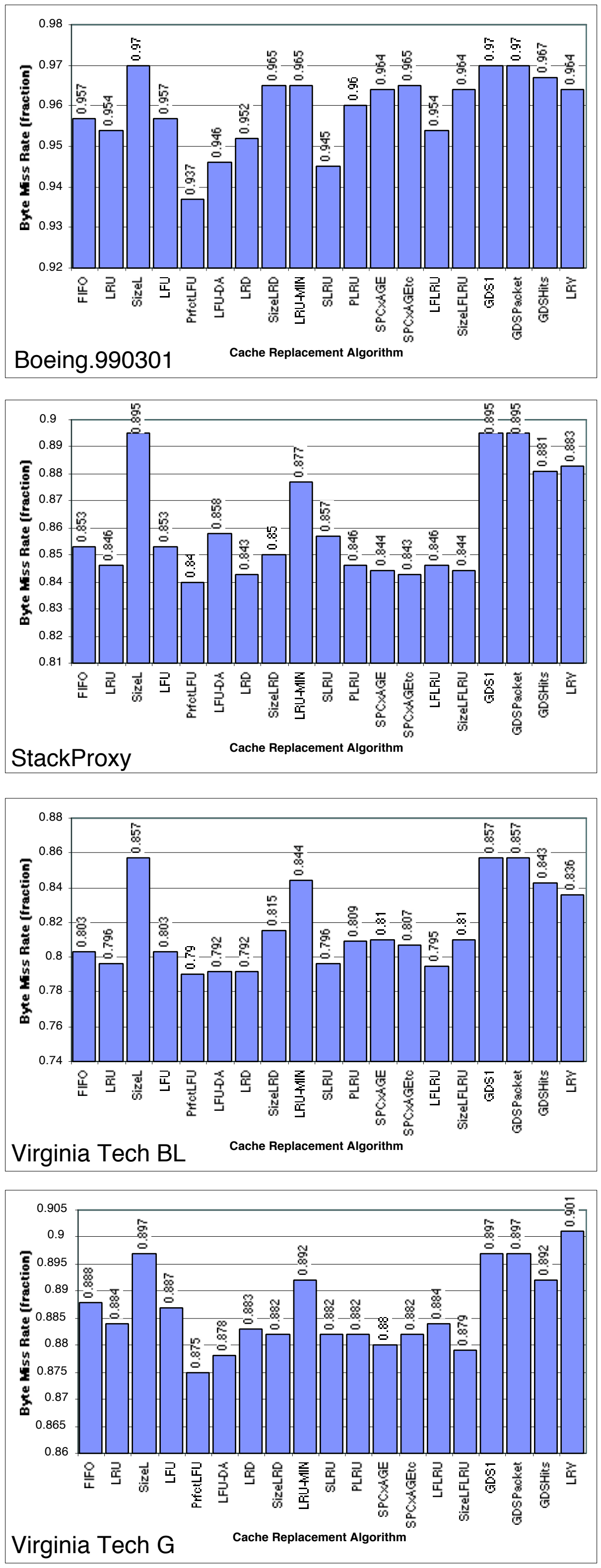


Fig. 6.86. Composite of Average Byte Miss Rates for Client Proxy Server Traces.

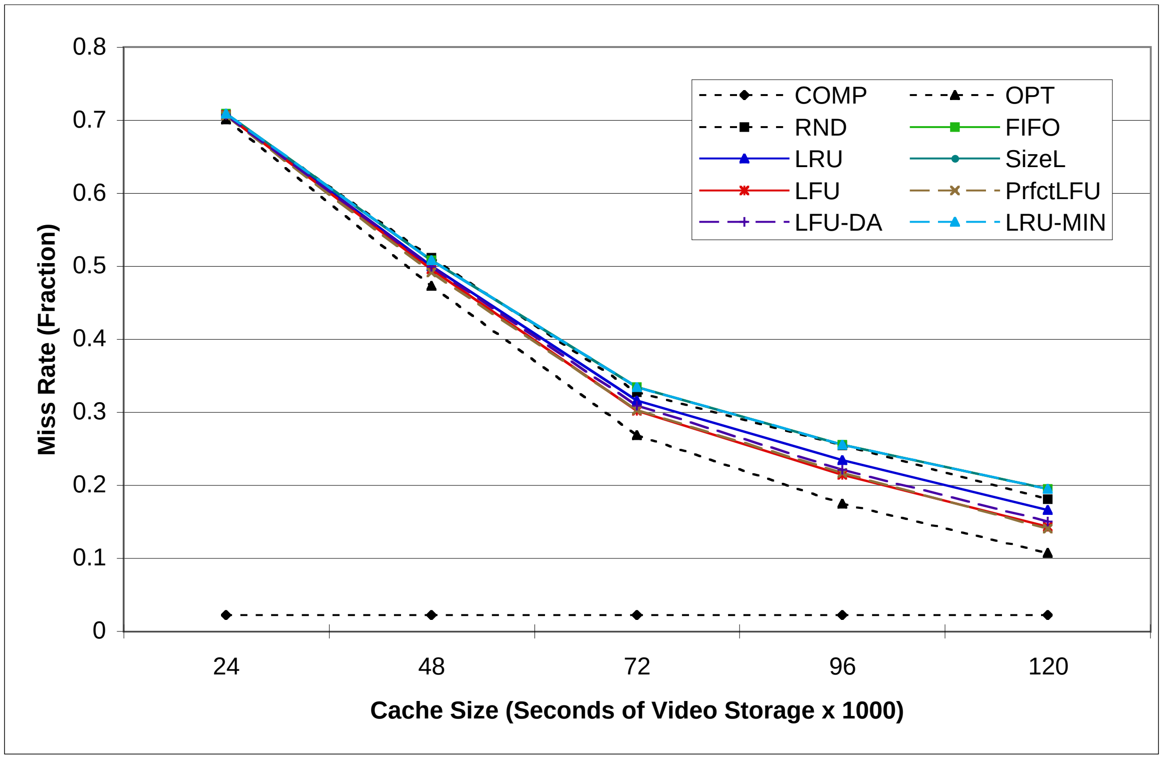


Fig. 6.87. Miss Rates of Simple Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

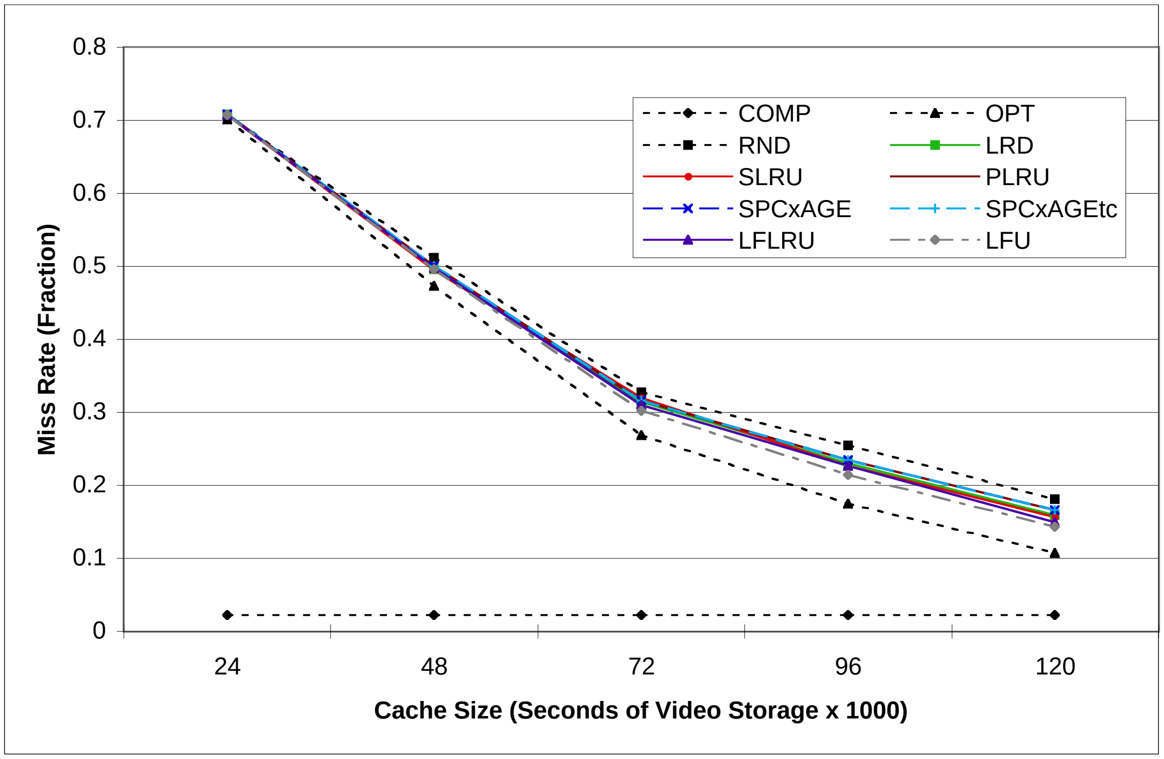


Fig. 6.88. Miss Rates of More Complex Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

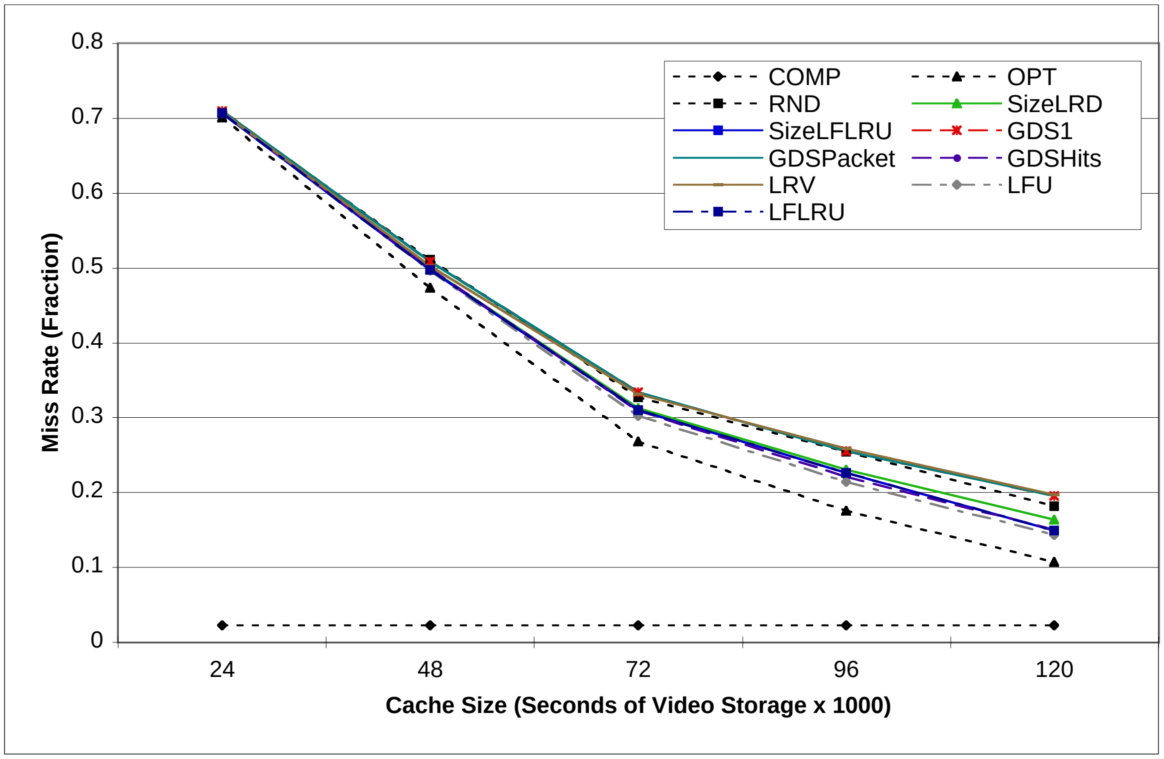


Fig. 6.89. Miss Rates of Complex Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

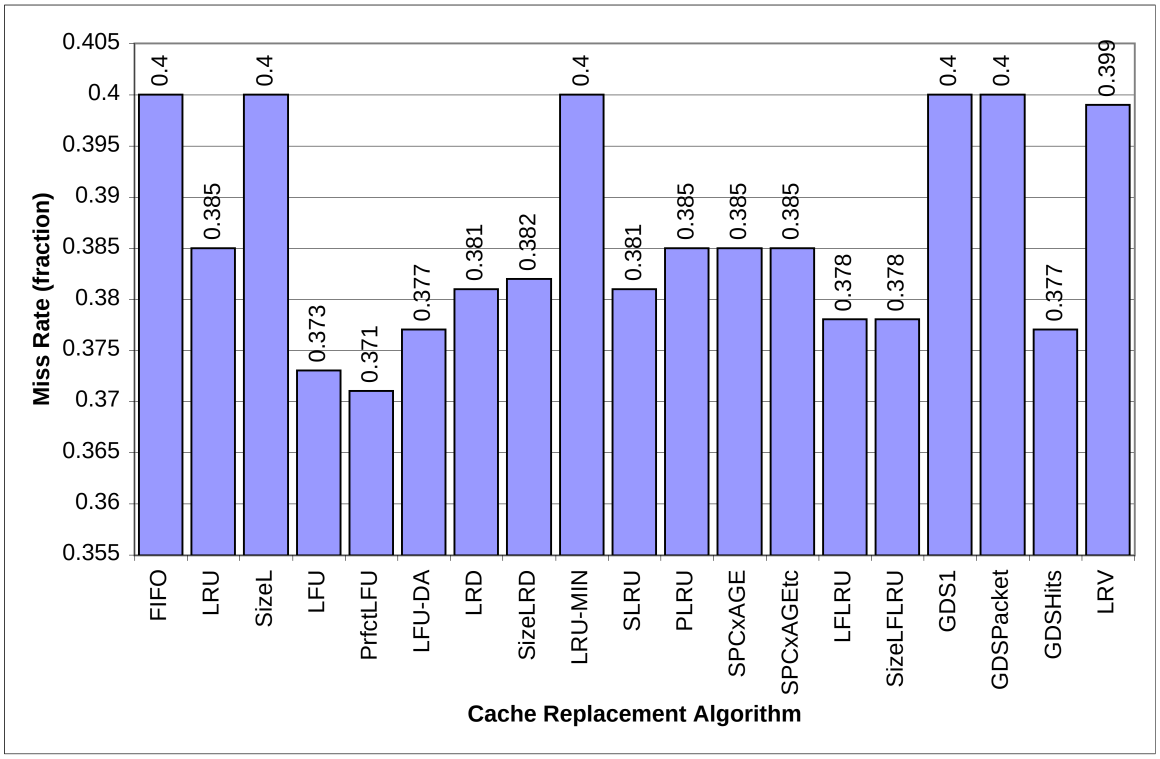


Fig. 6.90. Average Miss Rates on OnCommand On-Demand Movie Server daysum1 Trace.

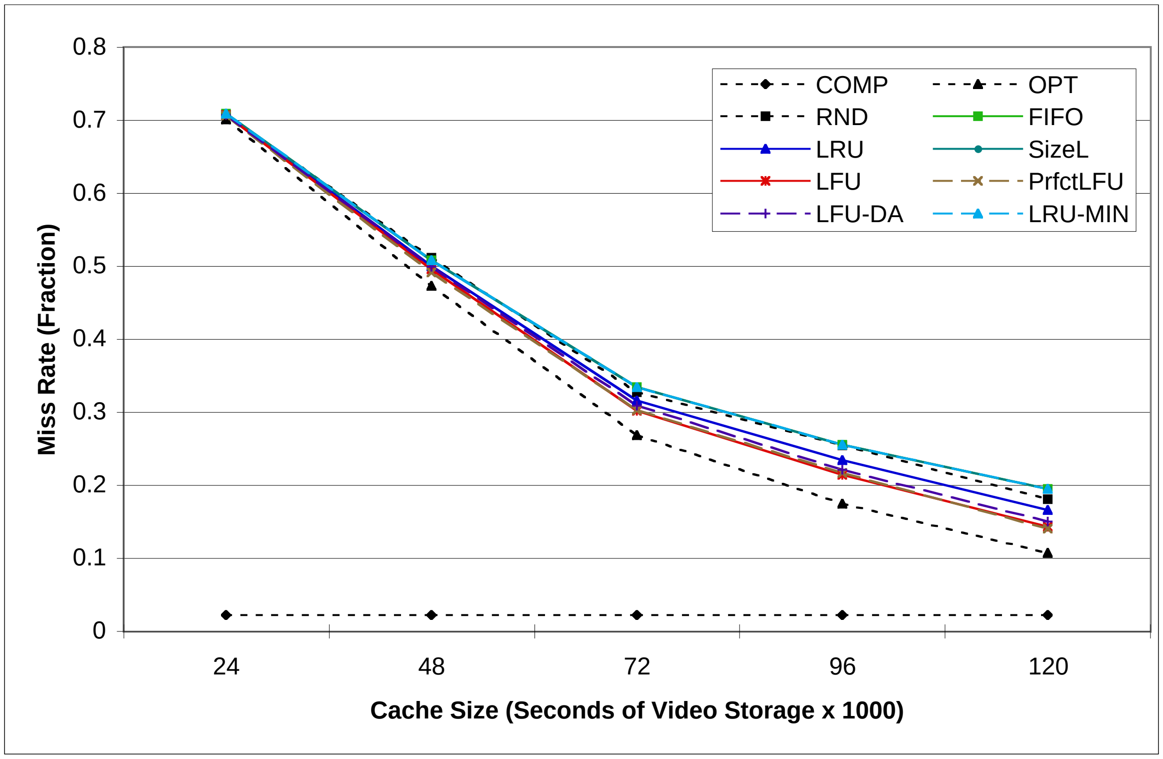


Fig. 6.91. Byte Miss Rates of Simple Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

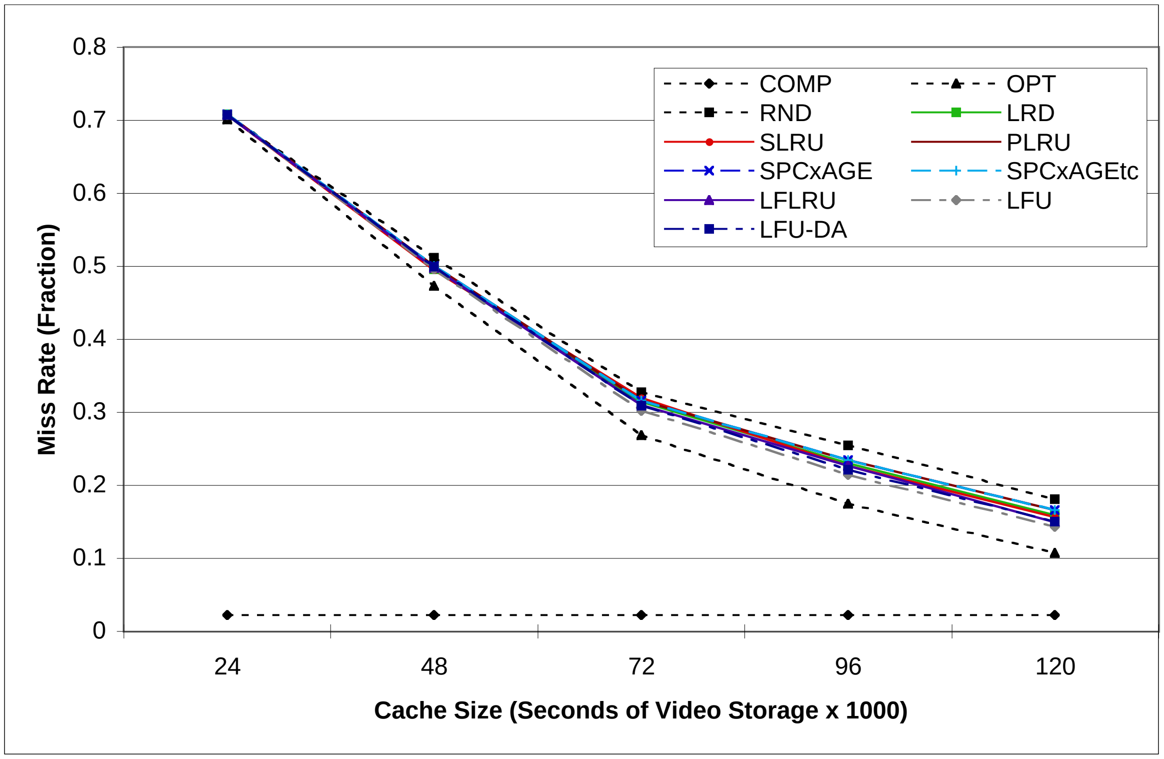


Fig. 6.92. Byte Miss Rates of More Complex Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

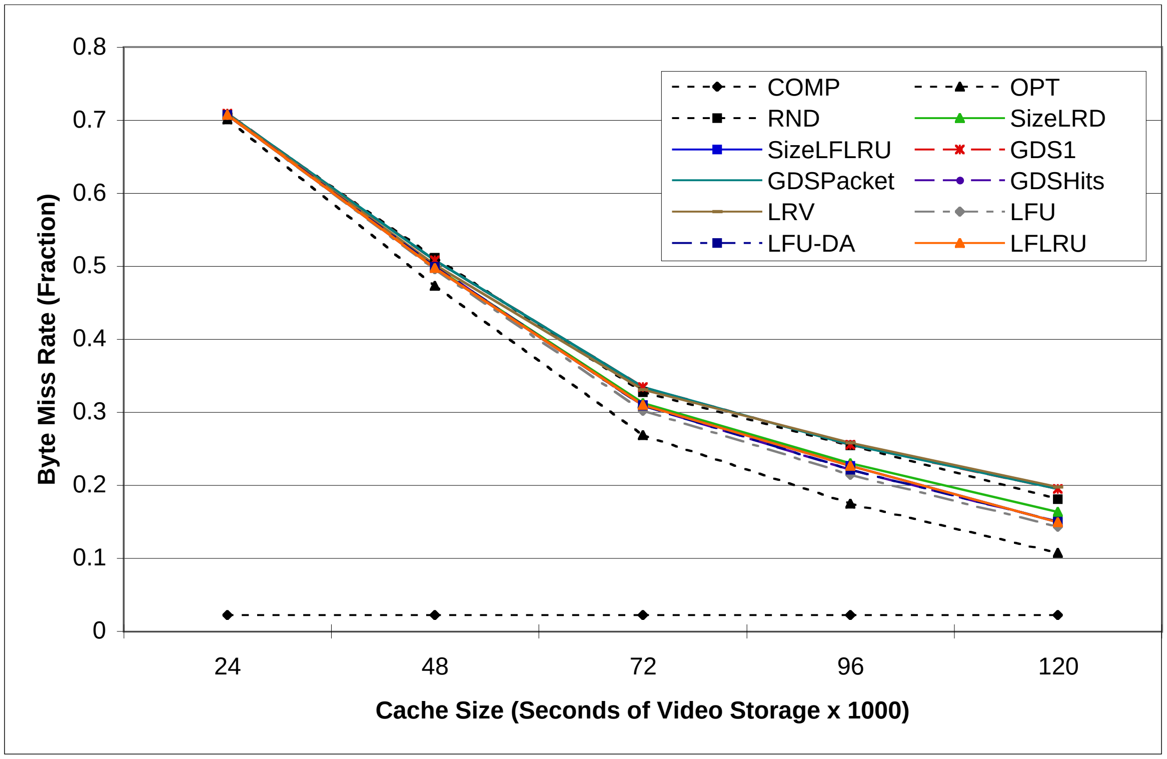


Fig. 6.93. Byte Miss Rates of Complex Cache Replacement Algorithms on OnCommand On-Demand Movie Server daysum1 Trace.

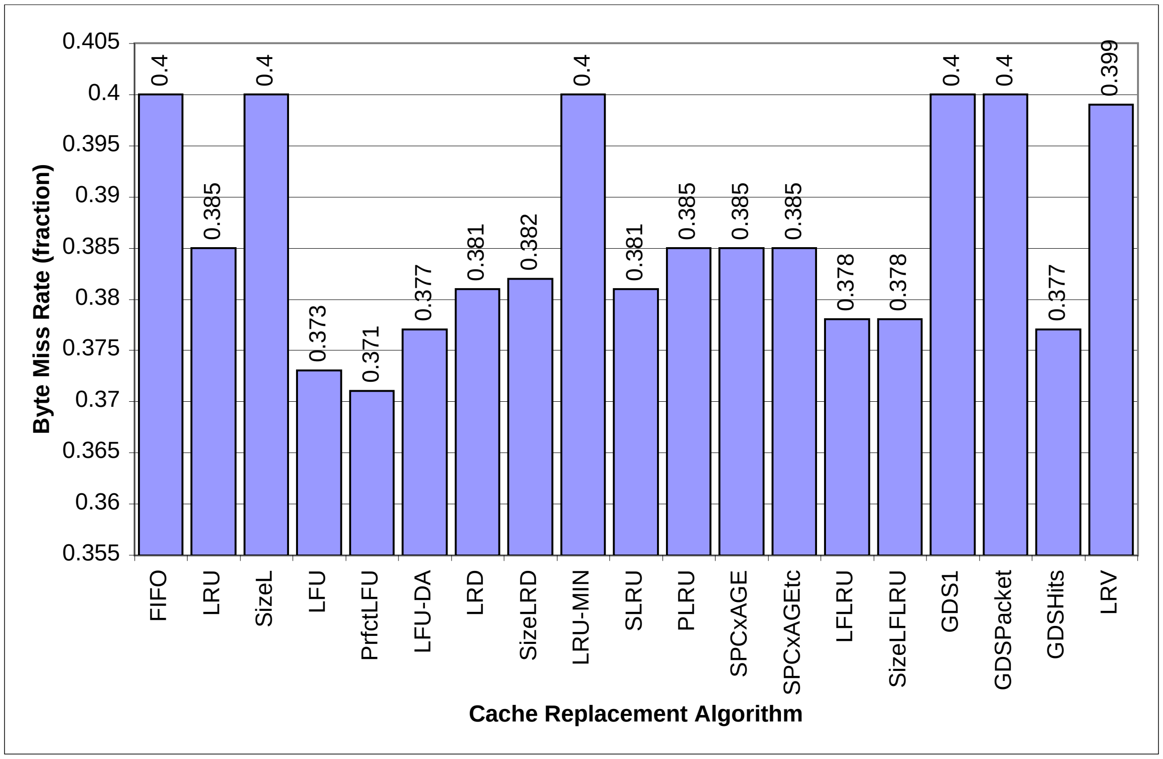


Fig. 6.94. Average Byte Miss Rates on OnCommand On-Demand Movie Server daysum1 Trace.

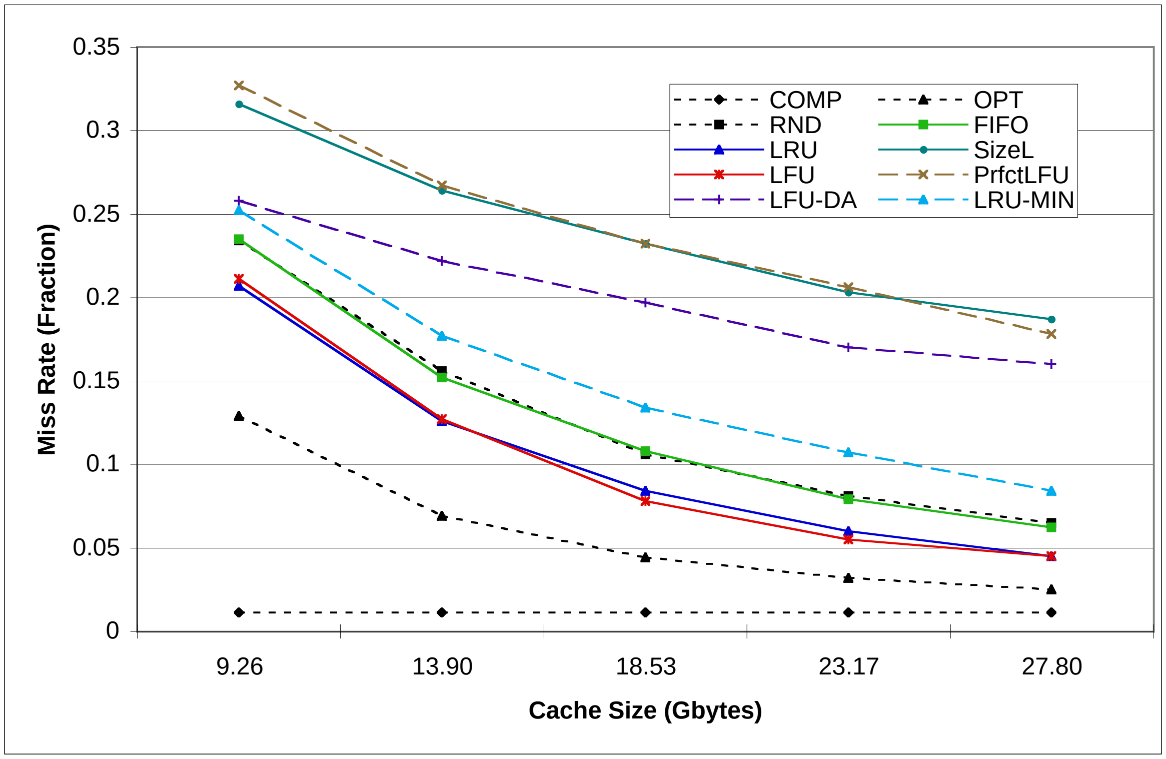


Fig. 6.95. Miss Rates of Simple Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

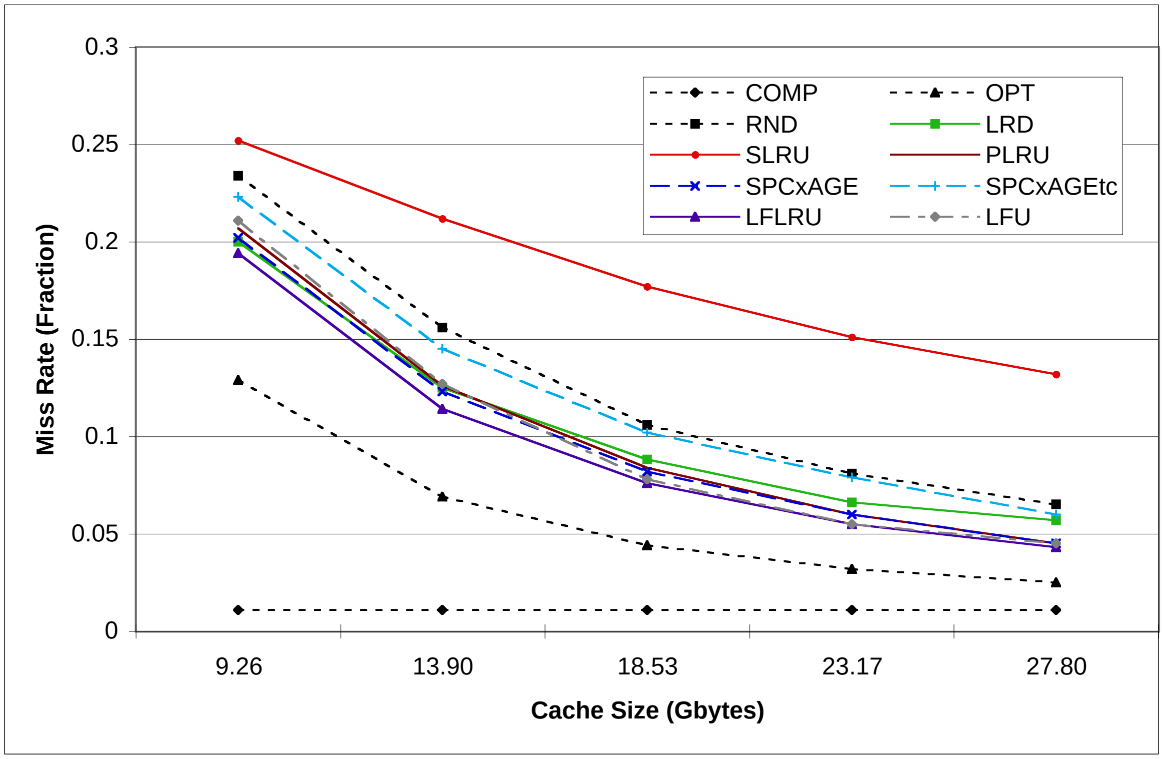


Fig. 6.96. Miss Rates of More Complex Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

SizeLFLRU, LFLRU, and SPACExAGE placed first, second, and third, respectively, for the DVJ2 system with miss rate as the measure as can be seen in Figure 6.97.

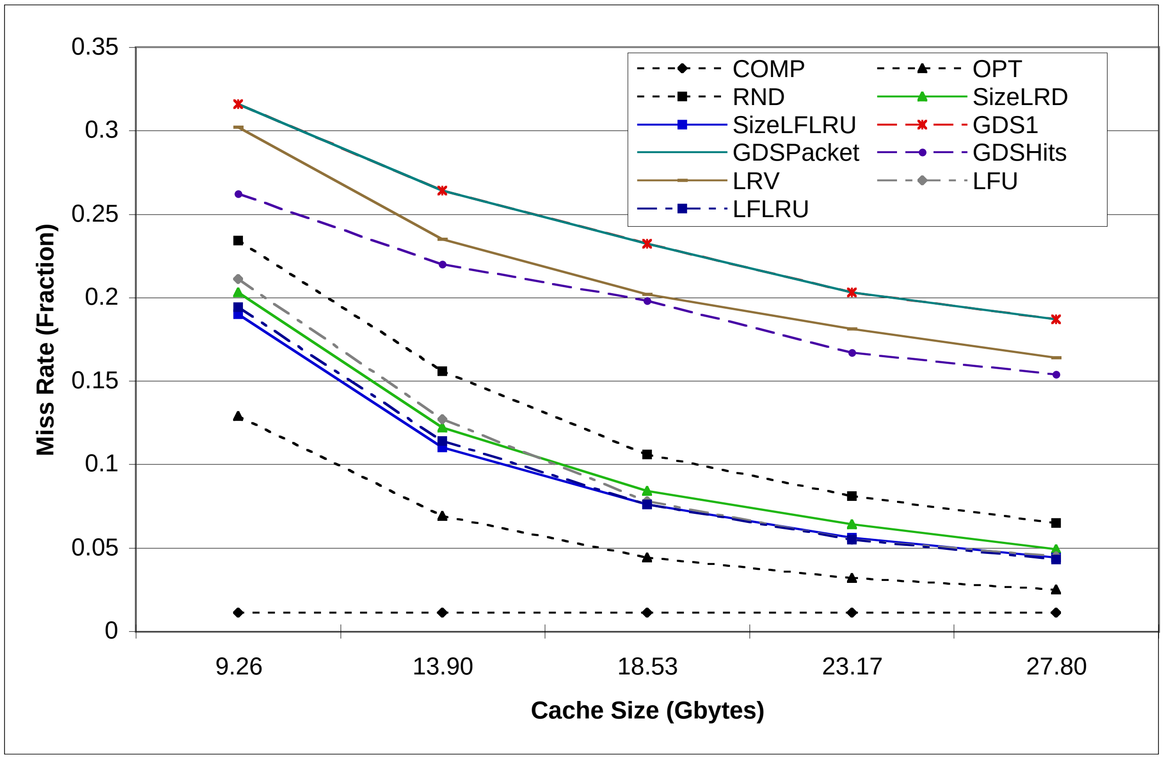


Fig. 6.97. Miss Rates of Complex Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

Unlike the OnCommand video objects, the DVJ2 video objects have files of different sizes. There is much less variation among the sizes of the files, though, than among the Web workload traces; generally, there is only about one order of magnitude of difference between the largest and smallest video object. Nevertheless, there is still some difference between the miss rate results and the byte miss rate results. Among the simple cache replacement algorithms in Figure 6.99, LFU is the best performing algorithm edging out LRU. The LFU results are then forwarded to Figure 6.100.

In Figure 6.100 for the more complex cache replacement algorithms, LFLRU and and LFU perform the best so their results are forwarded to Figure 6.101.

In Figures 6.101 and 6.102, LFLRU performs best, and its performance is virtually

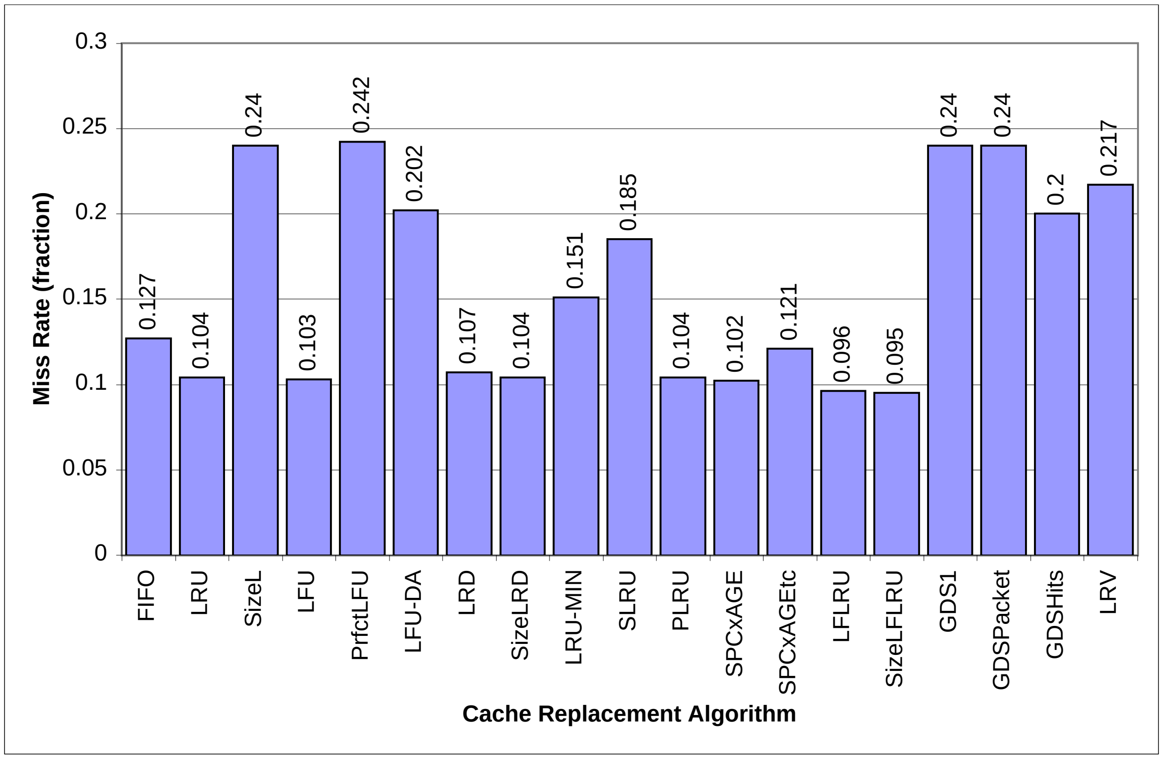


Fig. 6.98. Average Miss Rates on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

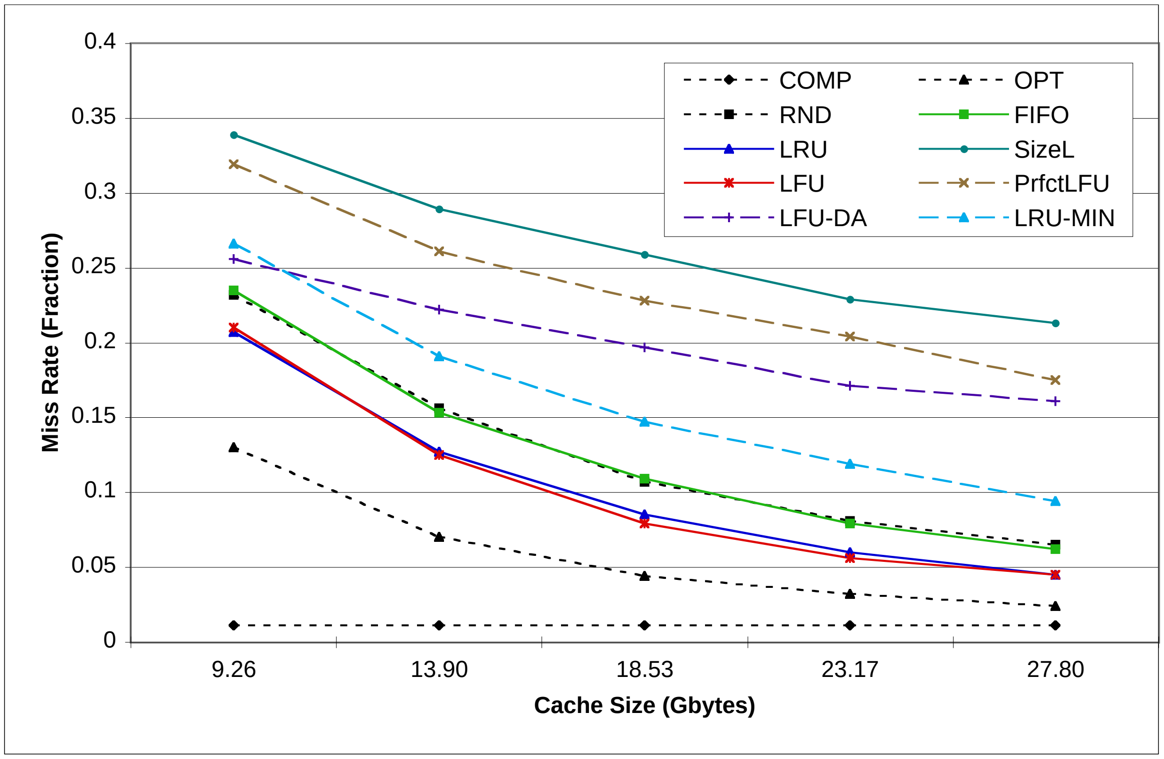


Fig. 6.99. Byte Miss Rates of Simple Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

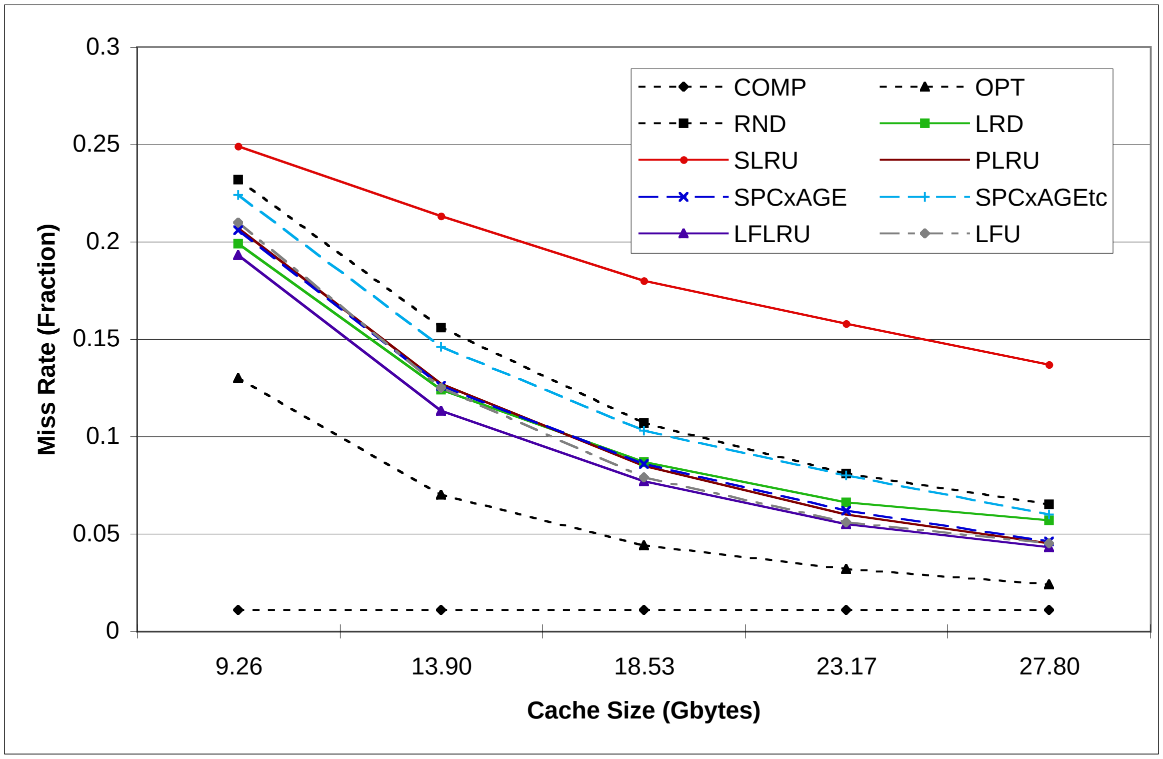


Fig. 6.100. Byte Miss Rates of More Complex Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

indistinguishable from SizeLFLRU which performs second best. Third place goes to the simple LFU algorithm.

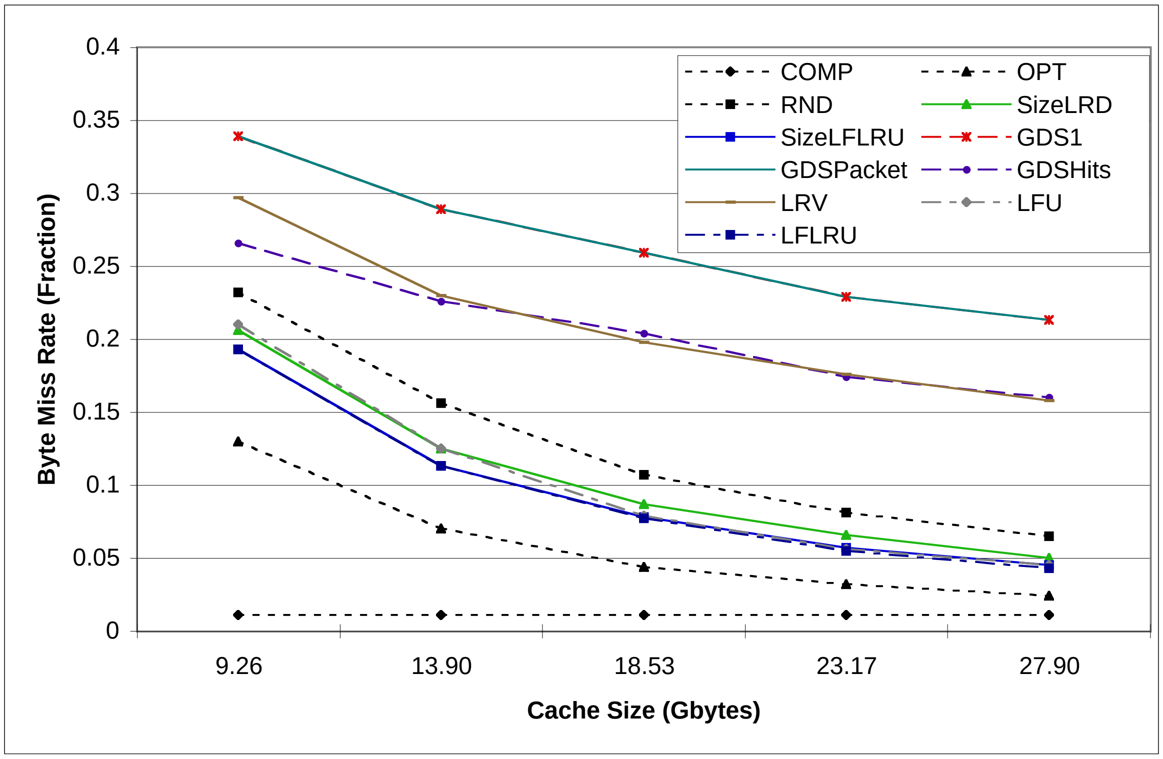


Fig. 6.101. Byte Miss Rates of Complex Cache Replacement Algorithms on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

Video Proxy Server Traces Summary

For this domain's summary, the table of top performing algorithms for each trace and both measures is presented in Table 6.5. Though both systems deliver digital video objects, they are used in very different ways and have different results with respect to cache replacement algorithms. Therefore, summaries will be drawn for each system individually.

Table 6.5 shows that LFU, LFU-DA, and GDSHits performed best for the OnCommand system for both miss rate and byte miss rate results. Looking at the average

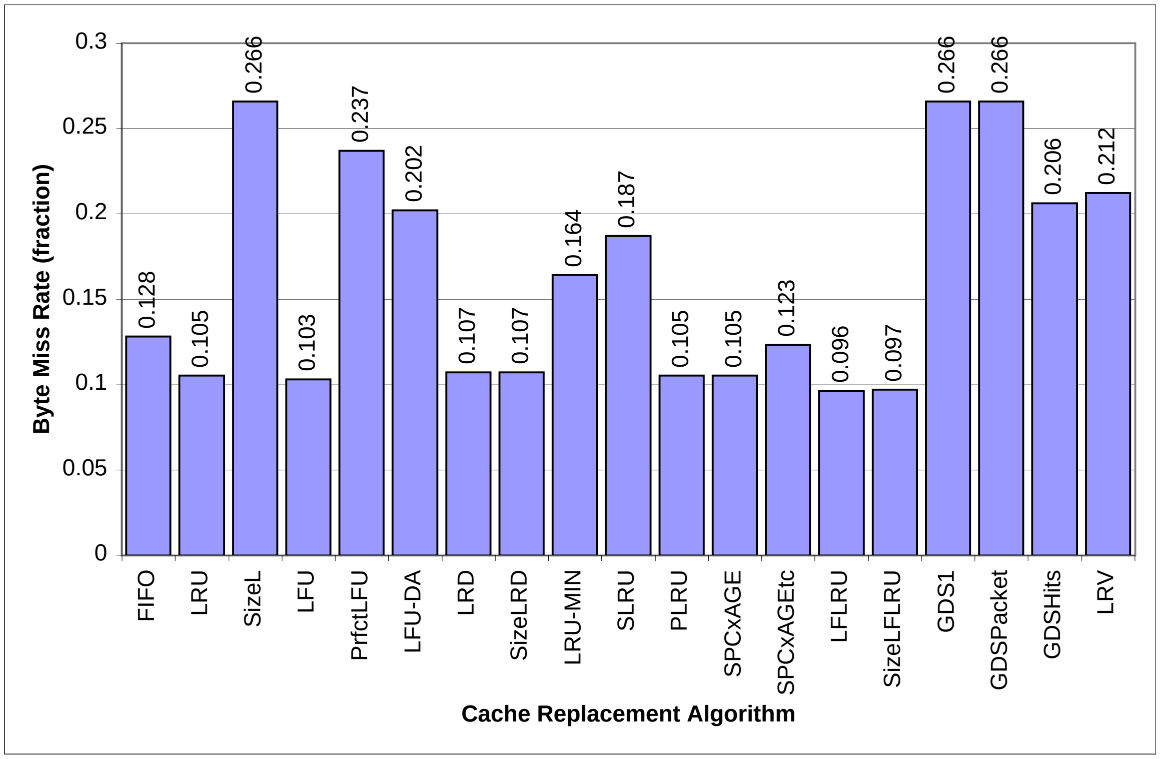


Fig. 6.102. Average Byte Miss Rates on DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

Table 6.5
Summary of Top Cache Replacement Algorithms for Video Proxy Server Traces

Trace	OnCommand	DVJ2
Miss	LFU (1st)	SizeLFLRU (1st)
Rate	LFU-DA (2nd-tie)	LFLRU (2nd)
	GDSHits (2nd-tie)	SPACExAGE (3rd)
Byte	LFU (1st)	LFLRU (1st)
Miss	LFU-DA (2nd-tie)	SizeLFLRU (2nd)
Rate	GDSHits (2nd-tie)	LFU (3rd)

miss rate graph in Figure 6.90, LFU, LFU-DA, LFLRU, SizeLFLRU, and GDSHits all perform within 0.5% of each other as the top algorithms for the OnCommand workload trace. All of these have access count as a component in the computation of the cache replacement algorithm. It should be noted that these access count algorithms are better than LRU by as much as 1.2%. This finding strongly suggests that the access count statistic would be very important in the cache replacement algorithm of an OnCommand proxy caching server. In contrast with the access count advantage, the algorithms that rely more heavily on the size of the movies like SizeL, LRU-MIN, GDS1, and GDSPacket do not perform as well. They have up to 2.5% higher miss rate on average than the other algorithms. This can be attributed to the fact that each of the videos is assumed to be of the same size so the size can provide absolutely no input as to whether it will be accessed again. This is reinforced by SPACExAGE and SPACExAGEtc which perform exactly the same as LRU. Since there is only one file type, PLRU is of no advantage over LRU. Finally, SLRU only performs marginally better than LRU. SLRU counts on a significant percentage of objects being accessed only once which is not the case according to Table 5.105.

Table 6.5 shows that SizeLFLRU, LFLRU, and SPACExAGE perform the best for the DVJ2 system with miss rate as the measure while LFLRU, SizeLFLRU, and LFU perform best when the measure is byte miss rate. As mentioned above, the sizes of the video objects are within one order of magnitude of each other so the miss rate results and byte miss rate results are very similar. When looking at the average miss rate graph in Figure 6.98 and the average byte miss rate graph in Figure 6.102, several other algorithms also perform well such as LFU, LRU, LRD, and SizeLRD. Each of these algorithms use time differences and/or access counts as cache replacement determination statistics. So one of these other algorithms could be used without incurring horribly worse miss rates. (PLRU could be considered being part of this group but since there is only one file type, PLRU defaults to performing equivalently to LRU and is therefore not given the same attention as the above algorithms.) However, the same figure shows that choosing one of the inflation parameter algorithms (LFU-DA, GDS1, GDSPacket, and GDSHits), SLRU, or LRV could produce many more cache misses that could be avoided by using a more efficient algorithm. Since the students access the videos in a fairly regular, linear pattern as the semester progresses [RM97], perhaps the inflation parameter algorithms do not perform as well because the inflation parameter is not able to decay the objects' cache replacement scores fast enough to evict the objects soon enough. SLRU may not perform well because the objects are accessed so many times; it is not efficient to hold the objects in cache on a probationary basis. SizeL does not perform so well because the sizes are all similar and are not a good basis for evicting videos. Finally, PerfectLFU performs poorly because the videos are only popular for a certain period in the semester and should be evicted soon thereafter. PerfectLFU holds on to some objects far too long after they have been popular in the semester.

6.1.5 Stochastically Generated Video Server Traces

Truncated Discrete Exponential Distribution Model Traces

Using the Truncated Discrete Exponential Distribution Model trace set among the simple cache replacement algorithms, the LFU algorithm performs best as shown in Figure 6.103. The results from this algorithm are forwarded to Figure 6.104.

Among the more complex cache replacement algorithms in Figure 6.104, LRD and LFLRU perform the best, and their results are forwarded to Figure 6.105.

From Figures 6.105 and 6.106, the top performing algorithms for the Truncated Discrete Exponential Distribution Model trace set are SizeLFLRU (placing first), LFLRU (placing second) and LRD (placing third).

Examining the byte miss rate results, Figure 6.107 shows that LFU performs best on average among the simple cache replacement algorithms, and the LFU results are forwarded to Figure 6.108.

LFLRU and LRD are the top performing algorithms in Figure 6.108, and their results are forwarded to Figure 6.109.

Similar to the miss rate results for the Truncated Discrete Exponential Distribution Model trace set, the top performing algorithm was LFLRU, with SizeLFLRU

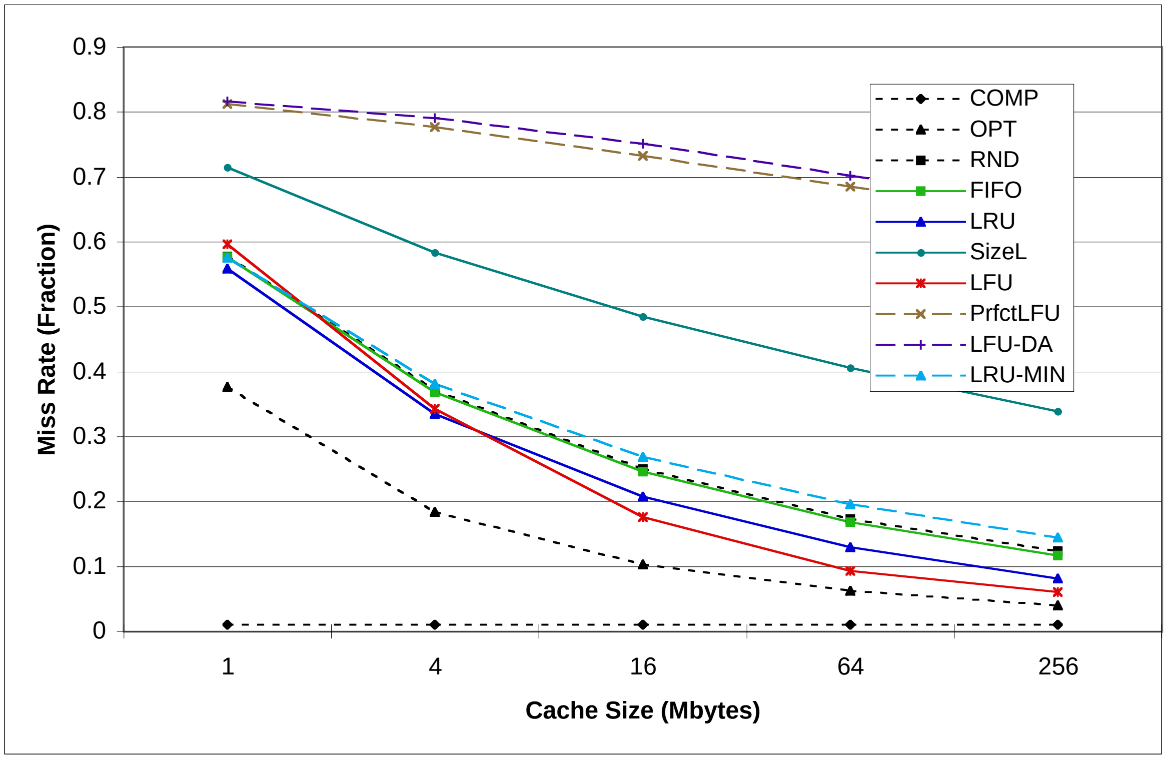


Fig. 6.103. Miss Rates of Simple Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

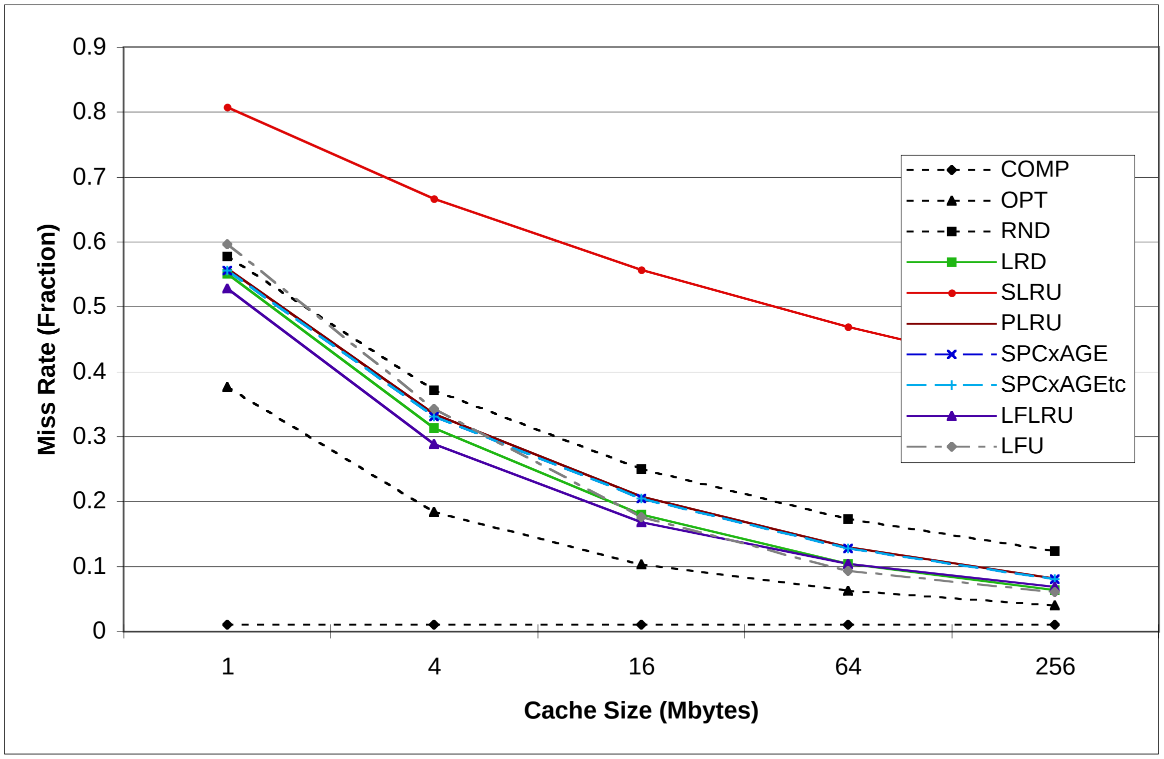


Fig. 6.104. Miss Rates of More Complex Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

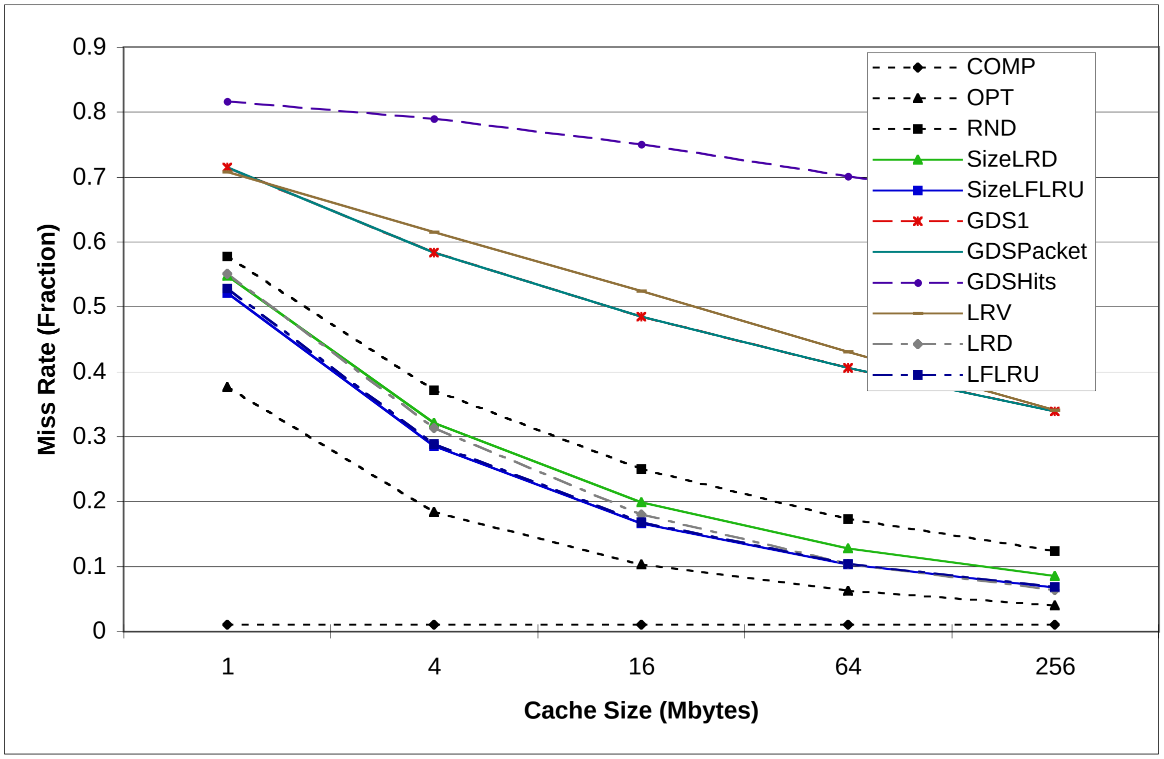


Fig. 6.105. Miss Rates of Complex Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

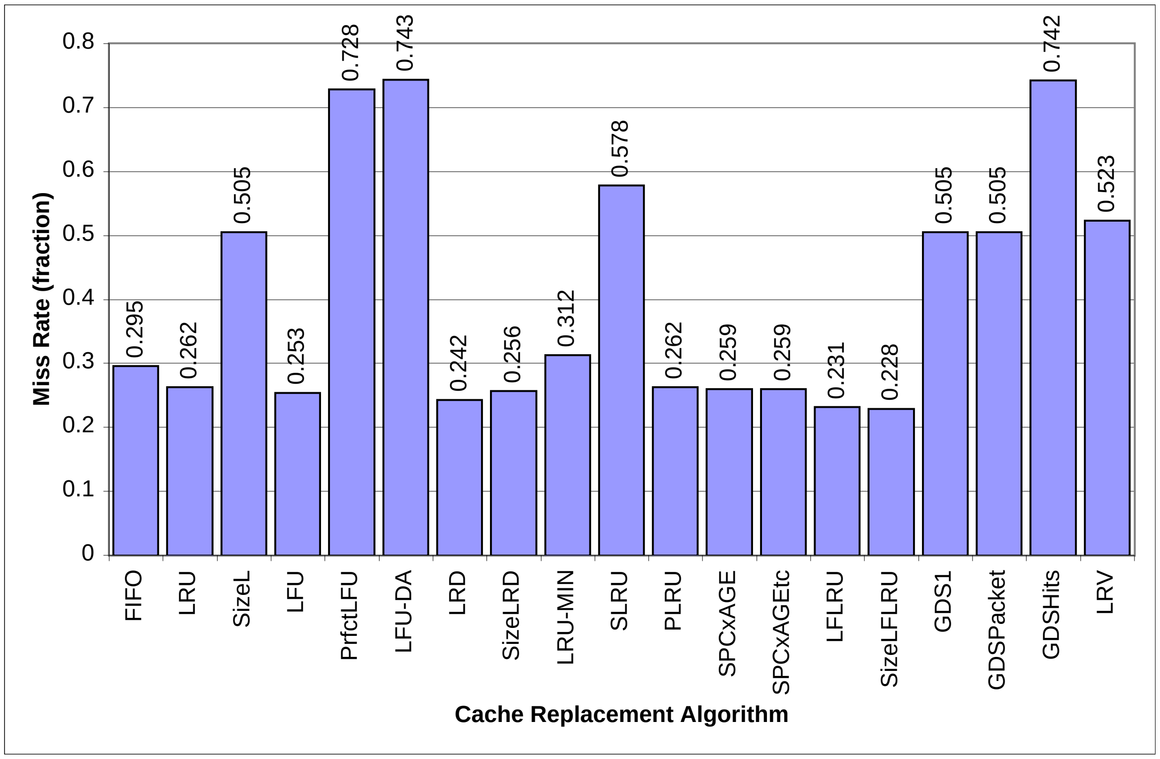


Fig. 6.106. Average Miss Rates on Truncated Discrete Exponential Distribution Model Traces.

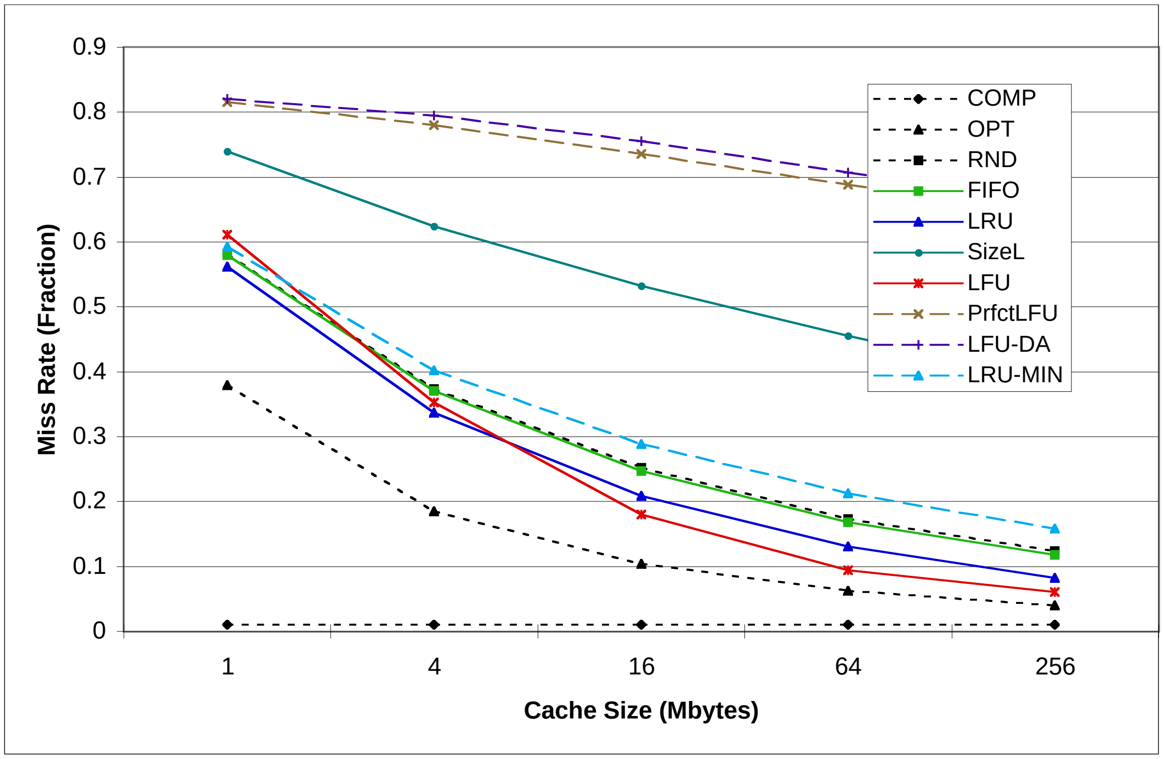


Fig. 6.107. Byte Miss Rates of Simple Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

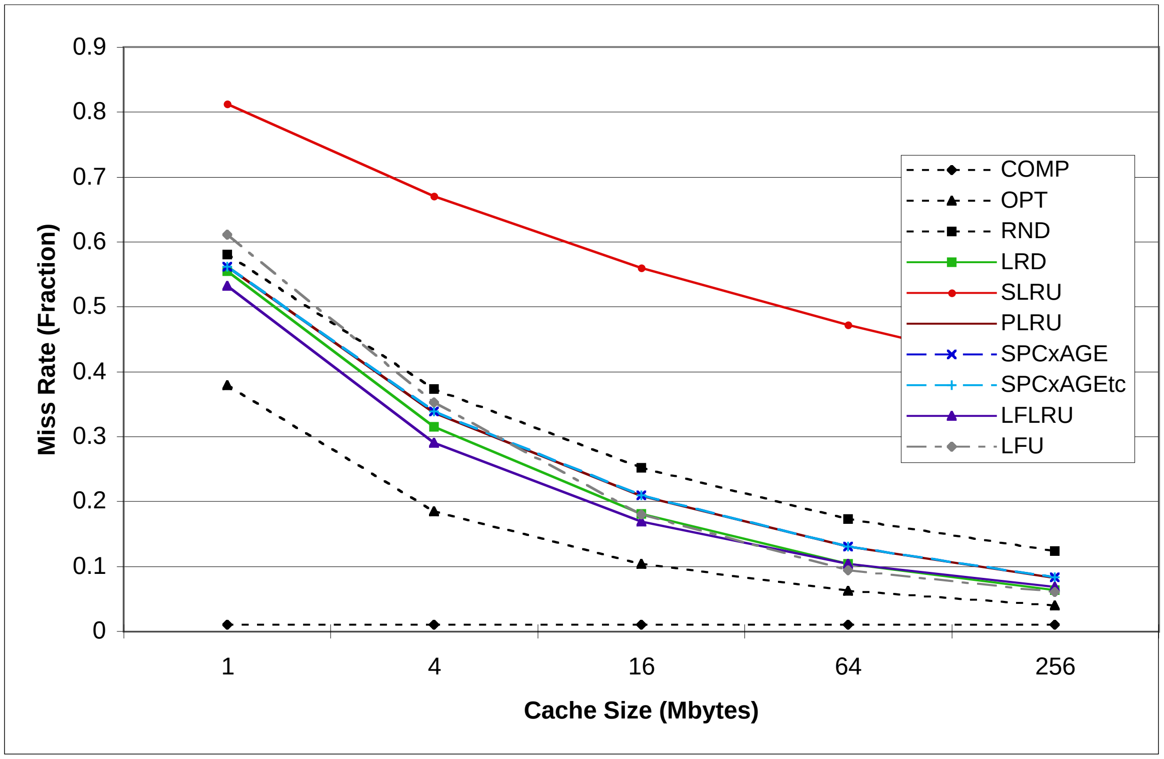


Fig. 6.108. Byte Miss Rates of More Complex Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

placing second and LRD placing third. These results can be see in Figures 6.109 and 6.110.

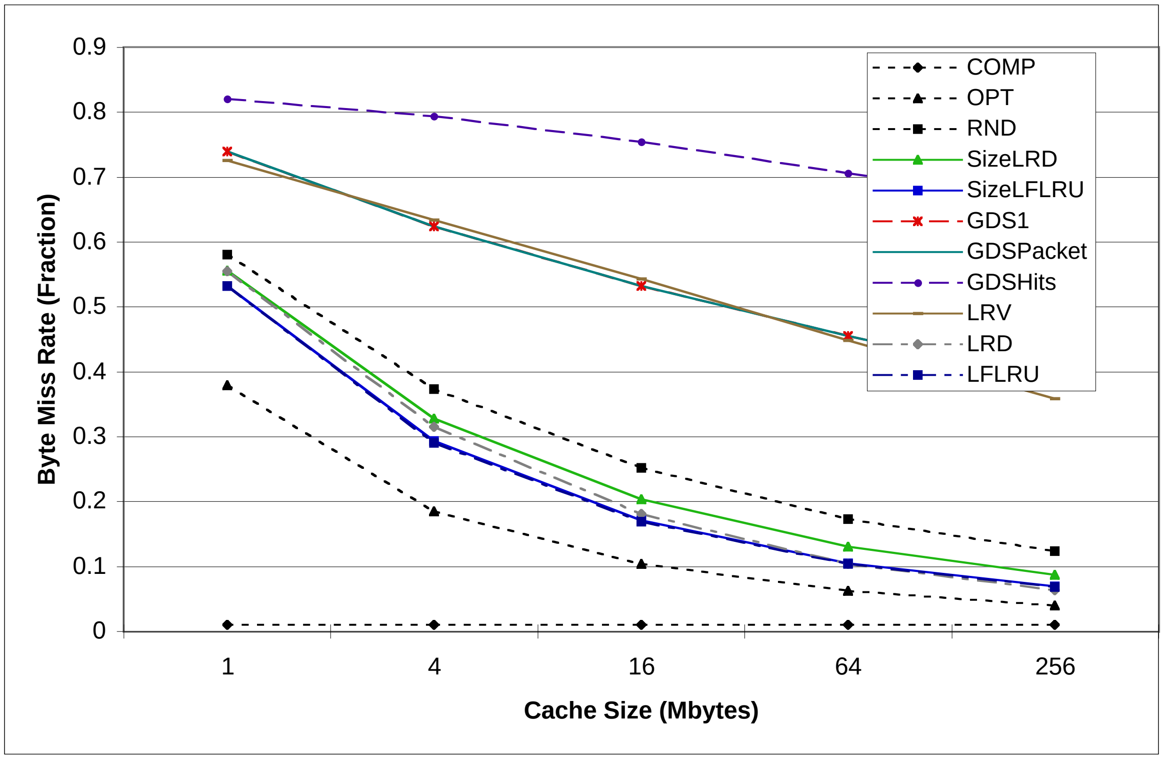


Fig. 6.109. Byte Miss Rates of Complex Cache Replacement Algorithms on Truncated Discrete Exponential Distribution Model Traces.

Binomial Distribution Model Traces

For the simple cache replacement algorithms on the Binomial Distribution Model trace set in Figure 6.111, the LRU algorithm performs best, and its results are forwarded to Figure 6.112.

In Figure 6.112, the top performing algorithms are SPACExAGE and LFLRU so their results are forwarded to Figure 6.113.

It is difficult to determine the top performing algorithms for miss rate from Figure 6.113 alone, but Figure 6.114 helps clarify that SizeLFLRU performs best followed

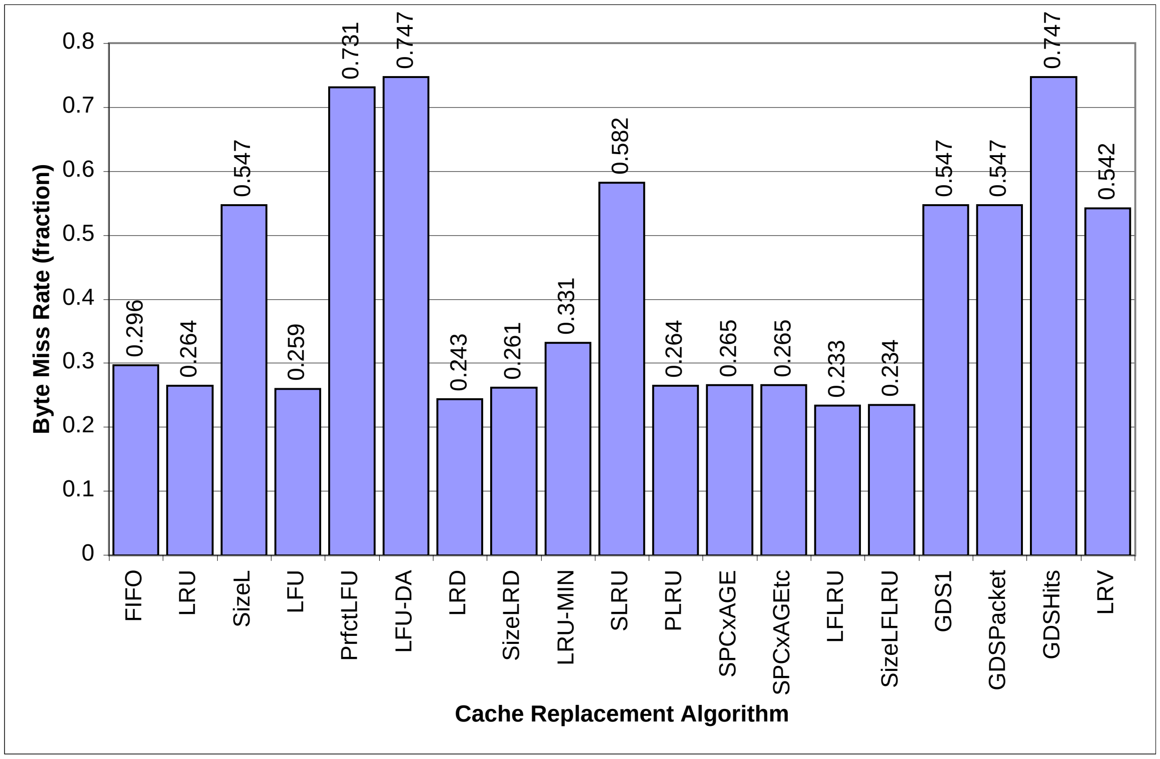


Fig. 6.110. Average Byte Miss Rates on Truncated Discrete Exponential Distribution Model Traces.

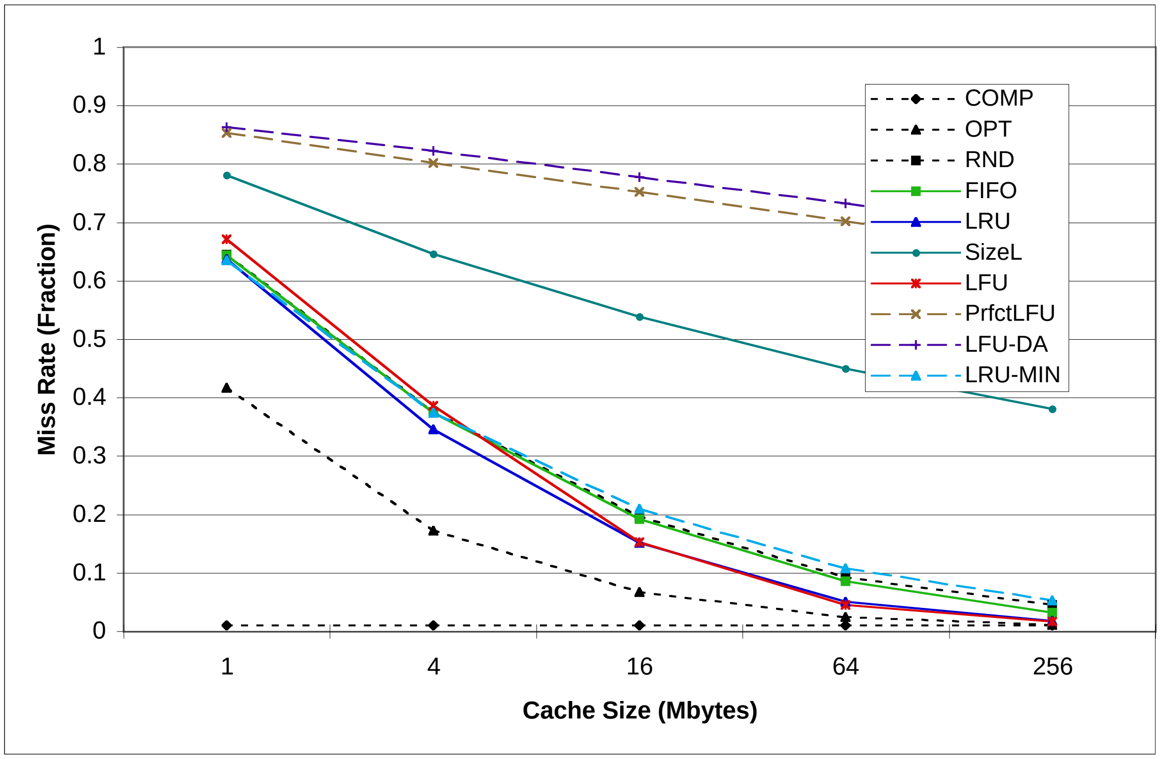


Fig. 6.111. Miss Rates of Simple Cache Replacement Algorithms on Binomial Distribution Model Traces.

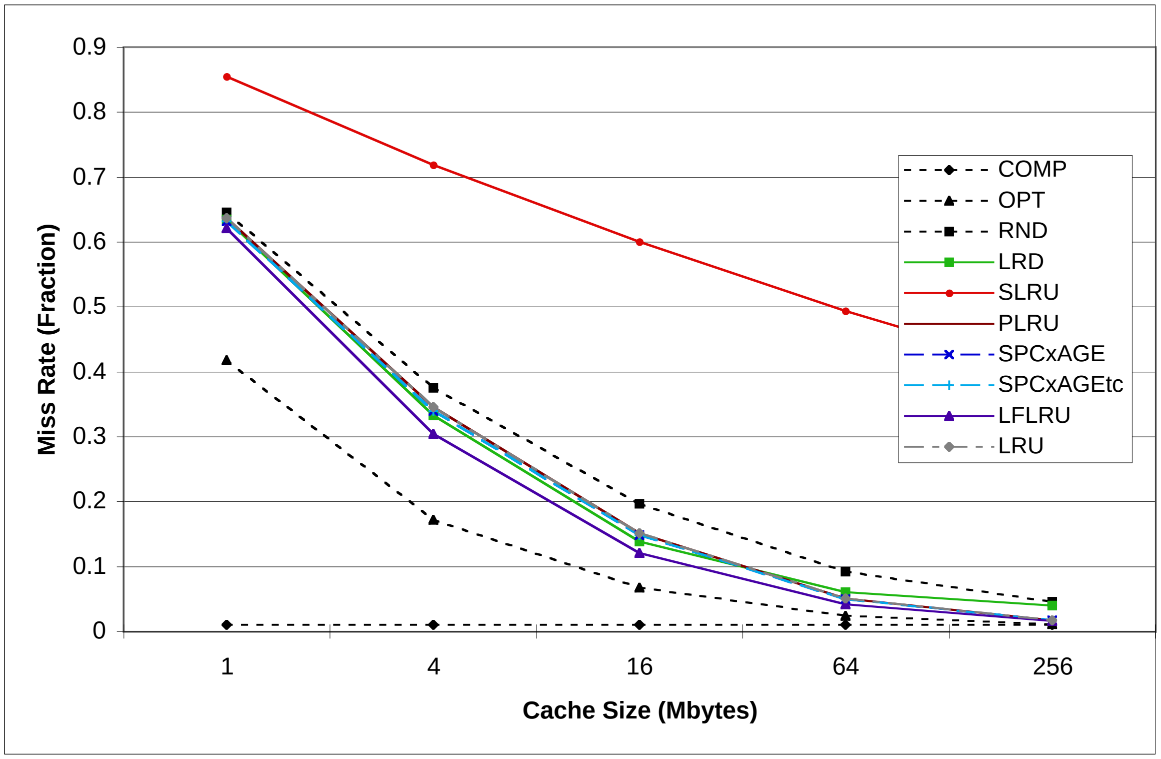


Fig. 6.112. Miss Rates of More Complex Cache Replacement Algorithms on Binomial Distribution Model Traces.

by LFLRU in second place and SPACExAGE and SPACExAGEtc tied for third.

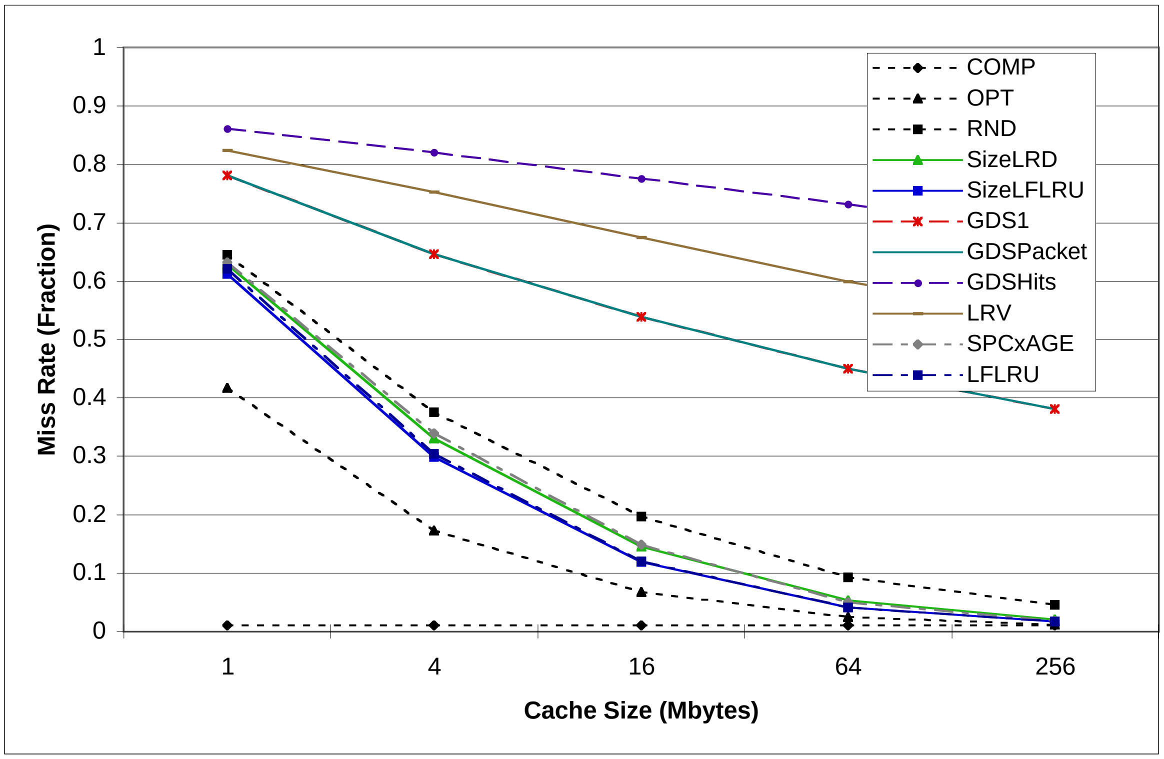


Fig. 6.113. Miss Rates of Complex Cache Replacement Algorithms on Binomial Distribution Model Traces.

For the byte miss rate measure on the Binomial Distribution Model trace sets, Figure 6.115 shows that LRU performs best among the simple cache replacement algorithm, especially among the smaller cache sizes. Its results are forwarded to Figure 6.116.

The SPACExAGE and LFLRU algorithms are the top performing algorithms whose results are depicted in Figure 6.116. Their results are forwarded to Figure 6.117.

The LFLRU algorithm performed best among all of the algorithms as evidenced in Figures 6.117 and 6.118. It is followed closely (by 0.1% on average) by SizeLFLRU. Then there is a four way tie for third for average byte miss rate between LRU, SPACExAGE, SPACExAGEtc, and PLRU.

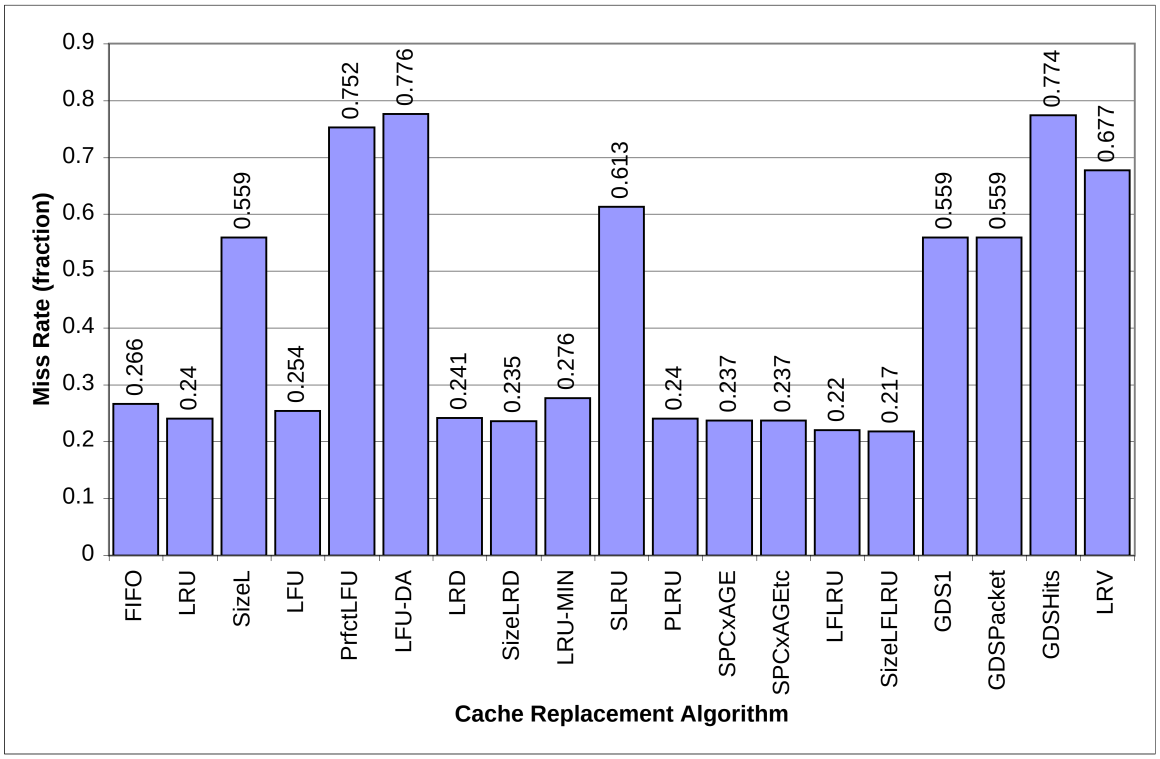


Fig. 6.114. Average Miss Rates on Binomial Distribution Model Traces.

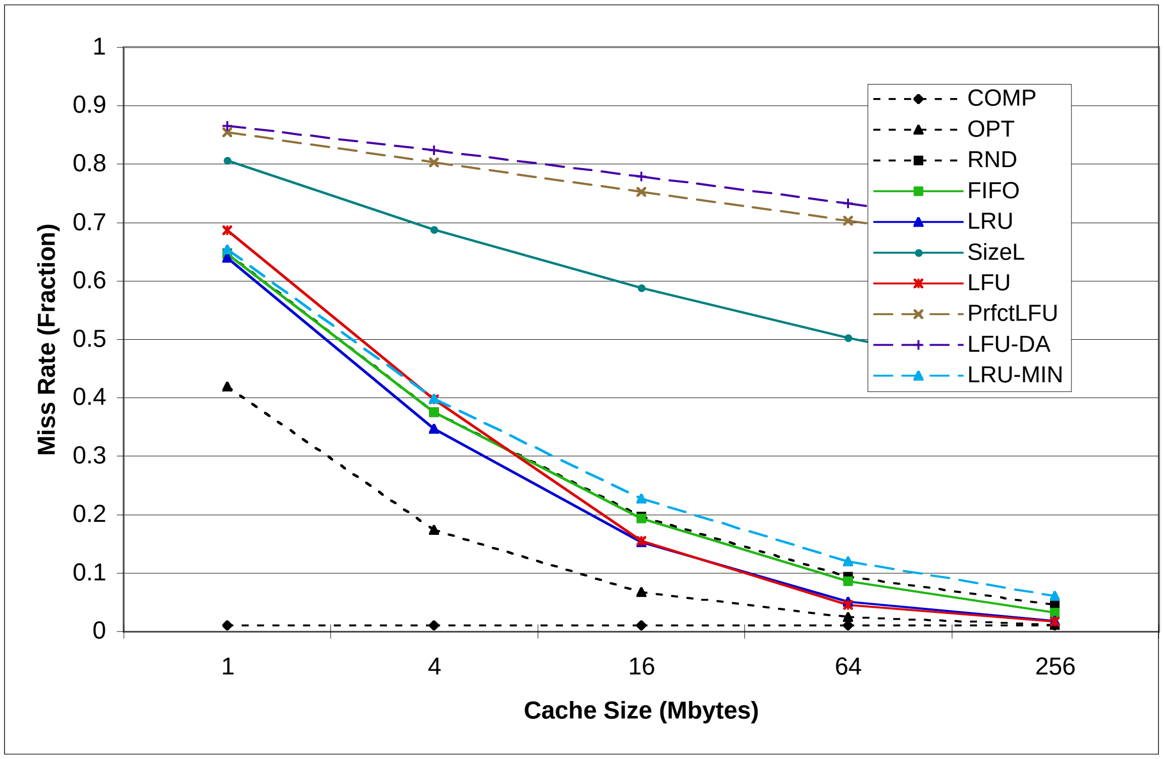


Fig. 6.115. Byte Miss Rates of Simple Cache Replacement Algorithms on Binomial Distribution Model Traces.

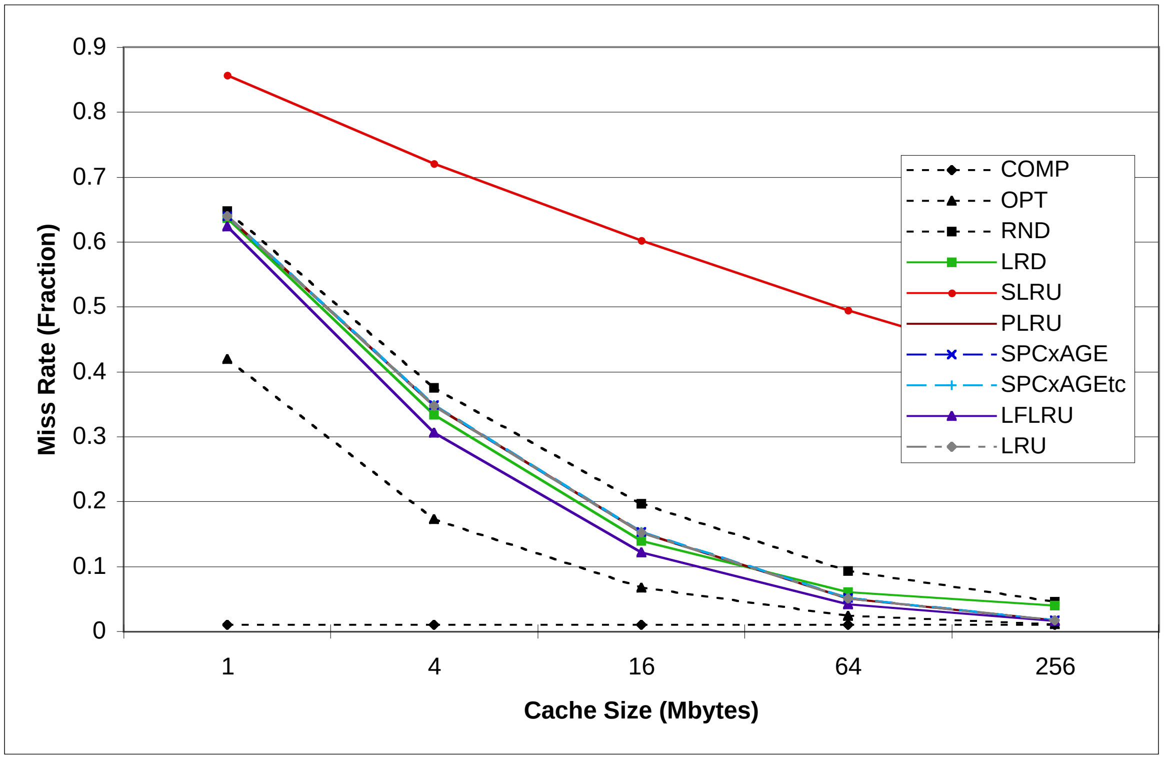


Fig. 6.116. Byte Miss Rates of More Complex Cache Replacement Algorithms on Binomial Distribution Model Traces.

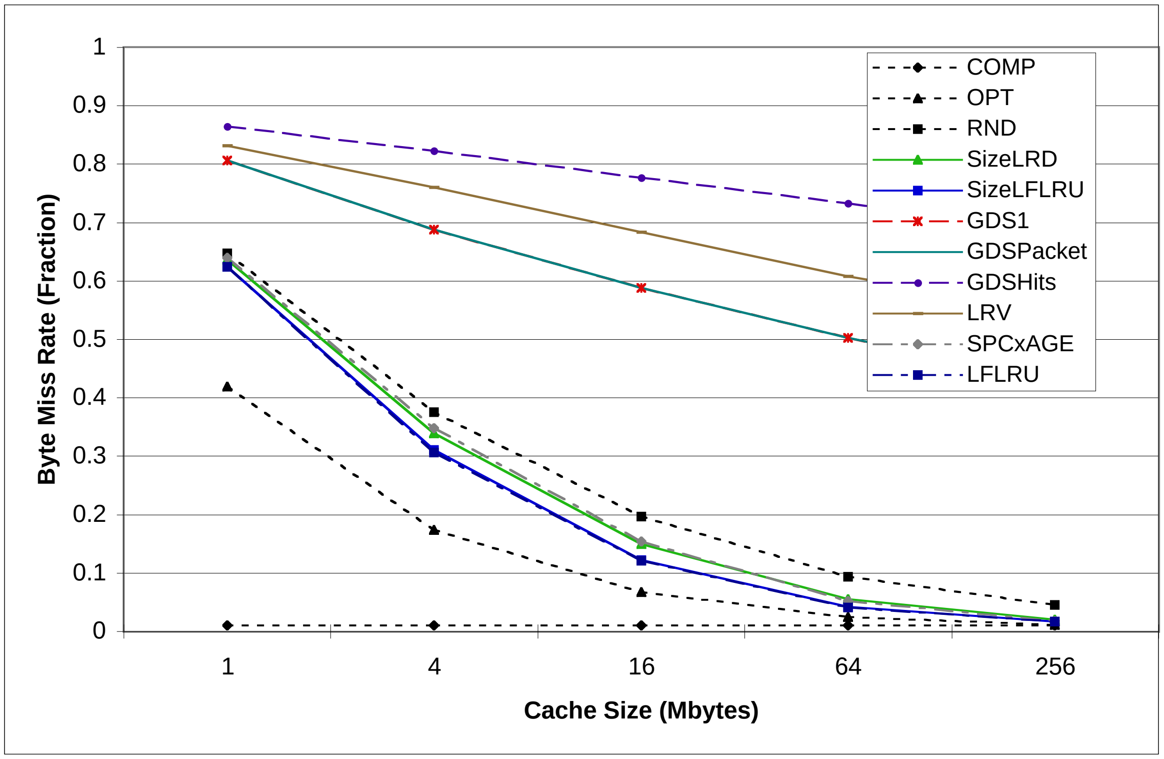


Fig. 6.117. Byte Miss Rates of Complex Cache Replacement Algorithms on Binomial Distribution Model Traces.

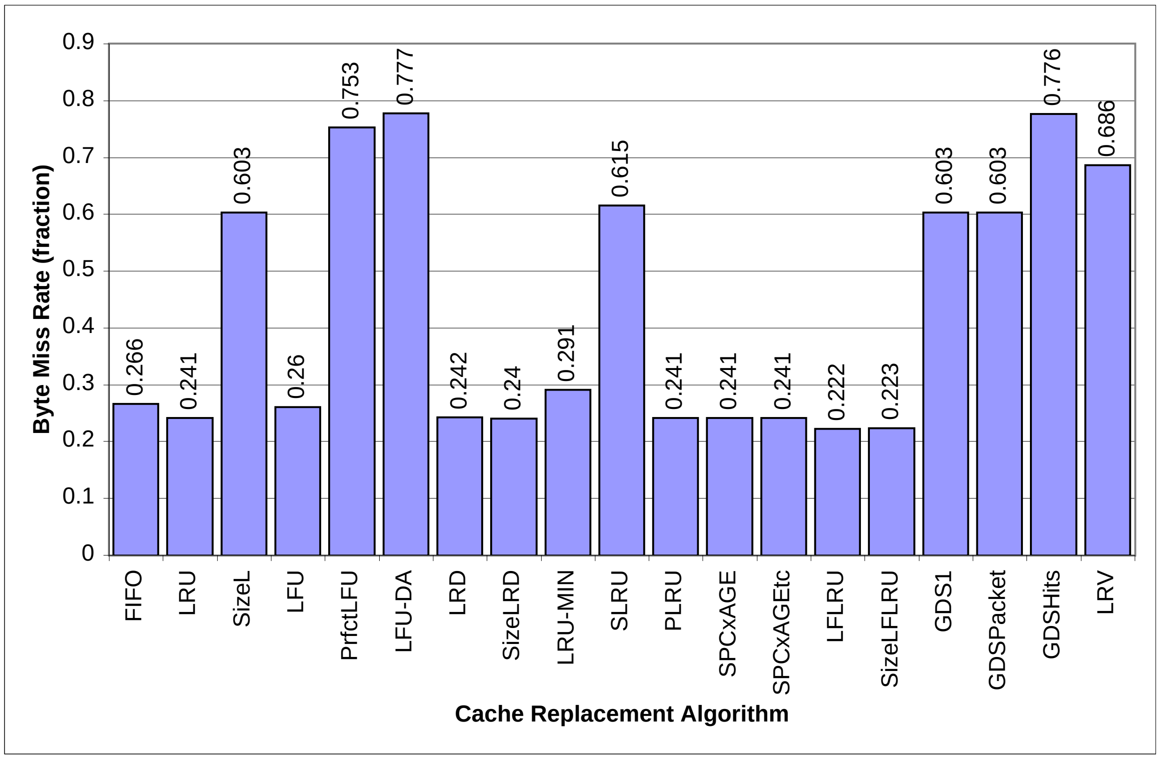


Fig. 6.118. Average Byte Miss Rates on Binomial Distribution Model Traces.

Triangular Window Distribution Model Traces

For the Triangular Window Distribution Model trace set, Figure 6.119 shows that the LFU algorithm performs best among the simple cache replacement algorithms. Its results are forwarded to Figure 6.120.

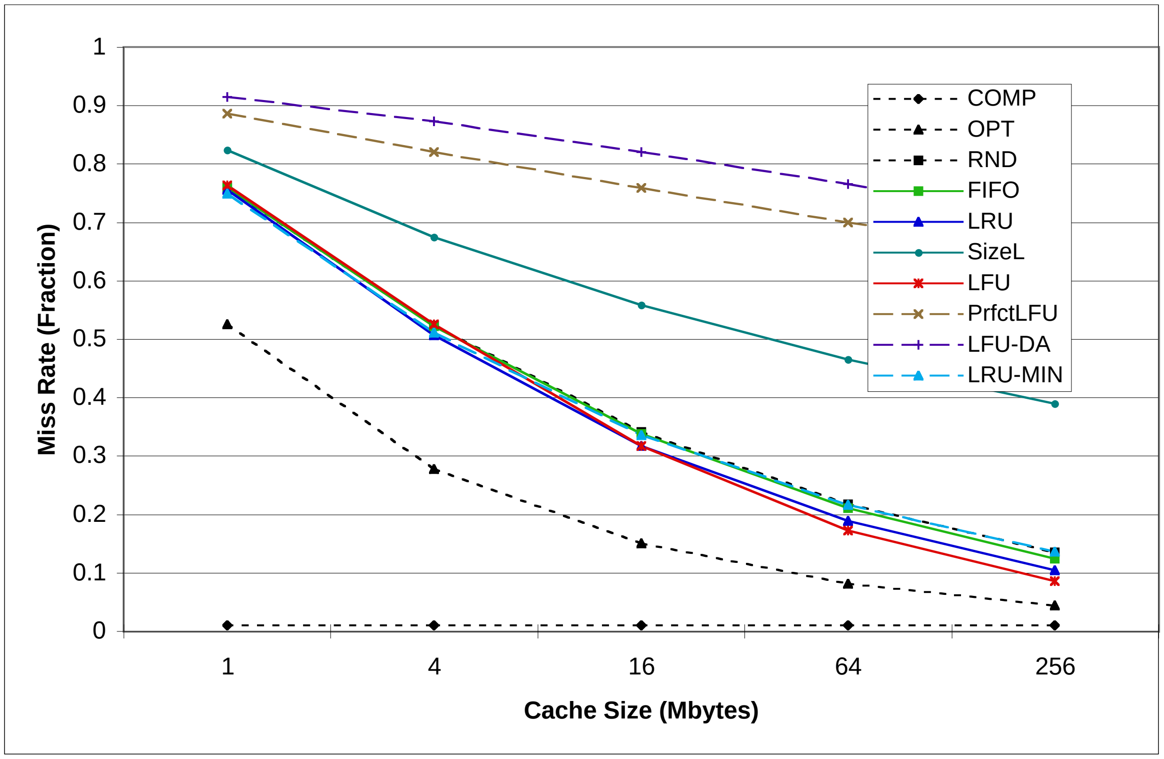


Fig. 6.119. Miss Rates of Simple Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

LRD and LFLRU are the top performing algorithms from Figure 6.120 so their results are forwarded to Figure 6.121.

Once again, it is difficult to determine which algorithms performed best from Figure 6.121, but Figure 6.122 helps clear up the congestion among the top algorithms. SizeLFLRU performed best, and LFLRU placed a close second with LRD and SizeLRD tying for third.

Using byte miss rate as the measure, Figure 6.123 shows that the LRU algorithm

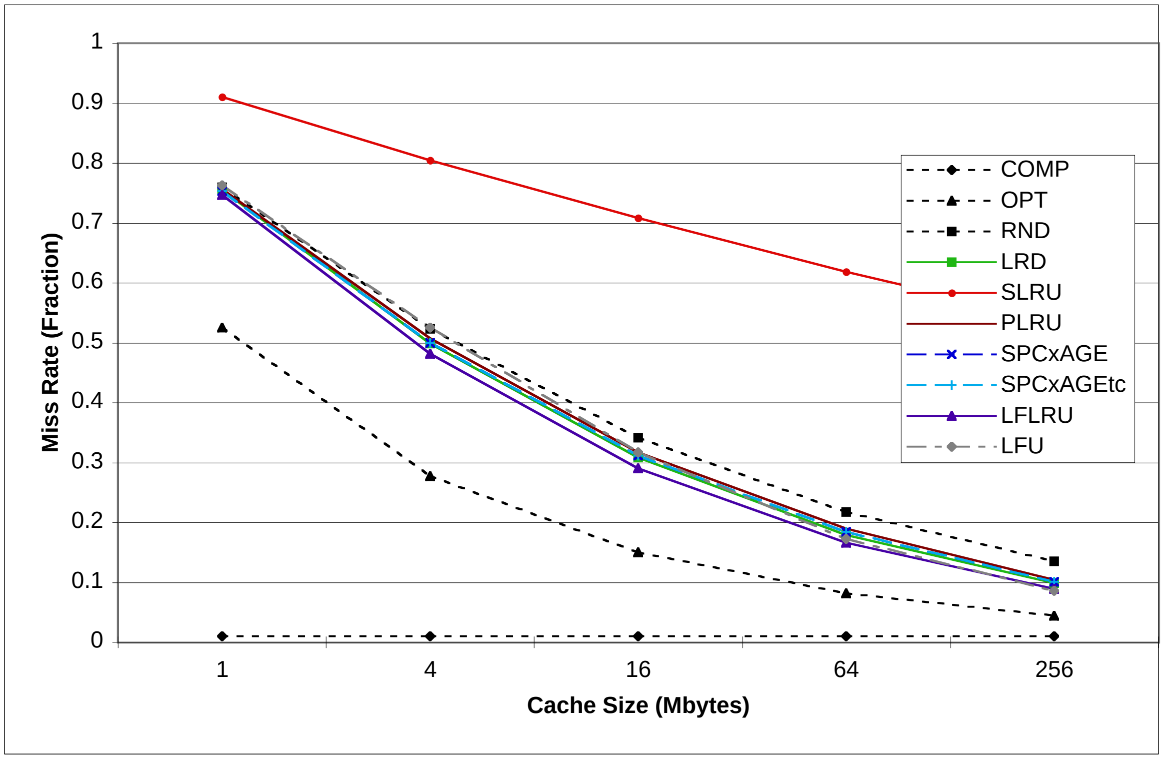


Fig. 6.120. Miss Rates of More Complex Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

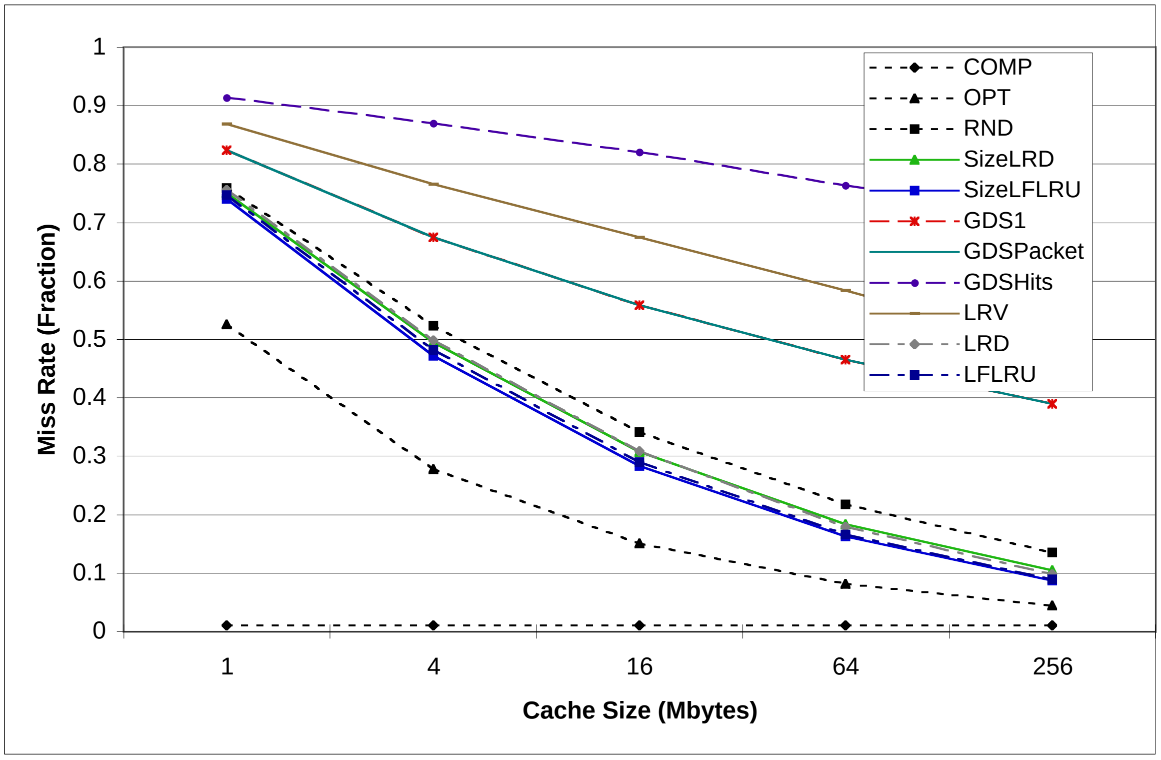


Fig. 6.121. Miss Rates of Complex Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

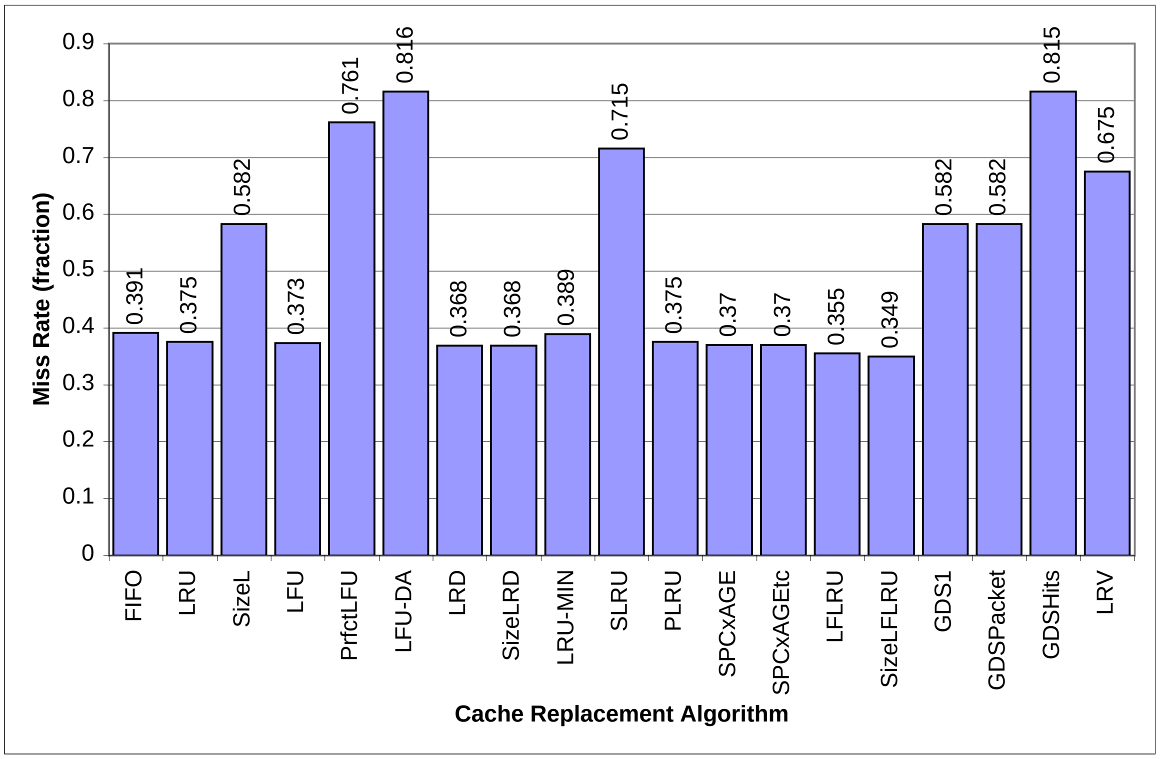


Fig. 6.122. Average Miss Rates on Triangular Window Distribution Model Traces.

edged out LFU for the top performing simple algorithm. The LRU results are forwarded to Figure 6.124.

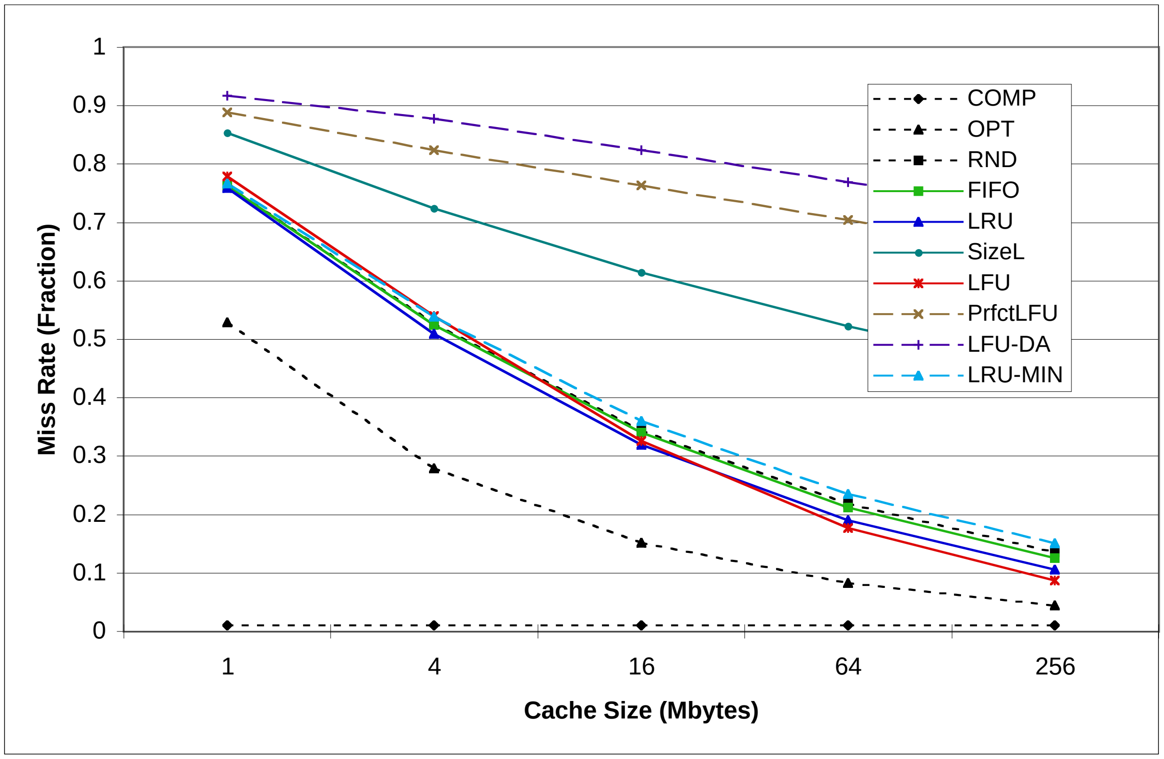


Fig. 6.123. Byte Miss Rates of Simple Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

In Figure 6.124, LRD and LFLRU performed best, and their results are forwarded to Figure 6.125.

Finally, the top algorithm for byte miss rate are LFLRU with SizeLFLRU and LRD taking second and third places respectively. These results are shown in Figures 6.125 and 6.126.

Stochastically Generated Video Server Traces Summary

For these stochastically generated video server trace sets, the table of top performing algorithms for each trace and both measures are presented in Table 6.6. Though each of these three trace sets are based on very different distributions, the

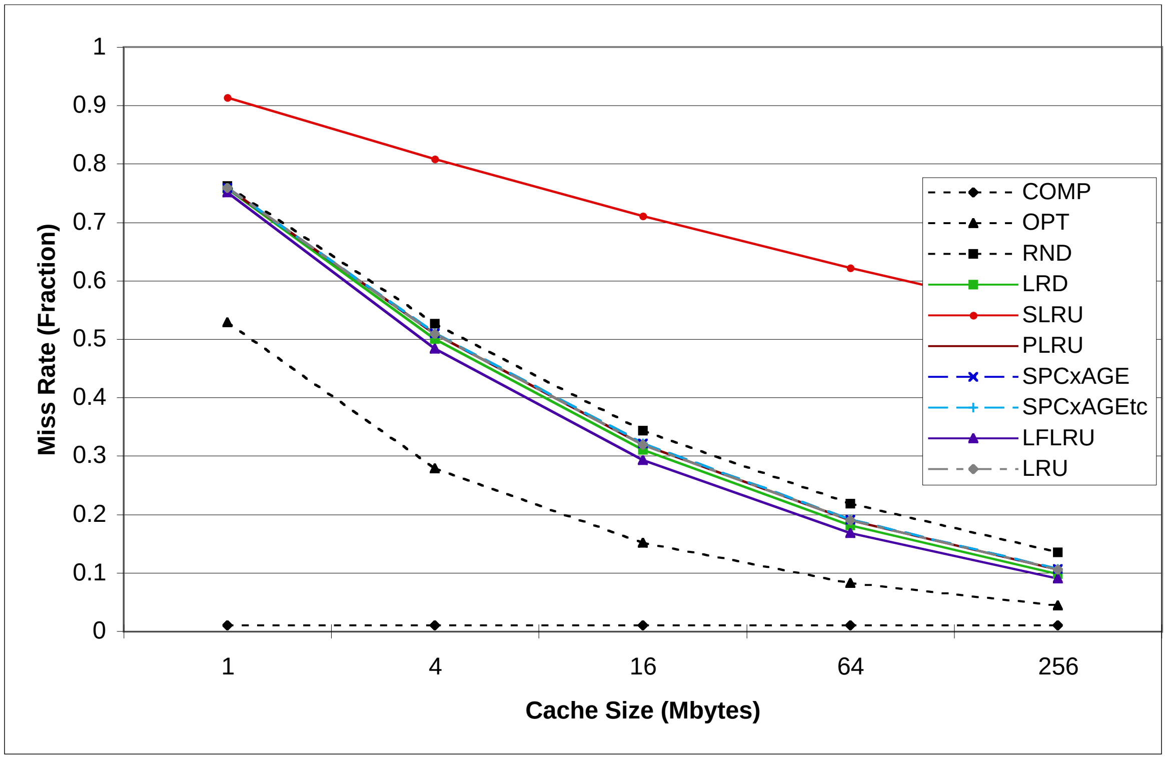


Fig. 6.124. Byte Miss Rates of More Complex Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

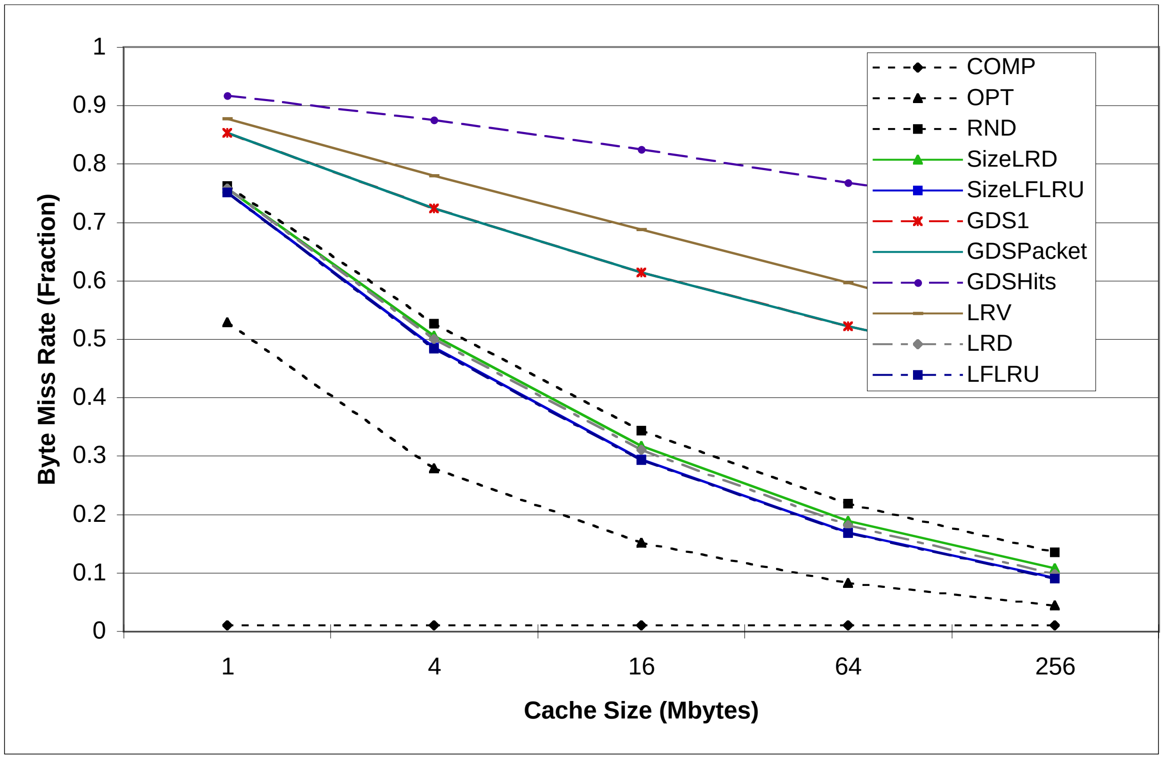


Fig. 6.125. Byte Miss Rates of Complex Cache Replacement Algorithms on Triangular Window Distribution Model Traces.

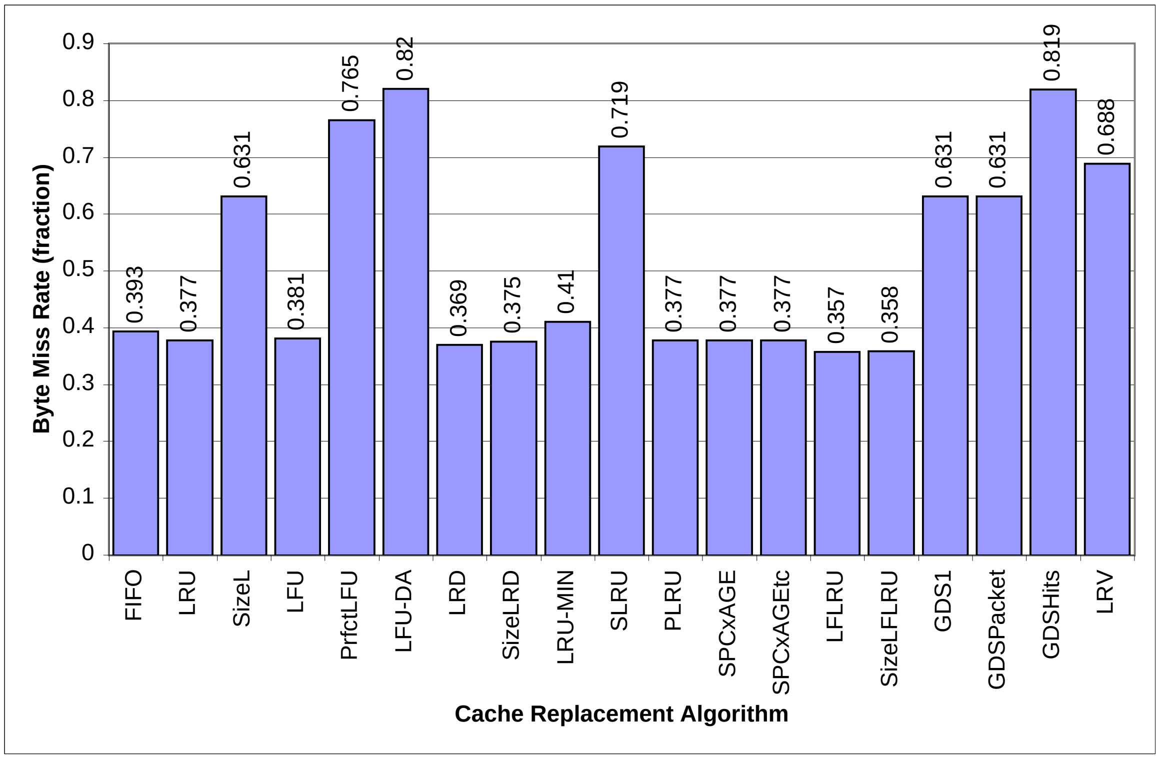


Fig. 6.126. Average Byte Miss Rates on Triangular Window Distribution Model Traces.

overall top performing cache replacement algorithms are very consistent. For miss rates, SizeLFLRU always performed best, and LFLRU was close behind. For byte miss rates, the two algorithms exchanged positions with LFLRU always performing best with SizeLFLRU close behind. This consistency can be explained by the fact that all three models mimic an educational environment in which there is a certain sequence in which the videos should be viewed. Though clients do view them out of order, there is an underlying set of video objects that are very popular for a given time, and then lose their high popularity ranking to other sets as time passes by. So the popularity-weighted LRU and size- and popularity-weighted LRU algorithms perform best. Interestingly, but not unexpectedly, for DVJ2 the top two miss rate algorithms were SizeLFLRU and LFLRU, respectively, and the top two byte miss rate algorithms were LFLRU and SizeLFLRU, respectively. The DVJ2 top performing algorithms match those of the stochastically generated video server trace sets.

Table 6.6
Summary of Top Cache Replacement Algorithms for Stochastically Generated Video Server Traces

Trace	Exponential	Binomial	Triangular
Miss Rate	SizeLFLRU (1st) LFLRU (2nd) LRD (3rd)	SizeLFLRU (1st) LFLRU (2nd) SPACExAGE (3rd)	SizeLFLRU (1st) LFLRU (2nd) LRD (3rd-tie) SizeLRD (3rd-tie)
Byte Miss Rate	LFLRU (1st) SizeLFLRU (2nd) LRD (3rd)	LFLRU (1st) SizeLFLRU (2nd) LRU (3rd-tie) SPACExAGE (3rd-tie) SPACExAGEtc (3rd-tie) PLRU (3rd-tie)	LFLRU (1st) SizeLFLRU (2nd) LRD (3rd)

Many of the algorithms perform close to the same miss rates and byte miss rates as the top performers. These algorithms generally do a good job of following the popularity of the cached objects, and they evict the objects soon after they lose their popularity. But while there are a number of cache replacement algorithms that perform well, there are several that do not perform well at all. SizeL, PerfectLFU, LFU-DA, SLRU, GDS1, GDSPacket, GDSHits, and LRV all perform about equally poorly. Surprising among those are GDSHits and LFU-DA which are among the best performers for Web-based proxy servers. PerfectLFU, LFU-DA, SLRU, and GDSHits all rely too heavily on access counts. Because of this, they do not evict objects that were popular soon enough after their time of popularity. This causes the currently popular objects to miss too often early in their popularity. Conversely, SizeL, GDS1, and GDSPacket rely too heavily on the size of the object which gives no indication of the current popularity of the objects in the cache. And LRV has a combination of problems which will be discussed in the next section.

6.1.6 A Summary of the Results

First, a number of general conclusions can be made about the effectiveness of various statistics in cache replacement algorithms. Augmenting the LRU and FIFO cache replacement algorithms with access count and size object statistics generally improve the cache performance. Also, if only one statistic were to be used with a recency measure like LRU, then for miss rate it should be object size while for byte miss rate it should be object access count. This can be explained by the fact that when optimizing for miss rate, many small, less popular objects in the cache will satisfy many more requests than a few popular, large objects. However, for byte miss rate, the popularity of the object comes into play more, and those popular large objects cause a greater byte miss rate if they miss than those many small, less popular objects. Also, using the object type in the cache replacement algorithm generally did not gain any performance improvement. This can be explained by looking at the tables that involved object types in Chapter 5. In these tables there is very little distinction in percentages between first time accesses and subsequent accesses. In other words, there is no way of confidently determining whether objects will be reaccessed based on the object type.

The findings in this study help validate the findings of many studies that have already been published. [RD90] and [PR94] both found that using reference frequency (access count) along with recency (LRU) was important in improving cache performance over just using LRU. Also, [Red97], which disclosed the SPACExAGE algorithm, concluded that object size, when used with LRU, improved cache performance. [WAS$^+$96] found that object size algorithms performed best for miss rate while frequency-based algorithms generally performed better for byte miss rate. [TVDS98] and [AW97] found that cache replacement algorithms did not improve much when using file type as a statistic.

All of the above findings from other studies were substantiated by the findings of this study, but this study went above and beyond these findings. By simulating so many cache replacement algorithms with so many actual and synthesized workload traces, a more thorough conclusion can be drawn as to which cache replacement algorithms were best for each domain and performance measure. The results are shown in Tables 6.2, 6.3, 6.4, 6.5, and 6.6. Generally for the Web workload traces, GDSHits performed best when optimizing for miss rate while SPACExAGE and SizeLFLRU also performed well. When byte miss rate was the optimization factor, LFU-DA performed best on the Web workload traces while SLRU and LRU also performed well. However, this generalization begins to break down when considering the client proxy cache server results. For the client proxy cache servers, GDSHits was usually the best for miss rate while LFU-DA was usually the best for byte miss rate, but SizeLFLRU also took one first place performance for miss rate while SLRU, LRD, and SPACExAGEtc had first place performances for byte miss rate. These findings generally support Dilley, Arlitt, and Perret's findings (though not entirely). In [DAP99], they found that when comparing LRU, GDSHits, and LFU-DA, GDSHits performed best for miss rate while LFU-DA performed best for byte miss rate for a SPECWeb workload. When the video domain, along with the stochastically generated video server traces, is considered, the top performing cache replacement algorithms were different than those of the Web-based workload traces. LFU, LFU-DA, and GDSHits performed best on the OnCommand hotel movie video-on-demand system for both miss rate and byte miss rate. Alternately, for the DVJ2 and stochastically generated video server traces, the SizeLFLRU, LFLRU, and LRD algorithms performed best for miss rate while LFLRU, SizeLFLRU, and LRD performed best for byte miss rate. These findings help support and justify the discussion in Section 5.3.

A few more conclusions can be drawn from these findings. The fact that GDSHits usually performs better than SizeLFLRU in miss rates for Web proxy workload traces suggests that the inflation parameter does a better job of decaying cache replacement scores. Conversely, the inflation parameter is not as appropriate as the last access statistic in measuring recency of access for the video proxy server traces. Also, generally there is not much difference in the performance of SPACExAGE and SPACExAGEtc. This means that there is not much difference in using the number of accesses since an object was last accessed versus using the number of seconds since on object was last accessed. However, in the Virginia Tech G trace, the SPACExAGE and SPACExAGEtc algorithms have a difference in miss rate of over 1%, so there can be a significant difference between their performance.

Finally, the LRV algorithm from [LRV97] performs poorly for almost every workload trace in this study. In [CI97], Cao and Irani declared that this algorithm was "prohibitively expensive" in terms of computational cost and found that their GreedyDual-Size algorithms outperformed the LRV algorithm. One possible reason for the pitiful performance of the LRV algorithm is that the parameters were not "tuned" to the given workload traces. However, most proxy cache server installations are done without having to set such parameters. And if such parameters were to be set, there is no guarantee that the traffic characteristics will remain static. The LRV algorithm and the paper that introduced it provide some interesting theoretical insight, but it has too many issues to be seriously considered for implementation in a proxy cache server.

6.1.7 What About Computational Efficiency?

Is the extra computation of these more complex cache replacement algorithms worth it? From the findings of [DAP99], the answer is a resounding yes! In that article, Dilley, Arlitt, and Perret modified a Squid client proxy to implement LRU, LFU-DA, and GDSHits. They used SPECWeb simulated input and found that the CPU demand actually decreased even when the cache replacement algorithms were more complex. This was due to the decrease in miss rate. They also found that there was not a significant change is system utilization; that is, the system was not taxed any more than it was using the LRU replacement algorithm. These findings deserve some more discussion. So much of the latency within the server is in the I/O system. By reducing the miss rate, the loading on the I/O system is potentially reduced. And by reducing the byte miss rate, the bandwidth that the server requires from its network is decreased, which can lead to better throughput and lower network usage charges.

In its simplest implementation, LRU involves comparisons because only the time of each cached object's most recent request needs to be stored, and the object with the oldest recent request is evicted. Each of these algorithms (except for LRV) involves zero or more multiplications along with possibly an add and/or a division operation. Using the notation from Chapter 4, Cn is the number of objects currently occupying space in the cache. To simplify the notation, let $m = Cn$. So there are $O(m)$ mathematical operations involved in calculating the cache replacement scores of the objects currently in the cache since each cache replacement score calculation involves a constant number of addition, subtraction, multiplication, and/or division operations. And the potentially expensive division operations could be eliminated if the score calculation is of $1/x$ form by simply inverting the formula and evicting the highest scoring objects. The greatest amount of computation done with cache replacement algorithms is the comparison of all of the cache replacement scores, which is common to all of the algorithms including LRU. There are $\Omega(m \ln m)$ comparisons for any sorting algorithm [CLR90], which overshadows the $O(m)$ mathematical operations. Only LFU, SLRU, and LRV have a maintenance procedure that occasionally must be run and do not have any more complexity than a score comparison. And the overhead of these extra computations, and even the maintenance procedures, are rather small compared to the series of cache replacement score comparisons.

But do these few extra computations significantly improve the performance of the proxy cache server? Tables 6.7 and 6.8 show the average miss rates and average byte miss rates of LRU and the top performing cache replacement policy for each of the fifteen (including the model generated) workload traces. The right-most column shows the percentage improvement of the top cache replacement algorithm over LRU. For most of the workload traces, Table 6.7 shows that the average miss rate improvement is over 10%, with each of the origin server traces showing around 50% improvements. For the average byte miss rates in Table 6.8, the improvements are not as dramatic, though many still have 5% improvements. The exceptions are the network proxy server traces and the client proxy server traces. For these two domains, LRU performs quite well in optimizing for the byte miss rate. This can be attributed to the fact that the percentages of once-only requested object are quite high, and LRU is effective in determining which objects have not been accessed for a long time so those can be evicted.

So, in general, the added computational burden is reasonably offset by the improvements that are gained by using a more effective cache replacement algorithm. Yes, these lower miss rates and byte miss rates will mean more stress on the disk subsystems and motherboard backplanes of the proxy cache server, but those can usually be upgraded with more parallelism in the RAID disks, more (and sometimes faster) FibreChannel, SCSI, or Firewire chains, more PCI buses, and more main memory. The investment in better hardware will easily be offset by the savings in network bandwidth and/or the addition of more proxy cache server machines.

6.2 Does an Admission Policy Improve Cache Performance?

The second question built on the first question and asked: Does an admission policy improve cache replacement policy performance? And if so, which algorithms benefit from using an admission policy? This question asked whether it is more beneficial to include the currently requested object in the eviction process if there is no room to accommodate the currently requested object. Most cache replacement

Table 6.7
Comparison of Improvement of Top Cache Replacement Algorithm over LRU for Miss Rate (MR)

Trace	LRU MR	Top Algorithm	Top Algorithm MR	% Improvement
WorldCup	0.210	GDSHits	0.098	53.3%
dsml	0.269	GDSHits	0.137	49.1%
VaTech BR	0.407	GDSHits	0.197	51.6%
Boulder1	0.888	GDSHits	0.838	5.6%
San Jose	0.859	GDSHits	0.818	4.8%
Urbana Champaign	0.884	GDSHits	0.851	3.7%
Boeing	0.747	GDSHits	0.631	15.5%
Stack	0.678	SizeLFLRU	0.608	10.3%
VaTech BL	0.741	GDSHits	0.668	9.9%
VaTech G	0.732	GDSHits	0.658	10.1%
OnCommand	0.385	LFU	0.373	3.1%
DVJ2	0.104	SizeLFLRU	0.095	8.7%
Exponential	0.262	SizeLFLRU	0.228	13.0%
Binomial	0.240	SizeLFLRU	0.217	9.6%
Triangular	0.375	SizeLFLRU	0.349	6.9%

Table 6.8
Comparison of Improvement of Top Cache Replacement Algorithm over LRU for Byte Miss Rate (BMR)

Trace	LRU BMR	Top Algorithm	Top Algorithm BMR	% Improvement
WorldCup	0.331	LFU-DA	0.292	11.8%
dsml	0.590	LFU-DA	0.580	1.7%
VaTech BR	0.497	LFU-DA	0.475	4.4%
Boulder1	0.840	LFU-DA	0.812	3.3%
San Jose	0.954	LFU-DA	0.946	0.8%
Urbana Champaign	0.912	LFU-DA	0.901	1.2%
Boeing	0.954	SLRU	0.945	0.9%
Stack	0.846	LRD	0.843	0.4%
VaTech BL	0.796	LFU-DA & LRD	0.792	0.5%
VaTech G	0.884	LFU-DA	0.878	0.7%
OnCommand	0.385	LFU	0.373	3.1%
DVJ2	0.105	LFLRU	0.097	7.6%
Exponential	0.264	LFLRU	0.234	11.4%
Binomial	0.241	SizeLFLRU	0.223	7.5%
Triangular	0.377	LFLRU	0.357	5.3%

algorithms in use today do not use an admission policy and assume that the currently requested object is more valuable than some other objects already in the cache.

The fact that an admission policy will not necessarily affect all of the cache replacement algorithms has already been discussed. But which cache replacement algorithms specifically are affected has not been discussed. The algorithms that have different results when an admission policy is implemented are OPT, SizeL, PerfectLFU, LFU-DA, LRU-MIN, SLRU, GDS1, GDSPacket, and GDSHits; this was shown in the rightmost column of Table 6.1, labeled "Admission Effect". For each of these algorithms, the score of an incoming object could be the lowest score, and would then not be admitted into the cache.

For this comparison, bar graphs were generated for most of the actual workload traces described in Chapter 5. They are organized in the same order as they were discussed for cache replacement algorithms earlier in this chapter for each of the server types: origin, network, client and video. Each of the bar graphs have each of the cache replacement algorithms mentioned above along the x-axis. For each of the cache replacement algorithms, there are four bars. The left-most bar is the average miss rate (mr) for the cache replacement without an admission policy while the bar immediately to its right is the average miss rate (mr) for the cache replacement with an admission policy. So these two bars should be compared to determine whether using an admission policy improves the miss rate of the cache replacement algorithm. Furthermore, the right-most bar for a given cache replacement policy's set of bars is the average byte (or weighted) miss rate (wmr) for the cache replacement with an admission policy. This bar should be compared to the bar immediately to its left which is the average byte (or weighted) miss rate (wmr) for the cache replacement without an admission policy. This comparison determines whether using an admission policy improves the byte miss rate of a given cache replacement algorithm. These graphs are useful for comparing how much an admission policy improves (or does not improve) the miss rate and byte miss rate performance of a cache replacement algorithm.

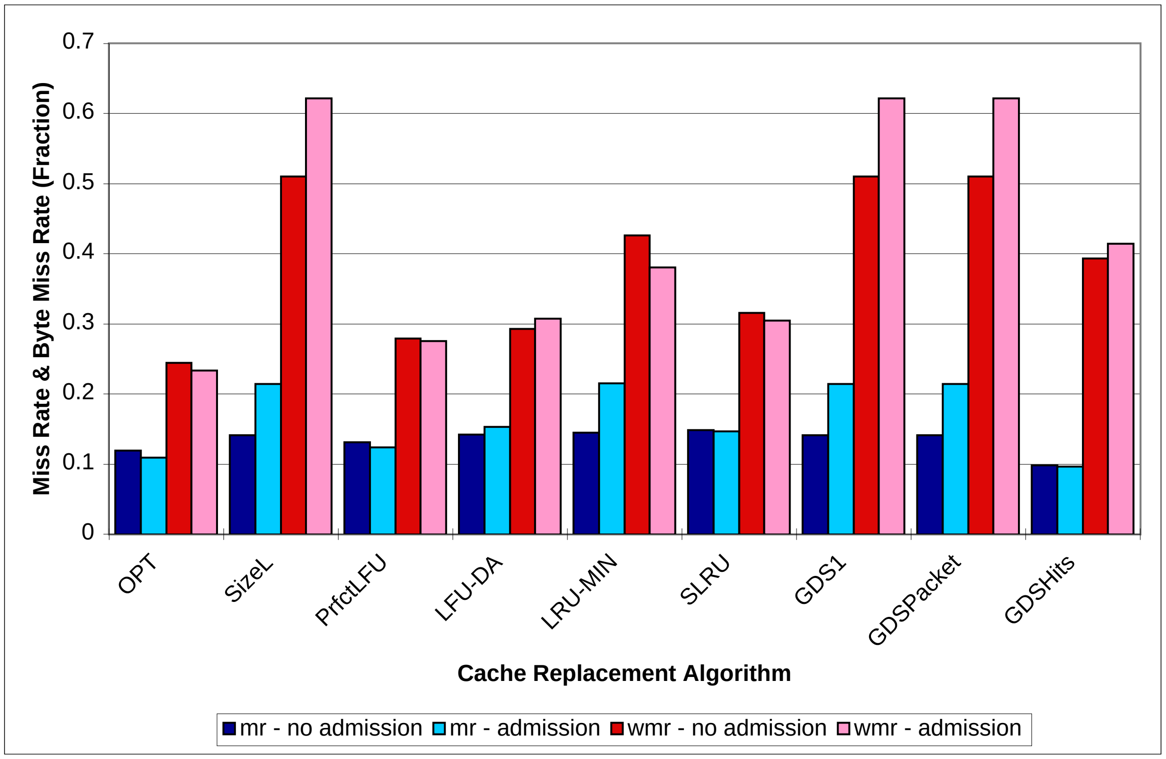


Fig. 6.127. Admission Policy Comparison of HP World Cup 1998 Web Server Farm wc_week14 Trace.

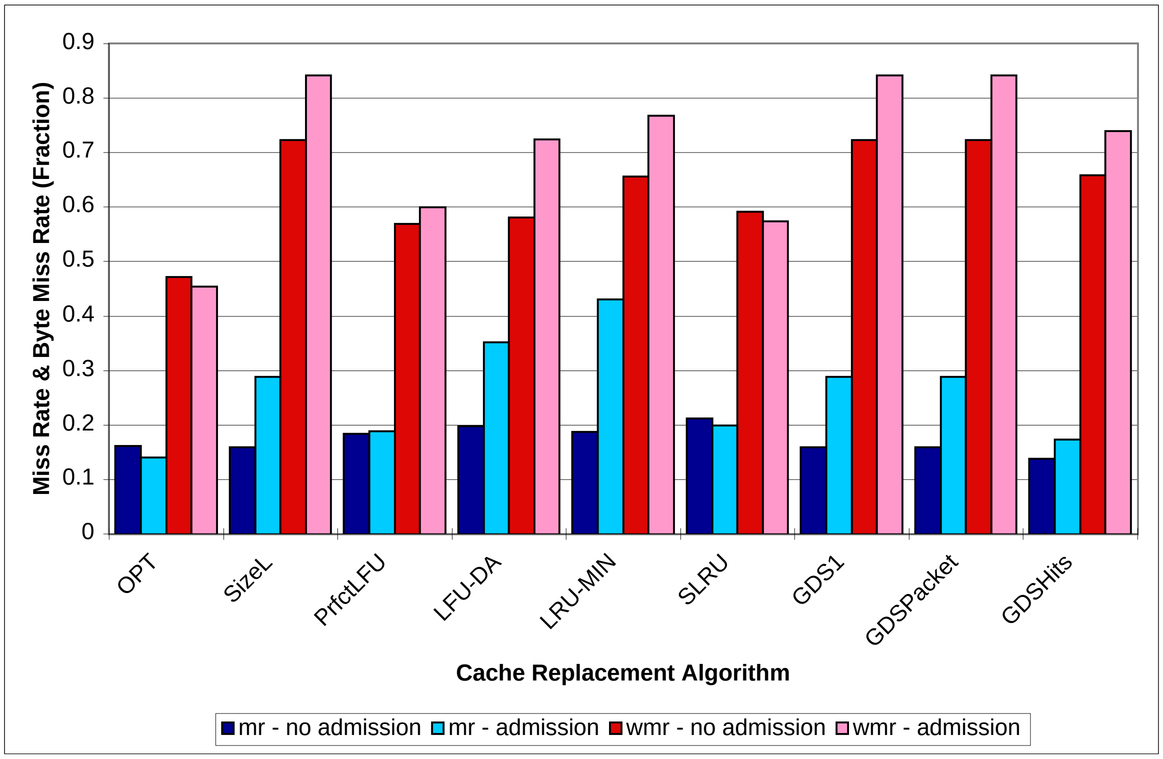


Fig. 6.128. Admission Policy Comparison of dsl_log DSML Web Server Trace.

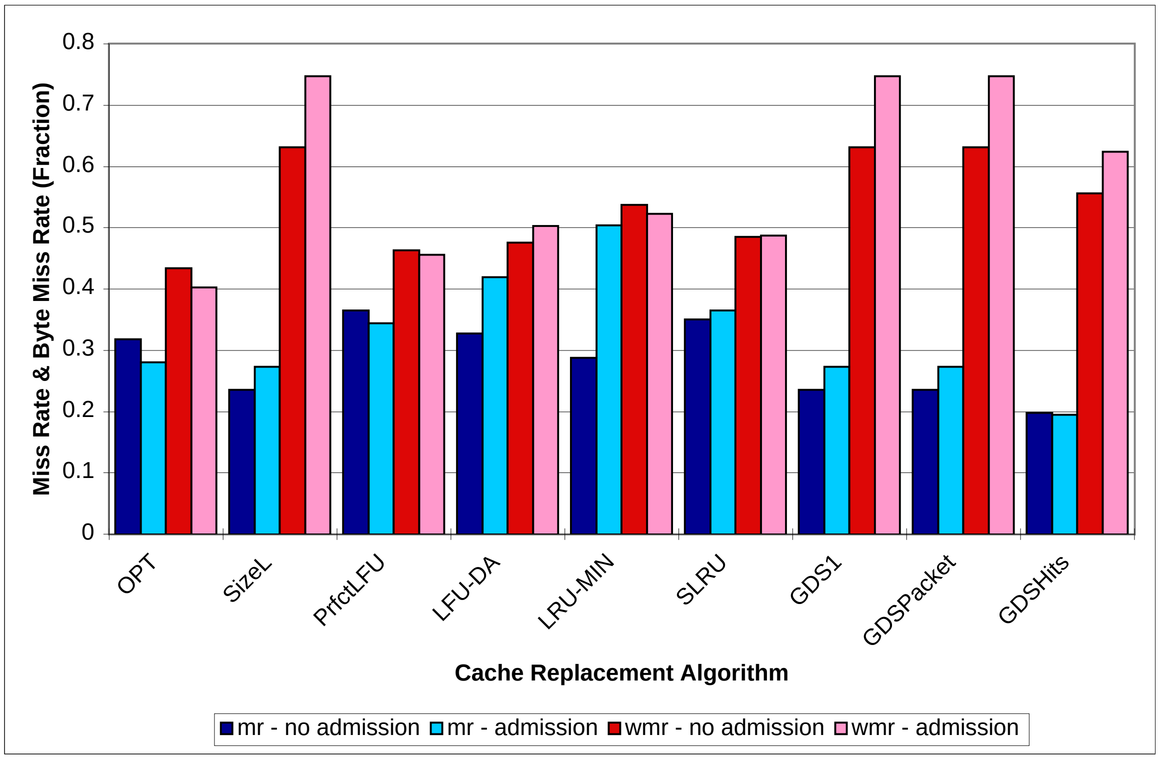


Fig. 6.129. Admission Policy Comparison of Virginia Tech BR Trace.

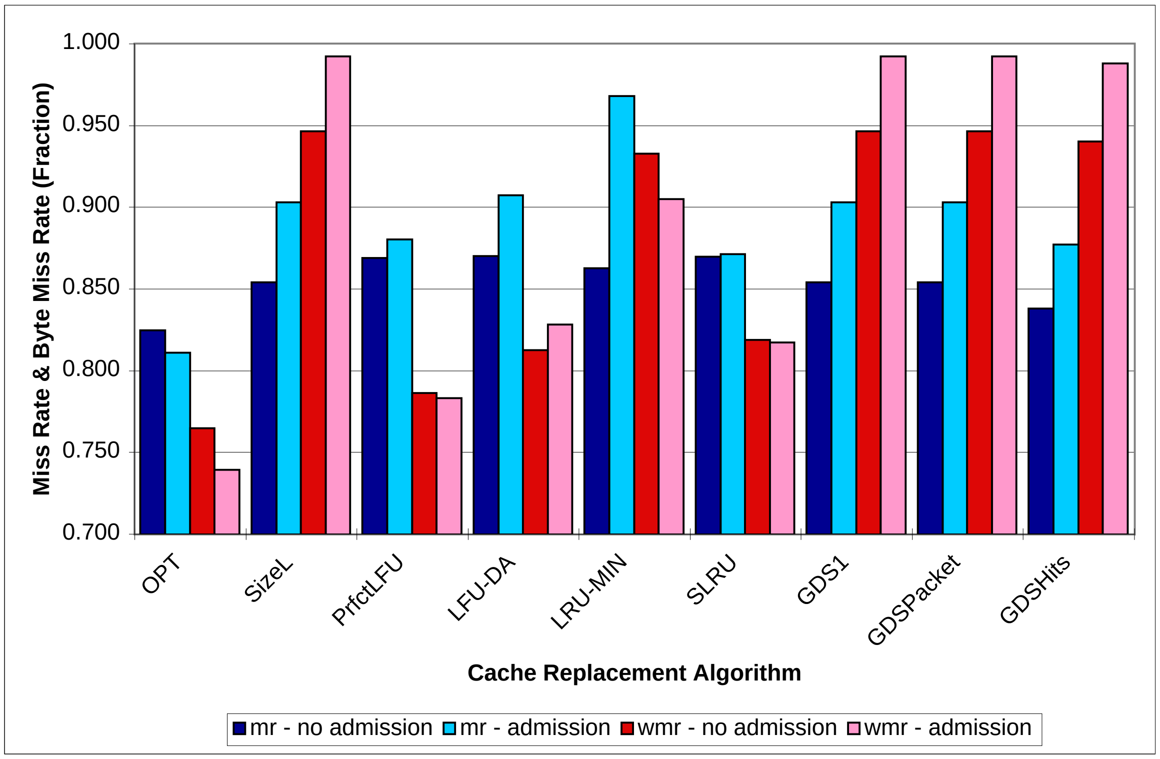


Fig. 6.130. Admission Policy Comparison of Boulder1 Proxy Server bo1_day1 Trace.

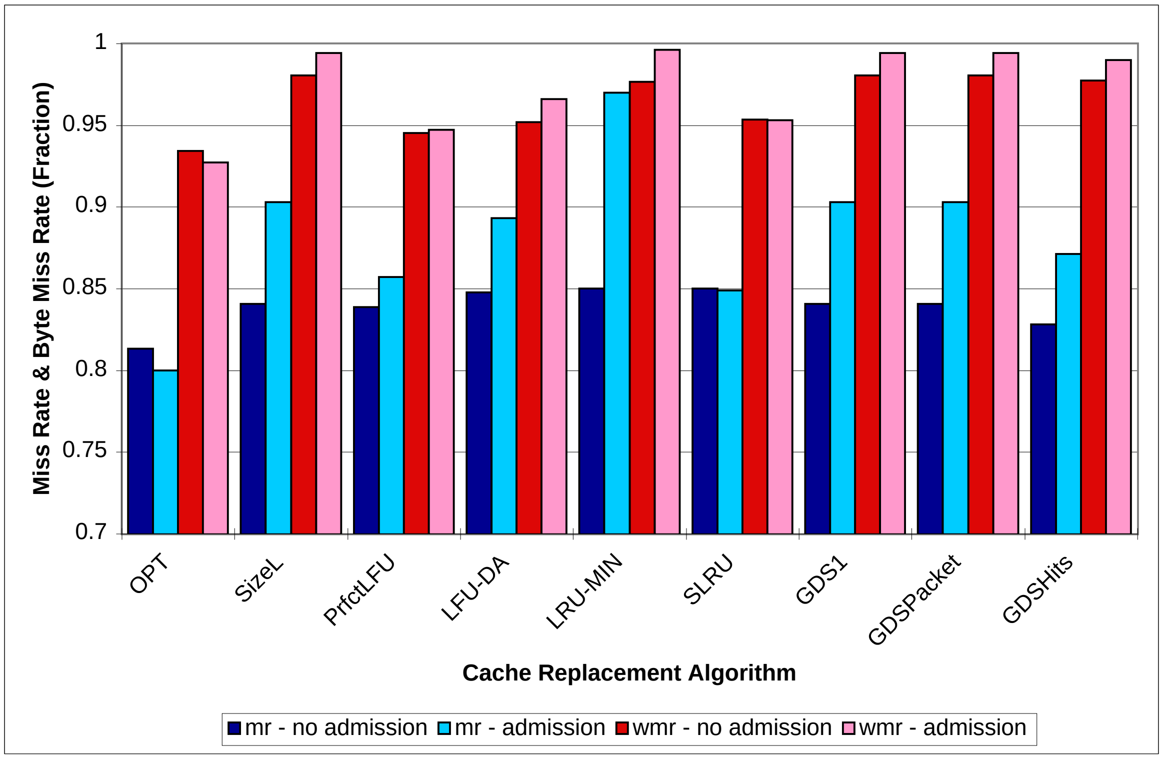


Fig. 6.131. Admission Policy Comparison of San Jose Proxy Server sj_novwk5 Trace.

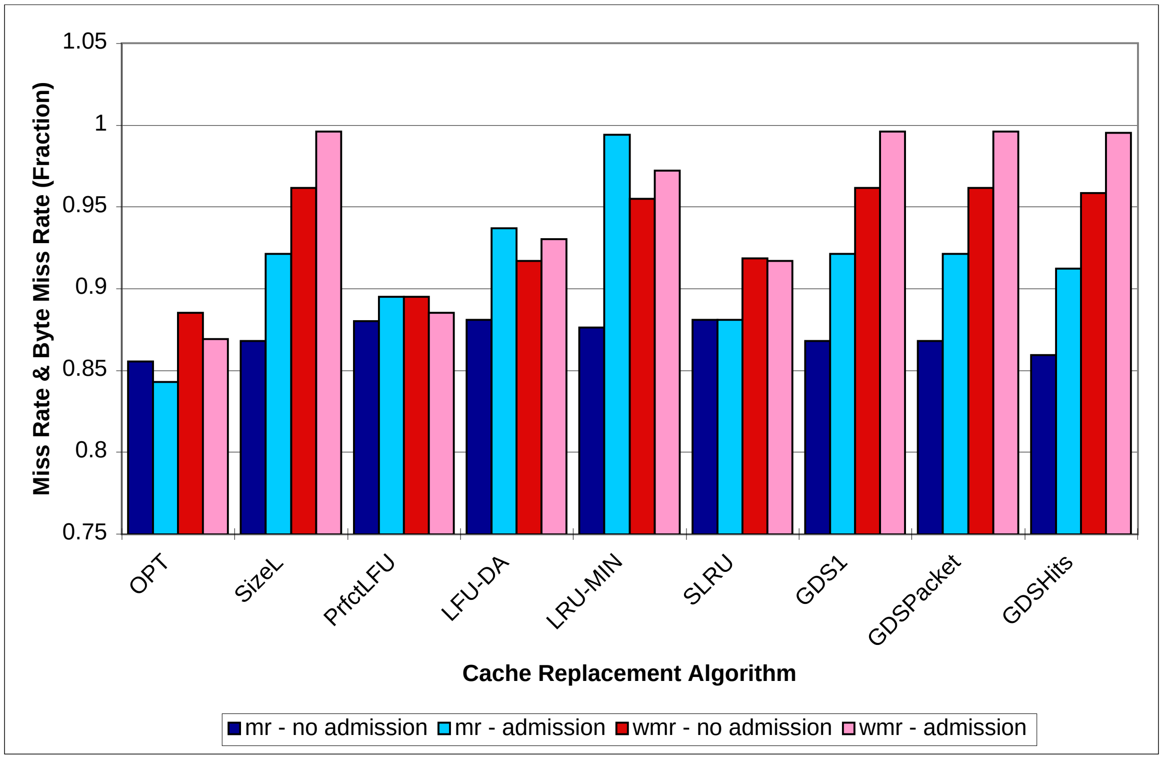


Fig. 6.132. Admission Policy Comparison of Urbana Champaign Proxy Server uc_day1 Trace.

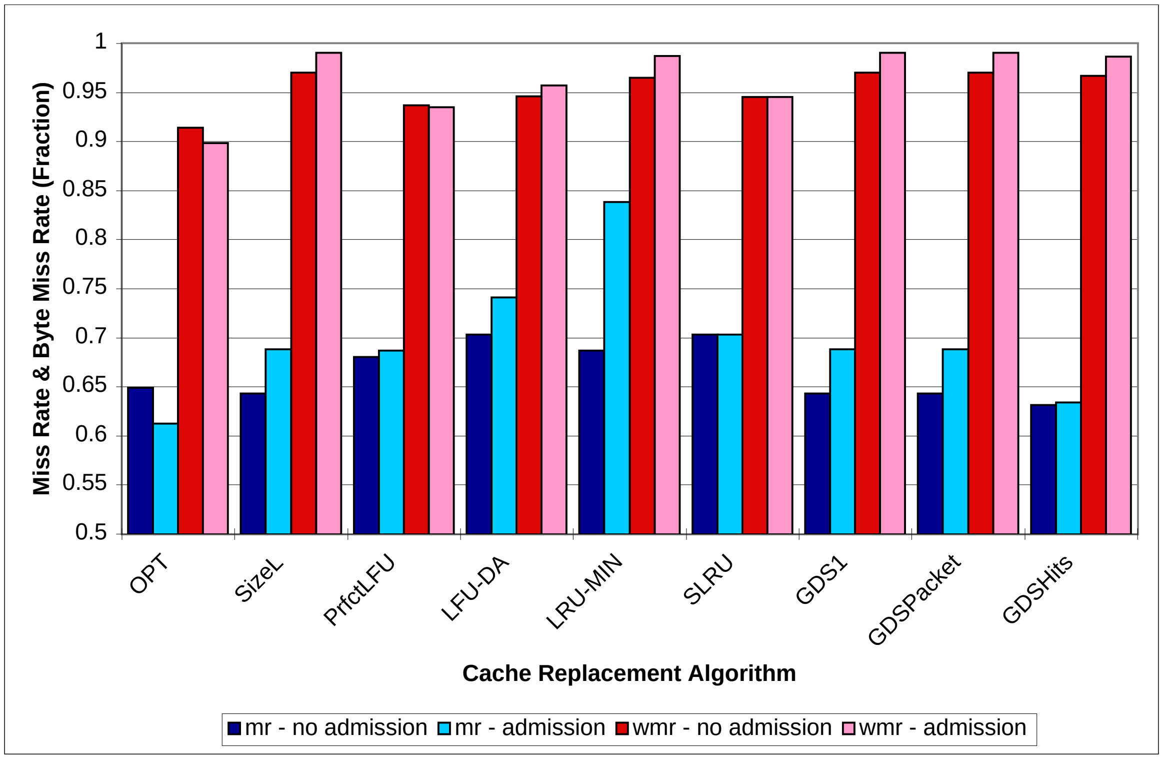


Fig. 6.133. Admission Policy Comparison of Boeing Proxy Cache Server boeing.990301 Trace.

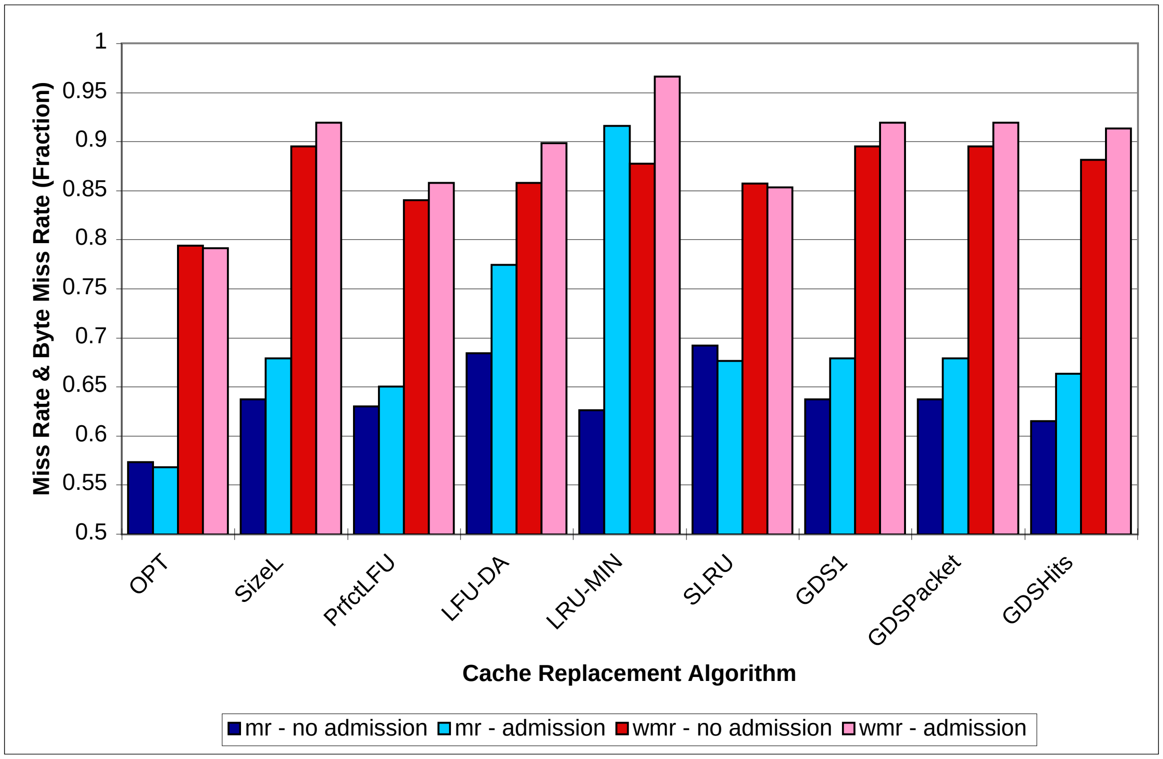


Fig. 6.134. Admission Policy Comparison of Purdue Stack Proxy Server stackproxy Trace.

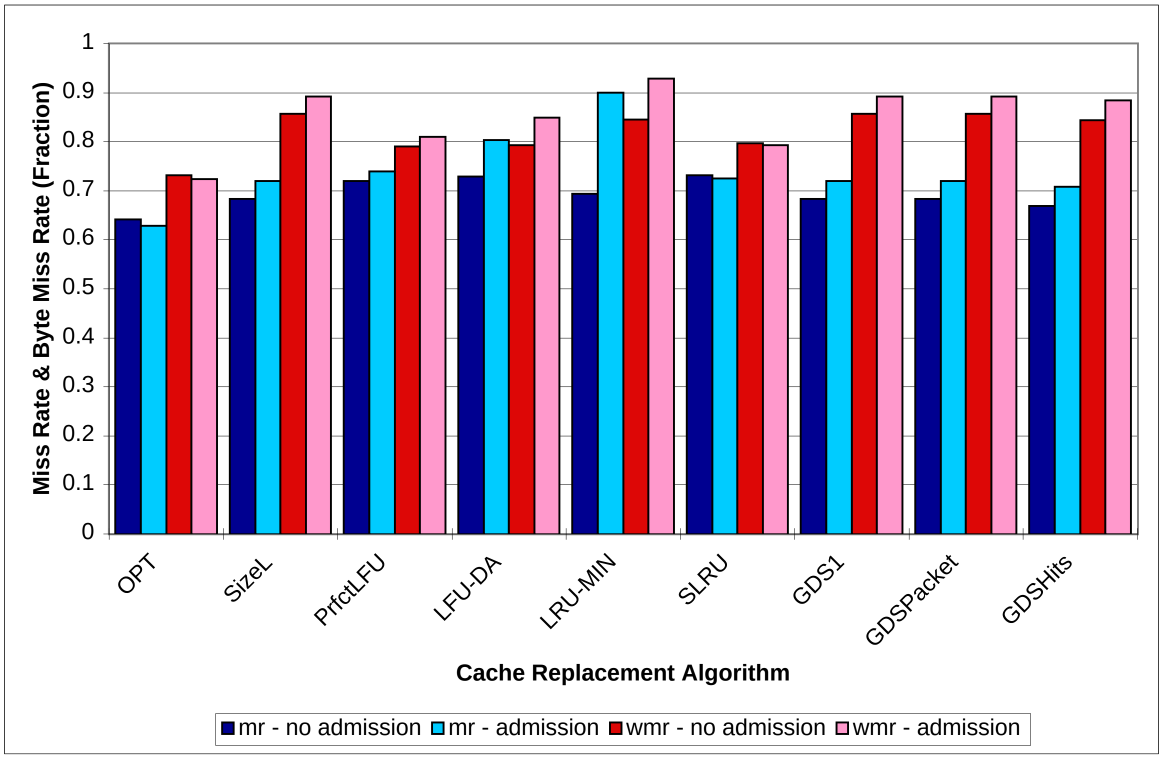


Fig. 6.135. Admission Policy Comparison of Virginia Tech BL Trace.

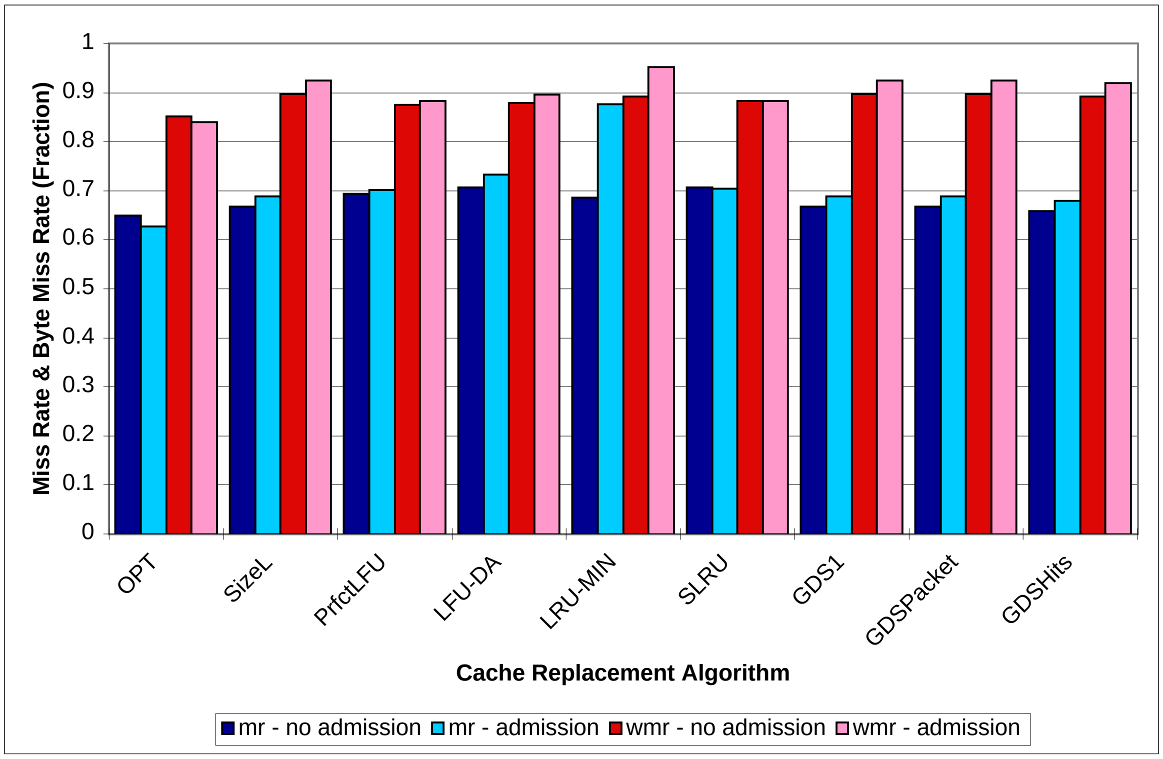


Fig. 6.136. Admission Policy Comparison of Virginia Tech G Trace.

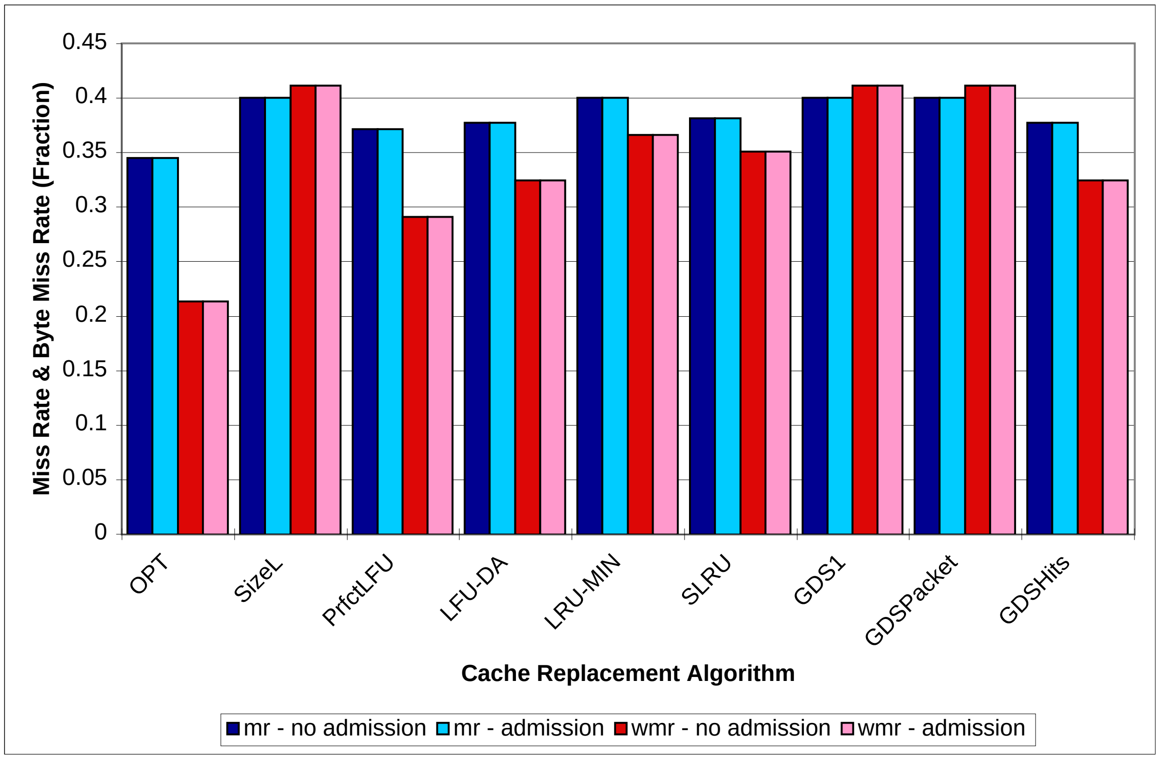


Fig. 6.137. Admission Policy Comparison of OnCommand On-Demand Movie Server daysum1 Trace.

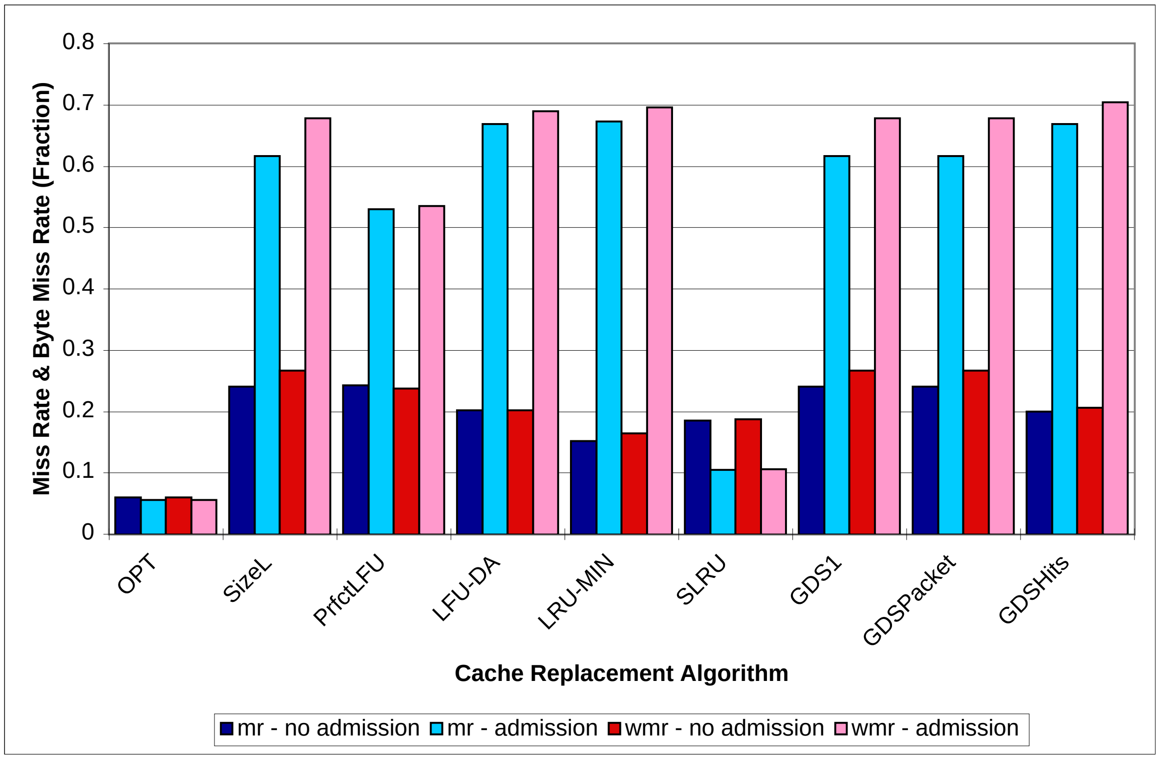


Fig. 6.138. Admission Policy Comparison of DVJ2 Educational Multimedia Video Server dvj2_99f Trace.

The results that are captured in the graphs described in the previous paragraph are also displayed in Tables 6.9 and 6.10. Table 6.9 presents the difference in miss rates for using admission policy with the cache replacement algorithms that are affected by using admission policy, while Table 6.10 shows the difference in byte miss rates for the same. In the tables, if a number is in parentheses, the corresponding cache replacement algorithm with admission policy has a lower average miss rate or average byte miss rate than that same cache replacement algorithm without admission policy.

Table 6.9
Differences in Average Miss Rates of Cache Replacement Policy using Admission Policy

	OPT	SizeL	PrfctLFU	LFU-DA	LRU-MIN	SLRU	GDS1	GDSPkt	GDSHits
WorldCup	(0.010)	0.073	(0.008)	0.011	0.071	(0.002)	0.073	0.073	(0.002)
dsml	(0.021)	0.129	0.005	0.154	0.243	(0.014)	0.129	0.129	0.036
VaTech BR	(0.038)	0.038	(0.021)	0.092	0.216	0.014	0.038	0.038	(0.003)
Boulder1	(0.014)	0.049	0.011	0.037	0.106	0.001	0.049	0.049	0.039
San Jose	(0.013)	0.063	0.018	0.046	0.120	(0.001)	0.063	0.063	0.043
UrbChamp	(0.013)	0.053	0.015	0.056	0.118	0.000	0.053	0.053	0.053
Boeing	(0.037)	0.045	0.007	0.038	0.151	0.000	0.045	0.045	0.003
Stack	(0.005)	0.042	0.020	0.090	0.290	(0.016)	0.042	0.042	0.048
VaTech BL	(0.013)	0.036	0.020	0.074	0.207	(0.007)	0.036	0.036	0.039
VaTech G	(0.022)	0.021	0.008	0.027	0.191	(0.002)	0.021	0.021	0.021
OnCmnd	(0.132)	0.011	(0.080)	(0.053)	(0.034)	(0.030)	0.011	0.011	(0.053)
DVJ2	(0.005)	0.376	0.288	0.466	0.522	(0.081)	0.376	0.376	0.468

For any of these algorithms, adding the use of an admission policy allows them to deny admission to objects that don't score high enough to be brought into the cache. That is, the objects are denied admission because their cache replacement score is not competitive enough to be admitted into the cache.

From these tables, for both miss rate and byte miss rate, using a admission policy with the OPT and SLRU algorithms consistently improve upon the performance of those algorithms. This improvement is usually around 1% better average miss rate

Table 6.10
Differences in Average Byte Miss Rates of Cache Replacement Policy using Admission Policy

	OPT	SizeL	PrfctLFU	LFU-DA	LRU-MIN	SLRU	GDS1	GDSPkt	GDSHits
WorldCup	(0.011)	0.111	(0.004)	0.015	(0.046)	(0.011)	0.111	0.111	0.021
dsml	(0.017)	0.118	0.030	0.144	0.111	(0.018)	0.118	0.118	0.081
VaTech BR	(0.031)	0.116	(0.008)	0.027	(0.015)	0.002	0.116	0.116	0.068
Boulder1	(0.026)	0.046	(0.003)	0.016	(0.028)	(0.002)	0.046	0.046	0.048
San Jose	(0.007)	0.014	0.002	0.014	0.020	(0.000)	0.014	0.014	0.013
UrbChamp	(0.016)	0.035	(0.010)	0.013	0.017	(0.001)	0.035	0.035	0.037
Boeing	(0.016)	0.020	(0.002)	0.011	0.022	0.000	0.020	0.020	0.019
Stack	(0.003)	0.024	0.018	0.040	0.089	(0.004)	0.024	0.024	0.032
VaTech BL	(0.008)	0.035	0.020	0.057	0.084	(0.004)	0.035	0.035	0.041
VaTech G	(0.012)	0.027	0.007	0.017	0.060	0.000	0.027	0.027	0.027
OnCmnd	(0.132)	0.011	(0.080)	(0.053)	(0.034)	(0.030)	0.011	0.011	(0.053)
DVJ2	(0.005)	0.412	0.298	0.487	0.532	(0.082)	0.412	0.412	0.498

and average byte miss rate, but for some of the workload traces, the improvement is more significant. This finding should be expected for the OPT algorithm, because it allows the algorithm to deny admission to objects that are never requested again. (This is an advantage of being an ideal algorithm and being able to see into the future.) PerfectLFU also gains an advantage for using an admission policy for some of the workload traces, but it is not as consistent. However, PerfectLFU is similar to an ideal algorithm because it uses much more storage space for its metadata and does not have any cache replacement score decaying technique. The OnCommand trace had many of its cache replacement algorithms improved with an admission policy especially among the access count based algorithms of PerfectLFU, LFU-DA, SLRU, and GDSHits.

When the admission policy did not improve the performance of the cache replacement, usually the admission policy did not raise the average miss rate and average byte miss rate by more than 5%. The cache replacement algorithms for which the admission policy never improved the average miss rate and average byte miss rate are those based on object size. Therefore, object size is generally not an effective statistic for determining the cachability of requested objects with an admission policy. On the other hand, the access count can sometimes help with determining cachability as is displayed by SLRU and GDSHits.

These findings indicate that for most cache replacement policies, it is very difficult to consistently determine the cachability of objects immediately after they are requested. Therefore, it is not effective to use an admission policy for use with most cache replacement algorithms. This result supports the findings of Kurcewicz, Sylwestrzak, and Wierzbicki in [KSW98].

6.3 Should Video Objects Be Cached in Their Entirety?

For Web proxy cache servers, the question of whether to cache the entire object or just a portion of the object has never come into question; web proxy caches always cache entire objects. However, the same question should still be debated for video cache proxy servers. Some of this indecision stems from the fact that video proxy servers are only in their infancy. Only a few rough systems have been released or will be released soon (as mentioned in Section 2.2). Another reason for this indecision has been the extensive work on using buffers, rather than caches, in the video object delivery stream. Among these studies are [NY94, PRR94, DDM$^+$95, KRT95, NS95, SC98], as well as several papers from the Fellini project at Bell Labs, including [OBRS95, ORS96].

As disk drive, networking, and motherboard bus technologies continue to improve, the possibility of quickly downloading an entire video from the origin video server to a video proxy cache server is becoming more realistic. When a video is requested, the origin video server pushes the video object data onto the video proxy cache server. Once the video proxy cache server has received enough of the video data to start serving it to the client, the proxy starts the stream to the client's machine. Meanwhile, the origin video server continues to push the rest of the video data file to the proxy. Since the backbone of the Internet has large bandwidth, the entire video data is done being pushed to the proxy server long before the end of the video is reached by the client.

The determination of whether to cache entire video objects relies upon whether the clients are likely to watch a significant enough amount of each video to make it worthwhile to download the entire video to an intermediate video proxy cache server. To make this determination, a number of statistics will be investigated about the OnCommand and DVJ2 workload traces.

First, the viewing sessions of the OnCommand workload trace will be investigated. As was mentioned in Section 5.6, it was assumed that all videos were two hours long (7,200 seconds). There is a preview function which allows clients to view a video for two minutes before being billed. A breakdown of video session duration percentages is shown in Figure 6.139. From this graph, one can see that only 26% of the video sessions were of the preview duration; all of the other sessions viewed the preview and continued viewing the video to its end. This strongly influences the mean and median video viewing duration which are 5,308 seconds (88 minutes) and 7,200 seconds (120 minutes), respectively, as shown in Table 6.11. The mean equals 73.7% of the total (assumed) duration of the video objects. These statistics imply that a large percentage of the videos that are requested are viewed in their entirety. This result should be expected since the clients are paying to view each of the videos and generally will watch the whole program to get their money's worth. Figure 6.140 shows the raw network traffic bandwidth (in video seconds) used by the three top performing byte miss rate cache replacement algorithm (LFU, LFU-DA, and GDSHits) and compares their performance to the bandwidth usage for when clients only view an average of 10, 25, 50, and 75 percent of the total videos. The percentages are the lines with legend labels of 10% NC, 25% NC, 50% NC, and 75% NC (where NC stands for no cache). So even for the smallest cache size in this simulation, caching an entire video object is more efficient than not caching at all or just buffering. These findings strongly suggest that video objects for the OnCommand system should be cached in their entirety.

Table 6.11
Statistic of OnCommand and DVJ2 Workload Video Session Durations

Statistic	OnCommand	DVJ2
Mean (sec)	5308.03	1068.60
Median (sec)	7200	289
St. D. (sec)	3162.63	1582.12
Minimum (sec)	1	0
Maximum (sec)	7200	23741
Count	1479	11136

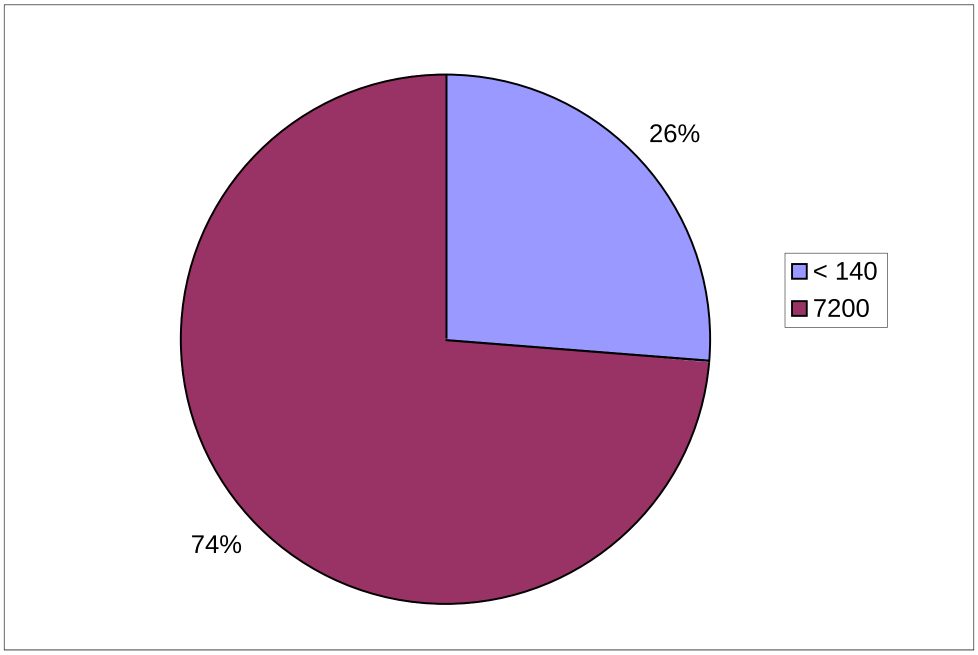


Fig. 6.139. OnCommand Workload Video Session Duration Percentages.

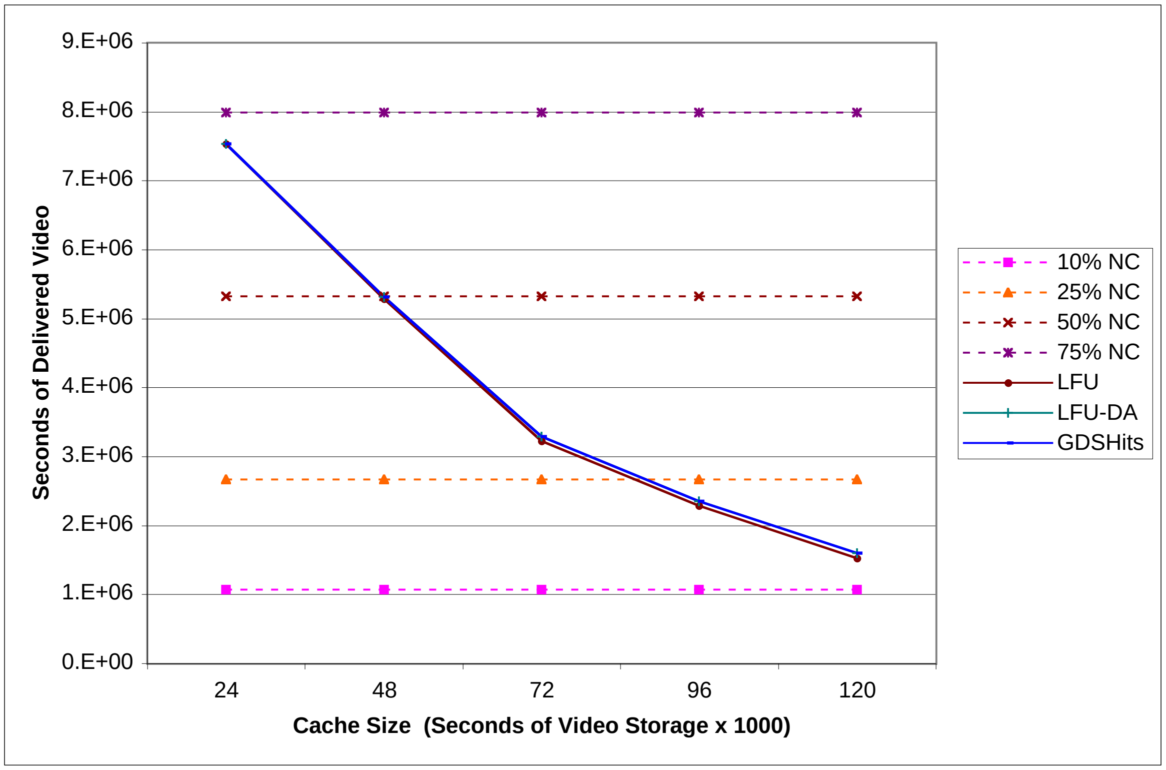


Fig. 6.140. OnCommand Network Traffic Comparison of Top Cache Replacement Algorithms and No Cache Bandwidth Usage.

The DVJ2 system is far different from the OnCommand system. As was explained in Section 5.5, the DVJ2 system is used by students in the School of Electrical and Computer Engineering at Purdue University to view lectures and class support material as videos. This implies that the students do a great deal of starting and stopping the playing of the videos to take notes, complete homework problems, etc. This expectation is supported by statistics. The mean and median video viewing duration are 1068 seconds (17.8 minutes) and 289 seconds (4.8 minutes), respectively, as shown in Table 6.11. If it is assumed that each video session is an average of 45 minutes (2700 seconds) long (which is a reasonable estimate of an average recorded lecture object), the mean implies that 39.6% of an entire video is watched on average while the median implies that only 10.7% of an entire video is watched on average. Figure 6.141 shows that 57% of all video viewing sessions lasted less than 600 seconds (10 minutes). Assuming again that each video session is an average of 45 minutes (2700 seconds) long, 57% of all video requests result in less than 23% of the requested video being viewed. Also, Figure 6.141 presents a histogram of video session durations. As can be seen, the durations for the DVJ2 video sessions are skewed toward short sessions. Figure 6.142 shows the raw network traffic bandwidth used by the three top performing byte miss rate cache replacement algorithms (LFLRU, SizeLFLRU, and LFU) and compares their performance to the bandwidth usage for when clients only view an average of 5%, 15%, and 25% of the total videos. The percentages are the lines with legend labels of 5% NC, 15% NC, and 25% NC (where NC stands for no cache).

For the usage patterns of the DVJ2 system, it is not as easy to definitively say that the videos should be cached in their entirety. Based on the mean video session duration of 39.6% of the entire average video and the cumulative percentage graph in Figure 6.141, it is more bandwidth effective to cache the entire videos. However, the median video session duration of 10.7% of the entire average video implies that the videos should not be cached and should just be buffered in the proxy server.

This suggests a solution for the DVJ2 system of having a probationary period

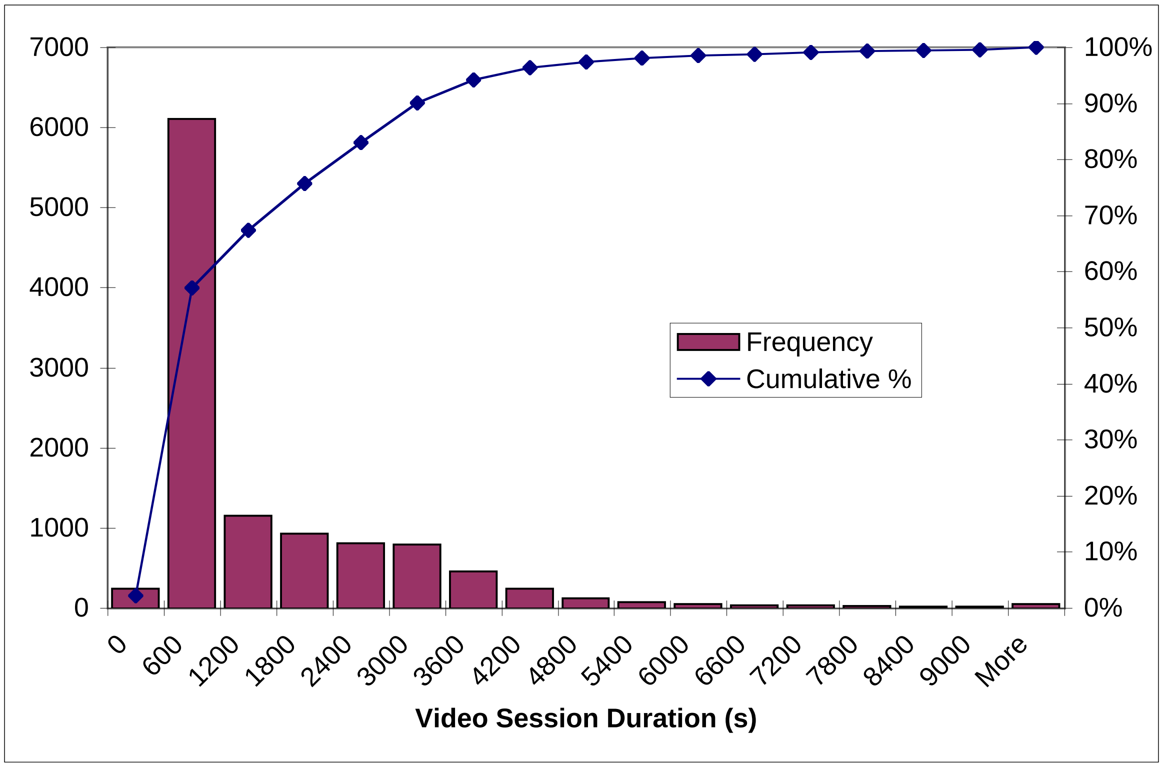


Fig. 6.141. DVJ2 Workload Video Session Duration Histogram.

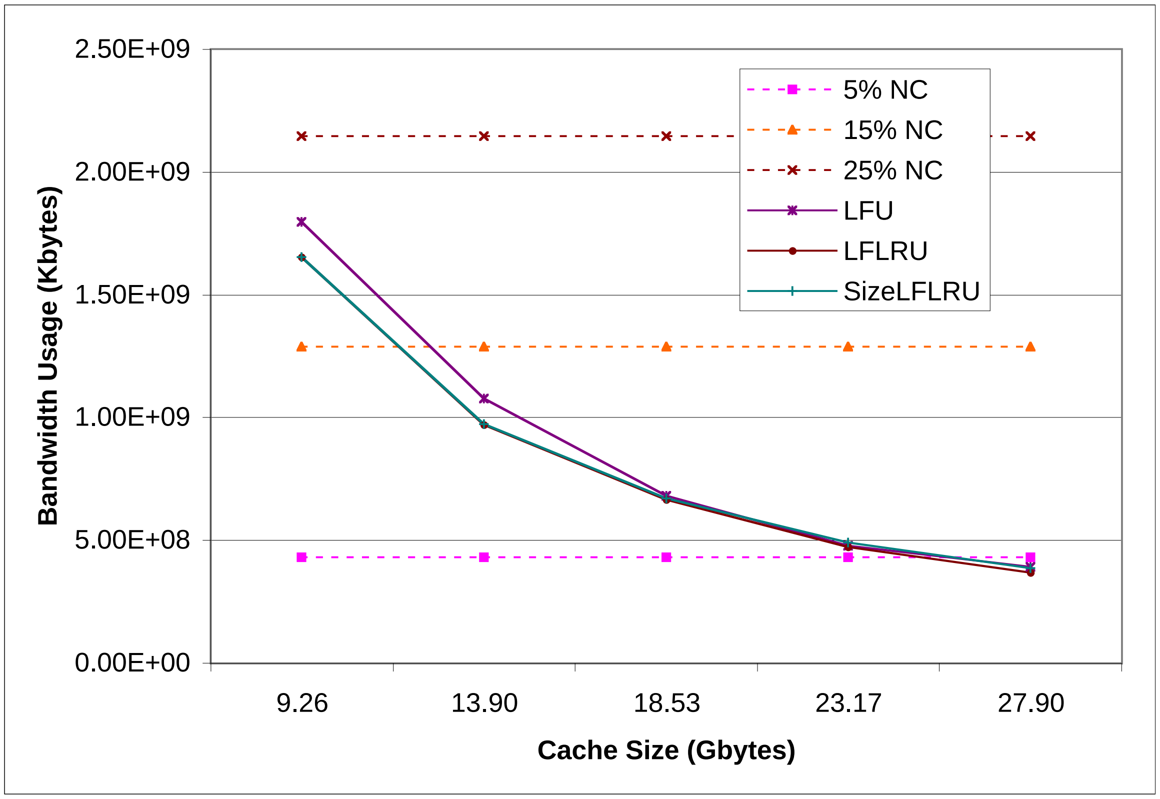


Fig. 6.142. DVJ2 OnCommand Network Traffic Comparison of Top Cache Replacement Algorithms and No Cache Bandwidth Usage.

in which a first access is directly streamed from the origin server directly to the client. As this stream is running, the cache proxy server could be making a copy of the probationary initial stream as it is passing its router. After the probationary period has passed and if the client is still viewing the video, then the proxy quickly downloads the entire video from the origin server. Once the entire video is resident on the proxy, a service handoff occurs from the origin server to the proxy server and the proxy cache server delivers the rest of the video to the client. Once it has received the entire video from the origin server, the proxy cache server can then also deliver that same video to other clients that it is serving.

In the next chapter, the findings of this dissertation will be brought together, and the broader implications of this research are discussed.

7. Summary and Conclusion

7.1 Summary

This study explored whether higher complexity cache replacement algorithms could improve the miss rates and weighted miss rates of proxy cache servers for multimedia data, and whether an admission policy improved the performance of cache replacement algorithms. Also, the question of whether whole video objects should be cache in the proxy servers was addressed.

The dissertation started with a discussion of caches in distributed multimedia systems, explained the importance of cache replacement algorithms, discussed how the cache replacement problem mapped to the 0-1 knapsack problem, and then showed how cache replacement algorithms could be implemented as cache replacement scoring algorithms. Chapter 3 discussed the related works from computer hardware memory caches, operating system virtual memory direct paging algorithms, file migration algorithms, disk caching algorithms, relational database buffer management algorithms, and World-Wide Web proxy caches. Then the simulator model, simulator functionality, and result presentation methods were discussed in Chapter 4.

The simulation data sources were presented in Chapter 5. A number of the statistics for each of the trace workloads were brought together in tables to discuss the justification of delineating the three cache proxy server domains – origin, network, and client – as well as a different domain of video cache proxy servers. The representative trace workload of each of the simulation data sources were described, and their statistics were given and discussed.

Then the results of the study were presented in Chapter 6. In terms of which cache replacement algorithms performed best, a number of general conclusions were made first. Augmenting the LRU and FIFO cache replacement algorithms with access count and object size statistics generally improved the cache performance. Also, if only one statistic were to be used with a recency measure like LRU, then for miss rate it should be object size while for byte miss rate it should be object access count. And using the object type in the cache replacement algorithm generally did not gain any performance improvement.

All of the above general conclusions validate findings from other published studies as was discussed in Section 6.1.6, but this study goes above and beyond these findings. By simulating so many cache replacement algorithms with so many actual and synthesized workload traces, a more thorough conclusion was drawn as to which cache replacement algorithms were best for each domain and performance measure. Generally, for the Web workload traces, GDSHits performed best when optimizing for miss rate while SPACExAGE and SizeLFLRU also performed well. When byte miss rate was the optimization factor, LFU-DA performed best on the Web workload traces while SLRU and LRU also performed well. However, this generalization began to break down when considering the client proxy cache server results. For the client proxy cache servers, GDSHits was usually the best for miss rate while LFU-DA was usually the best for byte miss rate, but SizeLFLRU also took one first place performance for miss rate while SLRU, LRD, and SPACExAGEtc had first place performances for byte miss rate. When the video domain, along with the stochastically generated video server traces, were considered, the top performing cache replacement algorithms were different than those of the Web-based workload traces. LFU, LFU-DA, and GDSHits performed best on the OnCommand hotel movie video-on-demand system for both miss rate and byte miss rate. Alternately, for the DVJ2 and stochastically generated video server traces, the SizeLFLRU, LFLRU, and LRD algorithms performed best for miss rate while LFLRU, SizeLFLRU, and LRD performed best for byte miss rate. All of these findings help support and justify the discussion in Section 5.3 of dividing the Internet caching server placement into multiple domains as well as a video proxy server domain.

A few more conclusions were drawn from these findings. The fact that GDSHits usually performed better than SizeLFLRU in miss rates for Web proxy workload traces suggested that the inflation parameter did a better job of decaying cache replacement scores. Conversely, the inflation parameter was not as appropriate as the last access statistic in measuring recency of access for the video proxy server traces. Also, generally there was not much difference in the performance of SPACExAGE and SPACExAGEtc. This meant that there was not much difference in using the number of accesses since an object was last accessed versus using the number of seconds since on object was last accessed.

Finally, the LRV algorithm from [LRV97] performed poorly for almost every workload trace in this study. One possible reason for the pitiful performance of the LRV algorithm was that the parameters were not "tuned" to the given workload traces. However, most proxy cache server installations are done without having to set such parameters. And if such parameters were to be set, there is no guarantee that the traffic characteristics are not going to change. The LRV algorithm and the paper that introduced it provide some interesting theoretical insight, but it has too many issues for it to be seriously considered for implementation in a proxy cache server. The discussion then went on to support the fact that the added computational complexity of these more complex cache replacement algorithms was worthwhile.

Section 6.2 showed that, in general, an admission policy did not improve the performance of non-ideal cache replacement algorithms. The findings showed that it was very difficult to determine the cachability of objects immediately after they were requested.

Finally in Section 6.3, it was concluded that caching the entire videos should be done in the OnCommand hotel movie video-on-demand system because most of the video sessions on this system lasted the entire length of the videos. On the other hand, it was not entirely clear that video objects should be cached in their entirety on the DVJ2 educational multimedia system. All but one of the statistics on the DVJ2 system suggested that the video objects should be cached in their entirety.

7.2 Contributions

This research makes several important contributions to what is understood about implementing distributed multimedia proxy caches.

- It presents the fully-associative, variable block size cache replacement algorithm problem as a unique model which illuminates a new method of looking at its issues.

- It explores the performance of the cache replacement algorithms using a wide variety of traces and situations that are encountered in distributed multimedia systems, including web server traces and multimedia video system traces.

- Also, the research is the first public study to actually use video server event traces from an educational multimedia testbed system (the DVJ2 systems) and from a commercial hotel video server system (the OnCommand OCX [OnC99b] system).

- This study analyzes the statistics of the workload traces, and justifies the need to determine which proxy caching domain in which the proxy caching server will be used by showing the differences in several significant statistics for the different proxy caching domains.

- Another contribution of this study is the introduction of four new cache replacement policies which are based on observations from other studies and on the characteristics of various workload access traces.

- Most previous studies have only compared a few cache replacement algorithms at a time. This research compares most of the cache replacement algorithms from previous studies using a variety of input workload access traces from various regions of the delivery network. One goal of this research was to determine which of these replacement policies perform best for different types of systems in different proxy caching domains, thereby determining which cache object statistics are most important for determining the future access patterns for various typical system traces.

- This study explores the effectiveness of using a cache admission policy in determining whether a document should be cached when it is first accessed. An admission policy is implemented as part of each cache replacement algorithm for which an admission policy affects the cache performance.
- Finally, this study examines whether video server traces should be cached in proxy servers or should merely be buffered.

7.3 Future Research

The next steps for this research involve developing new cache replacement algorithms, exploring much larger workload traces for simulation, and gaining access to and using other workload traces in the simulations.

In terms of new cache replacement algorithms, it would be very interesting to explore the more difficult cache object statistics like time-to-live and time-to-retrieve and how they compare with the cache replacement algorithms of this study. However, as mentioned in Section 2.5, these two values are very difficult to determine for cache objects. Until a method for collecting these values in workload traces is developed, this comparison will not be feasible.

Exploring much larger workload traces is a promising area of further research. Memory and processing speed limitations in the simulation computer systems kept the size of most of the workload trace to under 350,000 requests. It would be very interesting to run simulations on workload traces of over one million requests. This will involve several improvements. First, the primary simulation machine, an Intel Pentium3-Zeon based workstation running Linux, will be upgraded to at least 512 megabytes of main memory. Also, the current Perl scripts, which translated the raw workload logfiles into simulator input files while also statistically analyzing the requests, allowed for quick code generation, but they did not manage memory as efficiently as if they had been coded in C. The 350,000 request limit was caused by the memory allocation inefficiency of Perl. So, some of the Perl scripts will be rewritten in C to implement more efficient memory management thereby hopefully allowing much larger traces to be translated. Only three of the collected trace sets had over 350,000 requests in the entire set of traces, and they are the Boeing traces, the NLANR traces, and the HP World Cup 1998 traces. So rewriting the translators will take some effort, but the results these larger consecutive traces will be interesting.

Another possibility for exploring much larger workload traces would be to rewrite the simulator. The current version of the simulator was written to accommodate the OPT ideal algorithm. This meant having to load the entire trace being simulated into memory to be able to find the next request time for the currently requested object. The simulator could be rewritten to only load in the next 10,000 or 100,000 requests for OPT to use. With this implementation, the OPT algorithm would no longer truly be Belady's Optimal algorithm, but it would still provide a very good indication of how well a cache replacement algorithm could perform. In writing this revised simulator, the Perl script translators would no longer be necessary because the parsing and interpretation of the workload trace files would be migrated into the simulator software. This would eliminate the possibility of having the Perl scripts run out of memory as was explained in the previous paragraph. However, the rewritten simulator would run slower because it would need to translate the object string identifiers – the Web uniform resource locators, video titles, etc. – into the internal integer identifiers, and it also needs to maintain the data structure that keeps track of which internal integer identifier was assigned to which object string identifier. So this rewritten simulator would probably run slower, but it would be able to handle much larger workload traces.

Finally, it would be interesting to gain access to other workload traces. The traces in this study were difficult to retrieve for a variety of reasons. Most organizations are resolute about not allowing anyone access to the workload traces of their proxy cache servers because of privacy issues. They usually think that any study using such traces is a marketing study and that they are divulging private information of their employees and their company. Even when the technical research intent of the study is emphasized, they usually continue to be very reluctant. Furthermore, unbeknownst to most company officials, the workload traces can be "sanitized" with simple Perl scripts which remove such things as the requesting machines' network addresses and the users' logins if they are recorded in the trace file. Other companies either do not keep logfile records of the accesses that their proxy servers service, or do not archive them. But as companies, schools, and other organizations become more Web savvy, they will hopefully be more willing to share their request traces for the sake of improving our understanding of Internet performance enhancing techniques. There have been some inroads toward this already. For instance, ACM SIGCOMM sponsors a Web site called the Internet Traffic Archive at `http://ita.ee.lbl.gov/` at which links to several Web workload trace archives are stored. So as more appropriate workload traces are posted to the Internet Traffic Archive and other such sites, they could be used as input for the simulator once a Perl translator is written for them.

7.4 Research Implications

From these findings, it would be recommended to include several cache replacement algorithms into Squid and other such Web proxy cache server software. In the configuration file, new parameters could be added which characterize the environment in which the proxy server would be used. From these characterizing parameters, an appropriate cache replacement policy could be used. John Dilley, Martin Arlitt, and Stephane Perret at HP Labs have made some progress in this arena by programming LFU-DA and GDSHits into a prototype version of Squid 2.0 [DAP99]. Looking further ahead, it would be very efficient for the proxy cache server software to analyze the request stream in real time and then adjust the cache replacement algorithm that it uses thereby adapting to user access patterns as they are happening. And by implementing the cache replacement algorithm using the cache replacement scoring paradigm, it will become much easier to handle different types of objects with different cache replacement algorithms. These object differentiations can be based on object type (video content, text, etc.), content type (advertisements, articles, images, etc.), or other object characteristics.

As for video-on-demand systems, there are no hierarchical multimedia systems currently in use in educational institutions nor in corporations. But once such systems start being implemented, it would be strongly recommended that the system architects of those system use the results of this study as a basis for implementing the cache replacement policies of the video proxy servers. There are a number of Internet video caching systems being developed and deployed as mentioned in Section 1.2. Among those are systems by like Apple, Akamai, and Novell with QuickTime TV, RealNetworks with Broadcast.com, and Microsoft [Gro99]. It would be very interesting to apply the findings from this study to the video caching systems that these companies are deploying including the findings on caching entire videos versus only buffering portions of the videos.

Since it has been shown that an admission policy does not improve the performance of cache replacement algorithms, it is recommended that proxy cache servers in any domain should not implement an admission policy. The only exceptions may be using one with the SLRU cache replacement algorithm. Admission policies could also be used with video proxy servers in commercial systems similar to the OnCommand system. For this workload trace from the OnCommand system, the GDSHits and LFU-DA had over 5% lower miss rate using an admission policy giving them the lowest overall miss rates.

Another way to discuss these findings is to take some of the most popular information services on the Web and determine which of the workload traces from this research are closest to them in terms of characteristics. Among these popular Web services are digital libraries like the IEEE Digital Library, subscription services like magazines and newspapers, and online stores which use massive databases like Amazon.com. For each of these services, some of their transactions are conducted over secure session which are not cachable with any technology. However, many of their transactions are not secure session transactions, and these transactions can be broken down into static and dynamic transactions. Static transactions are direct requests for a certain web object, while dynamic transactions determine what web objects are needed in a just-in-time manner. Currently, almost all of the caching being done in the Internet are static transactions. The dynamic transactions can also involve transferring software code across the transaction connection. These dynamic transactions are difficult to cache because they usually contain data that is generated exclusively for the requesting client; no other client is likely to receive the same data. However, some objects of the requests can be cached. Also, as the technology like the Active Cache project at the University of Wisconsin's Computer Science Department [CZB98] becomes more mature, the caching of software agents and other code will be cached more often in proxy cache servers.

So which of the workload traces most resembles the transaction patterns that would be observed from these digital library, subscription, and online store services? In the network and client domains, these transaction requests would behave much like any other request pattern. This is because the service accesses are just another set of accesses. It is possible that such services would be implemented in a distributed set of servers much like the 30 servers of the HP World Cup 1998 site were. This brings up the origin server domain. Within this domain the request patterns for each of these services would probably most resemble the patterns of the HP World Cup 1998 server farm trace. In these services as well as the World Cup server farm, there are a few objects that are very popular while there are many more that are only accessed a few times. In the subscription services, there are certain news articles, images, and news videos that are the most popular while other objects are accessed relatively few times. The same is true for a digital library in which the latest articles are probably the most popular, and with a few exceptions, the older articles are accessed relatively few times. And with the online store services, there are certain items that are very popular while others are bought relatively infrequently.

The similarity of these service request patterns to the World Cup trace suggests that the services could be deployed using distributed server farms. A central set of servers would host the entire site while the remote servers could cache the most popular requests of the central servers thereby offloading some of the workload from the central servers. These remote servers would then use the GDSHits or LFU-DA cache replacement algorithms to determine what objects they should keep in their cache. It would be interesting to work with one or more of these service companies to determine how caching could improve their delivery techniques.

The work from this study will help bring universities, corporations, and individuals closer to realizing just-in-time distance learning and general video viewing across the entire Internet. By being able to predict the expected demand for multimedia objects and caching objects at strategically positioned proxy servers, the bandwidth usage on the organizations' networks will be reduced, thereby allowing more bandwidth to serve more users. Also, findings from this study can also impact the efficiency of distributed databases and other distributed information systems.

LIST OF REFERENCES

[ABCdO96] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of PDIS'96: IEEE International Conference in Parallel and Distributed Information Systems*, Miami Beach, FL, December 1996. Available at: http://www.cs.bu.edu/groups/oceans/papers/Home.html.

[ACD+99] Martin Arlitt, Ludmila Cherkasova, John Dilley, Richard Friedrich, and Tai Jin. Evaluating content management techniques for Web proxy caches. Technical Report HPL-1999-173, Hewlett Packard Labs, Palo Alto, CA, April 1999. Available at: http://www.hpl.hp.com/techreports/98/HPL-98-173.html.

[AFJ99a] Martin Arlitt, Rich Friedrich, and Tai Jin. Workload characterization of a Web proxy in a cable modem environment. Technical Report HPL-1999-48, HP Laboratories Palo Alto, Internet Systems and Applications Laboratory, April 1999. Available at: http://www.hpl.hp.com/techreports/.

[AFJ99b] Martin Arlitt, Richard Friedrich, and Tai Jin. Performance elvaluation of Web proxy cache replacement policies. Technical Report HPL-98-97R1, HP Laboratories Palo Alto, Internet Systems and Applications Laboratory, November 1999. Available at: http://www.hpl.hp.com/techreports/.

[AJ99] Martin Arlitt and Tai Jin. Workload characterization of the 1998 World Cup Web site. Technical Report HPL-1999-35, HP Laboratories Palo Alto, Internet Systems and Applications Laboratory, February 1999. Available at: http://www.hpl.hp.com/techreports/.

[AK96] Palmer W. Agnew and Anne S. Kellerman. *Distributed Multimedia: Technologies, Applications, and Opportunities in the Digital Information Industry.* Addison Wesley, 1996.

[Aka99a] Apple and Akamai create high quality network for Internet streaming. Online press release from Akamai Technologies available at: http://www.akamai.com/news/press1977.html, 1999. Posted July 21, 1999.

[Aka99b] Akamai teams with RealNetworks for streaming media delivery. Online press release from Akamai Technologies available at: http://www.akamai.com/news/press4110.html, 1999. Posted September 17, 1999.

[AOG92] David P. Anderson, Yoshitomo Osawa, and Ramesh Govindan. A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):331–337, November 1992.

[ASA$^+$95] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, pages 119–133, Boston, December 1995. O'Reilly. Available at: http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html.

[AW96] M. F. Arlitt and C. L. Williamson. Web server workload characteristics: The search for invariants. In *Proceedings of SIGMETRICS 96*, pages 126–137, Philadelphia, PA, 1996. ACM.

[AW97] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[Ban95] Mohana Krishna Bandaru. Interactive digital multimedia delivery for personal learning. Master's thesis, School of Electrical Engineering, Purdue University, 1995.

[BCF$^+$99] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Infocom'99 Proceedings*. IEEE, IEEE Press, March 1999. Available at: http://www.cs.wisc.edu/~cao/papers/zipf-implications.html.

[Bes97] Azer Bestavros. WWW traffic reduction and load balancing through server-based caching. *IEEE Concurrency*, 5(1):56–67, January–March 1997.

[BH96] Jean-Chrysostome Bolot and Philipp Hoschka. Performance engineering of the World Wide Web: Application to dimensioning and cache design. In *Proceedings of the Fifth International World Wide Web Conference*, Amsterdam, 1996. Elsevier Press. Available at: http://www5conf.inria.fr/fich_html/papers/P44/Overview.html.

[Bho95] Sudeep Bhoja. Interactive multimedia delivery for distance education. Master's thesis, School of Electrical Engineering, Purdue University, 1995.

[CD85] Hong-Tai Chou and David J. DeWitt. An evaluation of buffer management strategies for relational database systems. In A. Pirotte and Y. Vassiliou, editors, *Proceedings of the 11th Conference on Very Large Data Bases*, pages 127–141, Stockholm, August 1985.

[CDN$^+$96] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proceedings of 1996 Usenix Technical Conference*, pages 153–163, San Diego, CA, January 1996. Usenix.

[CGM97] Edward Chang and Hector Garcia-Molina. Reducing initial latency in media servers. *IEEE Multimedia*, 5(3):50–61, July–September 1997.

[CHK$^+$96] Kenneth K. Chan, Cyrus C. Hay, John R. Keller, Gordon P. Kurpanek, Francis X. Schumacher, and Jason Zheng. Design of the HP PA 7200 CPU. *Hewlett-Packard Journal*, 47(1):25–33, February 1996.

[CI97] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997.

[CLR90] Thomas H. Cormen, Charles E. Lieserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, New York, 1990.

[CZB98] Pei Cao, Jin Zhang, and Kevin Beach. Active cache: Caching dynamic contents on the Web. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 373–388, 1998.

[DAP99] John Dilley, Martin Arlitt, and Stephane Perret. Enhancement and validation of Squid's cache replacement policy. Technical Report HPL-1999-69, Hewlett Packard Labs, Palo Alto, CA, May 1999. Available at: http://www.hpl.hp.com/techreports/1999/HPL-1999-69.html.

[DDM$^+$95] Asit Dan, Daniel M. Dias, Rajat Mukherjee, Dinkar Sitaram, and Renu Tewari. Buffering and caching in large-scale video servers. In *Digest of Papers of IEEE Compcon Spring 95*, pages 217–224, Piscataway, New Jersey, March 1995. IEEE.

[Den68] P. J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, May 1968.

[DFJR96] John A. Dilley, Richard J. Friedrich, Tai Y. Jin, and Jerome Wide Rolia. Measurement tools and modeling techniques for evaluating Web server performance. Technical Report HPL-96-161, Hewlett Packard Labs, Palo Alto, CA, December 1996. Available at: http://www.hpl.hp.com/techreports/96/HPL-96-161.html.

[DHS93] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A case for caching file objects inside internetworks. Technical Report CU-CS-642-93, Dept. of Computer Science, University of Colorado, Boulder, Colorado, March 1993.

[DS78] Peter J. Denning and Donald R. Slutz. Generalized working sets for segment reference strings. *Communications of the ACM*, 21(9):750–759, September 1978.

[DSK94] Jayanta K. Dey, Chia-Shiang Shih, , and Manoj Kumar. Storage subsystem design in a large multimedia server for high-speed network environments. In *Proceedings of IS&T/SPIE Symposium on Electronic Imaging: in Science & Technology, Workshop on High-Speed Networking and Multimedia Computing*, pages 200–211, Bellingham, Wash, February 1994. SPIE.

[DT94] Yurdaer N. Doğanata and Asser N. Tantawi. Making a cost-effective video server. *IEEE Multimedia*, 1(4):22–30, Winter 1994.

[EH84] Wolfgang Effelsberg and Theo Haerder. Principles of database buffer management. *ACM Transactions on Database Systems*, 9(4):560–595, December 1984.

[GC92] Jim Gemmell and Stavros Christodoulakis. Principles of delay-sensitive multimedia data storage and retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability.* Freeman, New York, 1979.

[Gla94] Steven Glassman. A caching relay for the World Wide Web. In *Proceedings of the First International World Wide Web Conference*, pages 69–76, North-Holland, Amsterdam, 1994.

[Gro99] Niel Gross. The net's next battle royal – video: The technology isn't there, but the competition is. *BusinessWeek*, pages 108–109, June 28, 1999.

[GS95] James S. Gwertzman and Margo Seltzer. The case for geographical push-caching. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 51–55. IEEE CS Press, 1995.

[GS96a] James Gwertzman and Margo Seltzer. An analysis of geographical push-caching. Available at: http://www.eecs.harvard.edu/~vino/web/icdcs.ps, 1996.

[GS96b] James Gwertzman and Margo Seltzer. World-Wide Web cache consistency. In *Proceedings of 1996 Usenix Technical Conference*, pages 141–151, San Diego, CA, January 1996. Usenix.

[GVK$^+$95] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Lawrence A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[Has93] Roger L. Haskin. The Shark continuous-media file server. In *Proceedings of IEEE 1993 Spring COMPCON*, pages 12–17, San Francisco, CA, 1993. IEEE.

[Inf99] Infolibria and Avid Technology introduce Internet's most powerful streaming media system using Microsoft Windows Media. Online press release from InfoLibria available at http://www.infolibria.com/news/pressrel1999-1011b.html, 1999. Posted October 11, 1999.

[Ink99] Inktomi announces Traffic Server^{tm} 3.0: First open network cache platform enables value-added services through robust APIs – six new software providers announce support for platform. Online press release from Inktomi available at http://www.inktomi.com/new/press/ts3.html, 1999. Posted April 26, 1999.

[Jou90] Norman Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, Seattle, June 1990. ACM.

[KCS94] Dilip D. Kandlur, Mon-Song Chen, and Zon-Yin Shae. Design of a multimedia storage server. In *Proceedings of IS&T/SPIE Symposium on Electronic Imaging: in Science & Technology, Workshop on High-Speed Networking and Multimedia Computing*, pages 164–178, Bellingham, Wash., February 1994. SPIE.

[KLW94] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *IEEE Computer*, 27(3):38–46, March 1994.

[KMR95] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. NCSA's World Wide Web server: Design and performance. *IEEE Computer*, 28(11):68–74, November 1995.

[KRT95] Mohan Kamath, Krithi Ramamritham, and Don Towsley. Continuous media sharing in multimedia database systems. In Tok Wang Ling and Yoshifumi Masunaga, editors, *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications (DASFAA '95)*. World Scientific Publishing Co., April 1995.

[KSW98] Michal Kurcewicz, Wojtek Sylwestrzak, and Adam Wierzbicki. A filtering algorithm for Web caches. *Computer Networks and ISDN Systems*, 30(22–23):2203–2209, November 1998.

[LA94] Ari Luotonen and Kevin Altis. World-Wide Web proxies. In *First International Conference on the WWW*, Geneva, May 1994. Also appeared in Computer Networks and ISDN Systems 27, No. 2, 1994. Available at: http://www1.cern.ch/PapersWWW94/luotonen.ps.

[Lee98] Jack Y.B. Lee. Parallel video servers: A tutorial. *IEEE Multimedia*, 6(2):20–28, April–June 1998.

[LRB82] D. H. Lawrie, J. M. Randal, and R. R. Barton. Experiments with automatic file migration. *IEEE Computer*, pages 45–55, July 1982.

[LRV97] Paolo Lorenzetti, Luigi Rizzo, and Lorenzo Vicisano. Replacement policies for a proxy cache. Technical report, Department of Information Engineering, University of Pisa, 1997. Available at: http://www.iet.unipi.it/luigi/research.html.

[LS93] Philip Lougher and Doug Shepherd. The design of a storage server for continuous media. *The Computer Journal*, 36(1):32–42, 1993.

[LYW91] A. Leff, P. Yu, and J. Wolf. Policies for efficient memory utilization in a remote caching architecture. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 198–207, December 1991.

[Mar96] E. Markatos. Main memory caching of Web documents. *Computer Networks and ISDN Systems*, 28(11):893–905, May 1996.

[Mat99] Novell preps QTV caching server. Online news article on Macweek.com available at http://macweek.zdnet.com/1999/11/07/novell.html, 1999. Posted Tuesday, November 9, 1999.

[MBN96] David G. Meyer, M. Krishna Bandaru, and Christopher C. Niessen. The *VJ instructional multimedia testbed systems: Recent experiences in development and utilization. In *1996 Frontiers in Education Conference Proceedings*, Salt Lake City, Utah, November 1996.

[MH92] Dan Muntz and Peter Honeyman. Multi-level caching in distributed file systems or your cache ain't nuthing but trash. In *Proceedings of Winter 1992 Usenix*, pages 305–313. Usenix Assoc., 1992.

[Mim00] MIME (multi-purpose internet mail extensions). Online document available at http://www.whatis.com/mime.htm, 2000.

[MK94] David G. Meyer and Richard A. Krzyzkowski. Experience using the VideoJockey system for instructional multimedia delivery. In *Proceedings of the 1994 Frontiers in Education Conference*, pages 262–266, San Jose, California, November 1994.

[MLB95] Radhika Malpani, Jacob Lorch, and David Berger. Making World Wide Web caching servers cooperate. In *Proceedings of the Fourth International World Wide Web Conference*, pages 107–117, Boston, December 1995. O'Reilly. Available at: http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html.

[MNR97] David G. Meyer, Christopher C. Niessen, and Albert I. Reuther. Experimental multimedia-delivered course formats. In *Proceedings of*

1997 Frontiers in Education Conference, Pittsburgh, Pennsylvania, November 1997. Paper No. 97-1197 (published on CD-ROM).

[NJT93] Ciro A. Noronha Jr. and Fouad A. Tobagi. The evolution of campus networks toward multimedia. In *Proceedings of IEEE Compcon Spring 93*, pages 49–58, Los Alamitos, California, 1993. IEEE, CS Press.

[NS95] Klara Nahrstedt and Ralf Steinmetz. Resource management in networked multimedia systems. *IEEE Computer*, 28(5):52–63, May 1995.

[NY94] Raymond T. Ng and Jinhai Yang. An analysis of buffer sharing and prefetching techniques for multimedia systems. Technical report, Department of Computer Science, University of British Columbia, Vancouver, B.C., 1994.

[OBRS95] Banu Özden, Alexandros Biliris, Rajeev Rastogi, and Avi Silberschatz. A disk-based storage architecture for movie on demand servers. *Information Systems*, 20(6):465–482, 1995. Available on the Fellini home page: http://www.bell-labs.com/project/fellini/papers.html.

[OnC99a] On Command Corporation. Online document available from http://www.oncommand.com/, 1999.

[OnC99b] On Command announces commercial deployment of breakthrough OCX[tm] in-room entertainment platform for hotels. Online press release available at http://www.oncommand.com/ocxplatform.htm, 1999.

[ORS96] Banu Özden, Rajeev Rastogi, and Avi Silberschatz. Buffer replacement algorithms for multimedia databases. In *IEEE International Conference on Multimedia Computing and Systems*, June 1996. Available on the Fellini home page: http://www.bell-labs.com/project/fellini/papers.html.

[Pap91] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw Hill, Inc., New York, third edition edition, 1991.

[PF76] B. G. Prieve and R. S. Fabry. VMIN—an optimal variable space page-replacement algorithm. *Communications of the ACM*, 19(5):295–297, May 1976.

[PR94] James E. Pitkow and Margaret M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proceedings of the Second International WWW Conference*, pages 1039–1046, Chicago, 1994.

[PR97] Christos Papadimitriou and P. Venkat Rangan. System and method for selecting cache server based on transmission and storage factors for efficient delivery of multimedia information in a hierarchical network of servers. U.S. Patent number 5,592,626, January 1997.

[PRR94] Christos H. Papadimitriou, Srinivas Ramanathan, and P. Venkat Rangan. Information caching for delivery of personalized video programs on home entertainment channels. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 214–223. IEEE CS Press, 1994.

[Ran96] P. Venkat Rangan. System for efficient delivery of multimedia information using hierarchical network of servers selectively caching program for a selected time period. U.S. Patent number 5,583,994, December 1996.

[RD90] John T. Robinson and Murthy V. Devarakonda. Data cache management using frequency-based replacement. In *Proceedings of 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 18, pages 134–142, Boulder, Colorado, May 1990. Performance Evaluation Review.

[Red97] A. L. Narasimha Reddy. Caching strategies for a multimedia server. In *Proc. IEEE Int. Conf. on Multimedia Computing and Systems*, June 1997.

[Reu96] Albert I. Reuther. Analysis of educational multimedia delivery: Current and future testbed systems. Master's thesis, School of Electrical Engineering, Purdue University, 1996.

[RM97] Albert I. Reuther and David G. Meyer. Analysis of daily student usage of an educational multimedia system. In *Proceedings of 1997 Frontiers in Education Conference*, Pittsburgh, Pennsylvania, November 1997. Paper No. 97-1203 (published on CD-ROM).

[RR94] Srinivas Ramanathan and P. Venkat Rangan. Architectures for personalized multimedia. *IEEE Multimedia*, 1(1):37–46, January–March 1994.

[RV93] P. Venkat Rangan and Harrick M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):564–573, August 1993.

[Sat90] Mahadev Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5):9–21, May 1990.

[SC98] Chutimet Srinilta and Alok Choudhary. Performance enhancement using intra-server caching in a continuous media server. In *Eighth International Workshop on Research Issues in Data Engineering: Continuous-Media Databases and Applications*, Florida, USA, February 1998.

[Sel88] Timos K. Sellis. Intelligent caching and indexing techniques for relational database systems. *Information Systems*, 13(2):175–185, 1988.

[SG98] Abraham Silberschatz and Peter Galvin. *Operating System Concepts.* Addison Wesley, Reading, Massachusetts, 5th edition, 1998.

[SKK$^+$90] Mahadev Styanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Streere. Coda: A highly available file system for distributed workstation environments. *IEEE Transactions on Computers*, 39(4), April 1990.

[Smi81] Alan Jay Smith. Long term file migration: Development and evaluation of algorithms. *Communications of the ACM*, pages 521–532, August 1981.

[Smi82] Alan Jay Smith. Cache memories. *Computing Surveys*, 14(3):473–530, September 1982.

[SS82] G. M. Sacco and M. Schkolnick. A mechanism for managing the buffer pool in a relational database system using the hot set model. In *Proceedings of the 8th Conference on Very Large Data Bases*, pages 257–262, Mexico City, September 1982.

[SSV97] Peter Scheuermann, Junho Shim, and Radek Vingralek. A case for delay-conscious caching of Web documents. *Computer Networks and ISDN Systems*, 29:997–1005, 1997. Also in Proceedings of the Sixth International World Wide Web Conference.

[SSV99] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, July/August 1999.

[Ste90] Per Stenström. A survey of cache coherence schemes for multiprocessors. *IEEE Computer*, 23(6):12–24, June 1990.

[Tob95] Fouad A. Tobagi. Distance learning with digital video. *IEEE Multimedia*, 2(1):90–93, Spring 1995.

[TP93] Fouad A. Tobagi and Joseph Pang. StarWorks—a video applications server. In *Proceedings of IEEE Compcon Spring 93*, pages 4–11, Los Alamitos, California, 1993. IEEE, CS Press.

[TPBG93] Fouad A. Tobagi, Joseph Pang, Randall Baird, and Mark Gang. Streaming RAID: A disk array management system for video files. In *Proceedings of the First International Conference on Multimedia*, pages 393–400, New York, 1993.

[TVDS98] Renu Tewari, Harrick M. Vin, Asit Dan, and Dinkar Sitaram. Resource-based caching for Web servers. In *Proceedings of ACM/SPIE Multimedia Computing and Networking 1998 (MMCN'98)*, pages 191–204, San Jose, January 1998.

[WA97] Roland P. Wooster and Marc Abrams. Proxy caching the estimates page load delays. *Computer Networks and ISDN Systems*, 29:977–986, 1997. Also in Proceedings of the Sixth International World Wide Web Conference.

[WAS+96] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for World-Wide Web documents. In *Proceedings of Sigcomm96 Conference*. ACM, August 1996.

[Wes96] Duane Wessels. Squid Internet object cache. Online document available from http://squid.nlanr.net/Squid, May 1996.

[Wil65] Maurice Wilkes. Slave memories and dynamic storage allocation. *IEEE Transactions on Electronic Computers*, EC-14(2):270–271, April 1965.

[You94] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.

[Zip49] George K. Zipf. *Human Behaviour and the principles of Least Effort.* Addison-Wesley, Reading MA, 1949.

VITA

ALBERT I. REUTHER

EDUCATION

Purdue University, West Lafayette, Indiana, 1989-2000

Ph.D. in Electrical and Computer Engineering, July 2000, Area of Specialization: Computer Architectures.

M.S. in Electrical Engineering, 1996.

B.S. in Computer and Electrical Engineering, 1994, with Highest Distinction and Co-op Certificate in Engineering.

AWARDS AND HONORS

Intel Foundation Fellowship, 1999-2000.

Purdue Andrews Graduate Fellowship, 1994-1996.

US Air Force Graduate Fellowship (US D.O.D.), 1994. (Funding canceled in US Congress.)

National Science Foundation Graduate Fellowship Honorable Mention, 1995.

Eta Kappa Nu, inducted 1993.

Golden Key National Honor Society, inducted 1993.

Tau Beta Pi, inducted 1992.

RESEARCH

Doctoral Research: School of Electrical and Computer Engineering, Purdue University, 1995-2000 (research advisor: Prof. David G. Meyer).

- Investigation of higher complexity cache replacement algorithms for use in Web-based and video cache proxy servers.

- Development of storage recycling data replacement algorithms for data forwarding problem in networked systems of computers.
- Analysis of student usage patterns of educational multimedia testbed delivery systems.
- Exploration of the effect of personality types on the usage of a multimedia engineering education system.
- Study on task mapping algorithms in heterogeneous computing environments (with Prof. Howard J. Siegel).

Corporate Research: Research Engineering in the Internet/Applications Systems Lab, Hewlett-Packard Corporation, Roseville, California, 1999.

- Researched distributed system management techniques for a next generation web server system.
- Analyzed performance characteristics of high-performance web server applications and their interactions.

Corporate Research: Engineering Research and Development Department, Delco Chassis Division, General Motors Corporation, Dayton, Ohio, 1992-1993.

- Created and developed user-friendly graphical application software that interfaced with experimental microprocessor-based antilock brake system controller and tire inflation monitor system.
- Researched and developed helium leak test station and torque-controlled screw driver station for experimental motor actuator assembly. First use of helium leak detection at plant.

EXPERIENCE

Engineering Intern: Research Engineering, Internet/Applications Systems Lab, Hewlett- Packard, Corporation, Roseville, California, May 1999 to August 1999.

Researched remote management software and analyzed performance of next generation web servers.

Engineering Intern: Firmware Group, General Systems Lab, Hewlett-Packard, Corporation, Roseville, California, May 1997 to July 1997.

Co-developed firmware simulator design for highly parallel computer server and implemented reusable I/O modules for firmware simulator.

Research Assistant: Digital Systems Laboratory/Multimedia Learning Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, August 1995 to July 2000.

Analysis of student usage patterns of educational multimedia testbed delivery systems to aid in developing and improving educational multimedia delivery systems.

Teaching Assistant: Undergraduate Counseling Office, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, August 1994 to July 2000.

Provide undergraduate electrical engineering students with course, schedule, and career counseling. Lead tours of School of Electrical and Computer Engineering for prospective students.

Engineering Co-op Student: Delco Chassis Division, General Motors Corporation, Dayton, Ohio, May 1990 to May 1993.

Various positions in product research and development, manufacturing research, plant engineering, project management, and laboratory validation engineering.

PUBLICATIONS AND PRESENTATIONS

Publications

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund, A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, Technical Report, School of Electrical and Computer Engineering, Purdue University, March 2000, TR-ECE 00-4, 57 pp.

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys , Bin Yao, Debra Hensgen, and Richard F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," 8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, April 1999, pp. 15–29.

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, and Bin Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," IEEE Workshop on Advances in Parallel and Distributed Systems, West Lafayette, IN, October 1998, pp. 330–335, (included in the Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, 1998).

Albert I. Reuther and David G. Meyer, "Analysis of Daily Student Usage of an Educational Multimedia System," 1997 Frontiers in Education Conference Proceedings, Pittsburgh, Pennsylvania, November 1997, Paper No. 97-1203 (published on CD-ROM).

David G. Meyer, Christopher C. Niessen, and Albert I. Reuther, "Experimental Multimedia-Delivered Course Formats," 1997 Frontiers in Education Conference Proceedings, Pittsburgh, Pennsylvania, November 1997, Paper No. 97-1197 (published on CD-ROM).

Albert I. Reuther and David G. Meyer, "The AVJ Instructional Multimedia Testbed System: Analysis of Student Usage Patterns," 1997 ASEE Illinois/Indiana Sectional Conference Proceedings, Indianapolis, Indiana, March 1997, pp. 165-168.

Albert I. Reuther, Analysis of Educational Multimedia Delivery: Current and

Future Testbed Systems, Master's Thesis, School of Electrical Engineering, Purdue University, 1996.

Manuscripts in Preparation

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," submitted to the Journal of Parallel and Distributed Computing for a Special Issue on Software Support for Distributed Computing.

Albert I. Reuther and David G. Meyer, "Cache Replacement Algorithms and Admission Policies for Internet Cache Proxy Servers", work in progress for refereed journal submission.

Albert I. Reuther and David G. Meyer, "Cache Replacement Algorithms for Digital Video Cache Proxy Servers", work in progress for refereed journal submission.

Albert I. Reuther and David G. Meyer, "Analysis of Student Usage of an Educational Multimedia System", work in progress for refereed journal submission.

Albert I. Reuther and David G. Meyer, "The Effect of Personality Type on the Usage of a Multimedia Engineering Education System", work in progress for refereed journal submission.

Stacey Engel, Jeanine Friedrich, Albert I. Reuther, and David G. Meyer, "Analysis of Student Response to Asynchronous Learning with Technology," work in progress for refereed journal submission.

David G. Meyer, Ku Jei King, Christopher C. Niessen, and Albert I. Reuther, "The DVJ2 Educational Multimedia Delivery System", work in progress for refereed conference proceedings submission.

Posters and Presentations

Albert Reuther and David G. Meyer, Proxy Cache Servers in the Wireless Market, Poster for Purdue Electrical and Computer Engineering Industry Workshop, Purdue University, West Lafayette, Indiana, April 2000. Won second place in poster competition.

Albert Reuther, V830R/AV: Embedded Multimedia Superscalar RISC Processor, paper by Kazumasa Suzuki, Tomohisa Arai, Kouhei Nadehara, and Ichiro Kuroda, presentation given for Computer Architecture Seminar at Purdue, February 17, 2000.

Tracy Braun, Noah Beck, Ladislau Boloni, Albert Reuther, James Robertson, Mitchell Theys, Bin Yao, H. J. Siegel, Richard Freund, Debra Hensgen, and Muthucumaru Maheswaran, Static Mapping Heuristics for Meta-Tasks in Distributed Heterogeneous Computing Systems, Poster for Purdue Electrical and Computer Engineering Industry Institute, Purdue University, West Lafayette, Indiana, March 1999.

Albert Reuther, Memory-System Design Considerations for Dynamically-Scheduled Processors, paper by Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic, presentation given for Computer Architecture Seminar at Purdue, November 12, 1998.

Albert Reuther, Christopher Niessen, Ku-Jei King, and David G. Meyer, The *VJ Instructional Research Multimedia Testbed, Poster for Purdue Electrical and Computer Engineering Industry Institute, Purdue University, West Lafayette, Indiana, April 1998.

David G. Meyer, Christopher Niessen, Albert Reuther, and Ku-Jei King, Technology- Delivered Education: Some Questions and Answers, Workshop given at Teaching, Learning, and Technology Showcase, Purdue University, West Lafayette, Indiana, March 1998.

Albert Reuther, Christopher Niessen, Ku-Jei King, and David G. Meyer, The *VJ Educational Multimedia Delivery Systems, Poster for Purdue Electrical

Engineering Industry Institute, Purdue University, West Lafayette, Indiana, April 1997.

PROFESSIONAL SOCIETIES

Institute of Electrical and Electronics Engineers (I.E.E.E.) and I.E.E.E. Computer Society, , the Association of Computing Machinery (A.C.M.), Eta Kappa Nu Electrical and Computer Engineering Honor Society, and Tau Beta Pi Engineering Honor Society.

OTHER ACTIVITIES

Eta Kappa Nu: Workshop Chairman, Fall 1995 Fall 1997 and Fall 1998; Alumni Relations Committee Chairman, Fall 1997 Summer 2000.

Purdue University Marching Band: Fall 1989 Fall 1993; Rank Leader, Fall 1992; Assistant Section Leader, Fall 1993.

Purdue University Jazz Band: Fall 1994 Summer 2000.

Purdue Triathlon Club: Spring 1999 Summer 2000.

Worldwide Discipleship Association: Treasurer, Fall 1999 Spring 2000; Publicity Committee Fall 1994 Spring 2000.

Purdue Christian Campus House: Worship Band member, bass guitar.