TOWARDS HIGH PERFORMANCE PERSONALIZED VIDEO CONTENT DISSEMINATION ARCHITECTURE FOR MOBILE DEVICES

by

PIYUSH PARATE

(Under the Direction of Lakshmish Ramaswamy)

ABSTRACT

Video streaming on mobile devices such as PDA's, laptop PCs, pocket PCs and cell phones is becoming increasingly popular. These mobile devices are typically constrained by their battery capacity, bandwidth, screen resolution and video decoding and rendering capability. Consequently, video personalization strategies are used to provide these resource-constrained mobile devices with personalized video content that is most relevant to the client's request while simultaneously satisfying the client's resource constraints. Since mobile clients are typically in remote network location from the personalizing servers, it is often desirable to cache intelligently portions of the video files at the caching proxies in order to reduce latency observed at the client end, and in the process also offload the data load on the server to local caching proxies. This research proposes design and implementation of video personalization server and multiple caching proxies, which can efficiently disseminate personalized videos to multiple resource-constrained clients. The video personalization servers use an automatic video segmentation and video indexing scheme based on semantic video content. The caching proxies implement a novel cache replacement and Multi-stage client request aggregation algorithm, specifically suited for caching personalized video files generated by the personalization servers. The cache design also

implements a personalized video segment calculation algorithm based on client's content preference and resource constraints. The video personalizing server and cache architecture is well suited for personalized video dissemination, with low client-experienced latency, to resource-constrained multimedia-enabled mobile devices such as mobile phones, PDAs and pocket PCs.

KEYWORDS: Video Personalization, Caching, Cache Replacement, Request Aggregation

TOWARDS HIGH PERFORMANCE PERSONALIZED VIDEO CONTENT DISSEMINATION ARCHITECTURE FOR MOBILE DEVICES

by

PIYUSH PARATE

BTECH, National Institute of Technology Bhopal India, 2005

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2009

TOWARDS HIGH PERFORMANCE PERSONALIZED VIDEO CONTENT

DISSEMINATION ARCHITECTURE FOR MOBILE DEVICES

by

PIYUSH PARATE

Major Professor:	Lakshmish Ramaswamy
Committee:	Suchendra M. Bhandarkar
	Kang Li

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2009

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Recent proliferation of mobile computing devices and networking technologies has created enormous opportunities for mobile device users to communicate with multimedia servers. As handheld mobile computing and communication devices such as personal digital assistants (PDAs), pocket-PCs and cellular devices have become increasingly capable of storing, rendering and display of multimedia data, the user demand for being able to view streaming video on such devices has grown considerably. For example, a mobile handheld client may be interested in viewing a video showing sports highlights or weather forecast video for his/her travel destination. One of the natural limitations of typical handheld mobile devices is that they are resource constrained, i.e., constrained by their battery capacity, screen resolution, video decoding and rendering capability, and, in many situations, by the available network bandwidth due to their mobility. Thus, the original video content often needs to be personalized in order to fulfill the client's request while simultaneously satisfying various client-side and system-level resource constraints. Numerous video personalization strategies have been developed [14],[19],[20] in order to provide these resource-constrained devices with personalized video content that is most relevant to the client's request and the available client-side and network resources.

Since mobile clients are typically in remote network location from the personalizing servers, it is often desirable to intelligently cache portions of the video files at intermediate

caching proxies. This would not only reduce latency observed at the client end, but would also enable local caches to share the loads on origin servers thereby enhancing the system scalability. This research presents the design and implementation of a novel framework for a scalable disseminating personalized video content to a large number of resource constrained client-devices. The framework is characterized by the following two unique features:

I. Video personalization server (VPS) [23],[24] which performs automatic video segmentation and video indexing based on video content semantics.

II. Multiple caching proxies which calculate segments for personalized video content based on the client's content preferences and resource constraints using a Multiple-Choice Multi-Dimensional Knapsack Problem [1],[9] (MMKP)-based video personalization strategy. They perform Multi-stage client request aggregation algorithm. The caching proxies also incorporate a novel cache replacement policy that is sensitive to the existence of different personalized versions of same underlying video data.

The proposed system is evaluated through series of experiments. The results show the implemented techniques and mechanisms yield substantial performance benefits.

Figure 1 depicts the high level architecture of the personalized video dissemination framework. The environment consists of one or more *origin servers* that host video content and a number of *proxy caches*. Each *proxy cache* is responsible for clients in a certain geographical area in the sense that their requests to video content first reach the proxy cache. Upon receiving a client request, the proxy cache obtains respective video segments from the origin servers (if segments are not already available at the cache) composes the video stream and serves the client.

Communication between clients and the caching proxy typically occurs in two phases, wherein the client first asks for the content to be generated and then downloads the content.

Since the actual download occurs in the second phase of communication, the caching proxy serving as an intermediary between the clients and the VPS(s), can *learn* important statistics about the requests in order to pre-customize the video to be disseminated based on the download pattern that is to follow the first phase of communications.

The framework supports multiple levels of membership such as "paying" and "non-paying". The cache design uses a content-aware video re-encoding algorithm termed as Features, Motion and Object Enhanced Multi-Resolution FMOE-MR video encoding [2], in order to create two versions of the original video file of lower visual quality and correspondingly lower file size. These versions are useful in delivering videos of different visual quality to different clients depending on their levels of membership. Before the client request is processed to generate the content response, the cache performs semantic aggregation of the client requests; further improving the cache replacement policy and hence reducing the average client latency.

As the caching proxies serve clients based on a two-phase communication, and has the ability to generate videos of different visual quality to serve multiple clients with different levels of membership, it performs significantly better than standard caching techniques.

In the next section, the design of VPS is described, following which, we discuss the cache design and implementation of proxy caches.

CHAPTER 2

VIDEO PERSONALIZATION SERVER

The videos hosted by the video personalization server (VPS) are first segmented and indexed. The videos are then transcoded at multiple levels of abstraction based on their content. This allows for video personalization based on clients' preferences and resource constraints.

2.1 Video Segmentation and Indexing

A stochastic multi-level Hidden Markov Model (HMM)-based algorithm is used for video segmentation and indexing wherein the input video stream is classified frame by frame into semantic units [1]. A semantic unit within a video stream is a video segment that can be associated with a clear semantic meaning or concept, and consists of a concatenation of semantically and temporally related video shots or video scenes. Instead of detecting video shots or scenes, it is often much more useful to recognize semantic units within a video stream to be able to support video retrieval based on high-level semantic content. Note that visually similar video shots or video scenes may be contained within unrelated semantic units. Thus, video retrieval based purely on detection of video shots or video scenes will not necessarily reflect the semantic content of the video stream.

The semantic units within a video stream can be spliced together to form a logical video sequence that the viewer can understand. In well organized videos, such as TV broadcast news and sports programs, the video can be viewed as a sequence of semantic units that are

concatenated based on predefined video program syntax. Parsing a video file into semantic units enables video retrieval based on high-level semantic content and playback of logically coherent blocks within a video stream. Automatic indexing of semantic components within a video stream can enable a viewer to jump straight to points of interest within the indexed video stream, or even skip advertisement breaks during video playback.

In the proposed scheme, a video stream is modeled at both, the semantic unit level and the program model level. For each video semantic unit, an HMM is generated to model the stochastic behavior of the sequence of feature emissions from the image frames. Each image frame in a video stream is characterized by a multi-dimensional feature vector. A video stream is considered to generate a sequence of these feature vectors based on an underlying stochastic process that is modeled by a multi-level HMM.

In this work, two categories of features are extracted from each image frame in the video stream. The first category includes a set of simple features. The dynamic characteristics of the image frames comprising the video stream are captured by the differences of successive image frames at both, the pixel level and the histogram level. Various motion-based measures describing the movement of the objects in the image frames are used, including the motion centroid of the image, and intensity of motion. Measures of illumination change at both, the pixel level and the histogram level are also included in the multi-dimensional feature vector. Definitions of these features are given in [15]. In the second feature category, Tamura features [7] are used to capture the textural characteristics of the image frames at the level of human perception. Tamura contrast, Tamura coarseness and Tamura directionality have been used successfully in content-based image retrieval [7]. In this work, inclusion of these features helps to improve the accuracy of temporal video segmentation and video indexing.

The optimal HMM parameters for each semantic unit are learned from the feature vector sequences obtained from the training video data. The HMMs for individual semantic units are trained separately using the training feature vector sequences. This allows for modularity in the learning procedure and flexibility in terms of being able to accommodate various types of video data. The current scheme uses a universal left-to-right HMM topology, i.e., an HMM topology where no backward state transitions are allowed, with continuous observations of the feature vector emissions. The distribution of the feature vector emissions in the HMM is approximated by a mixture of Gaussian distributions.

The search space for the implemented single-pass video segmentation and video indexing procedure is characterized by the concatenation of the HMMs corresponding to the individual semantic units. The HMM corresponding to an individual semantic unit essentially models the stochastic behavior of the sequence of image features within the scope of that semantic unit. Transitions amongst these semantic unit HMMs are regulated by a pre-specified video program model. The Viterbi algorithm is used to determine the optimal path in the concatenation of the HMMs in order to segment and index video stream in a single pass [1].

CHAPTER 3

VIDEO PERSONALIZATION CACHE

In the previous section, we have detailed the design of a video personalization server (VPS) that receives requests from multiple clients, and creates personalized videos for them. The cache serves as an intermediary between the clients and the VPS (**Figure 1**). In the following subsections, we first detail the design of the proposed cache. Then, we describe the working of multi-stage client request aggregation in detail. Next, we describe how the proposed cache creates different versions of the video files using content-aware video processing. Finally, we describe in detail the protocol followed by the caching proxies and clients in order to synchronize with the VPS.

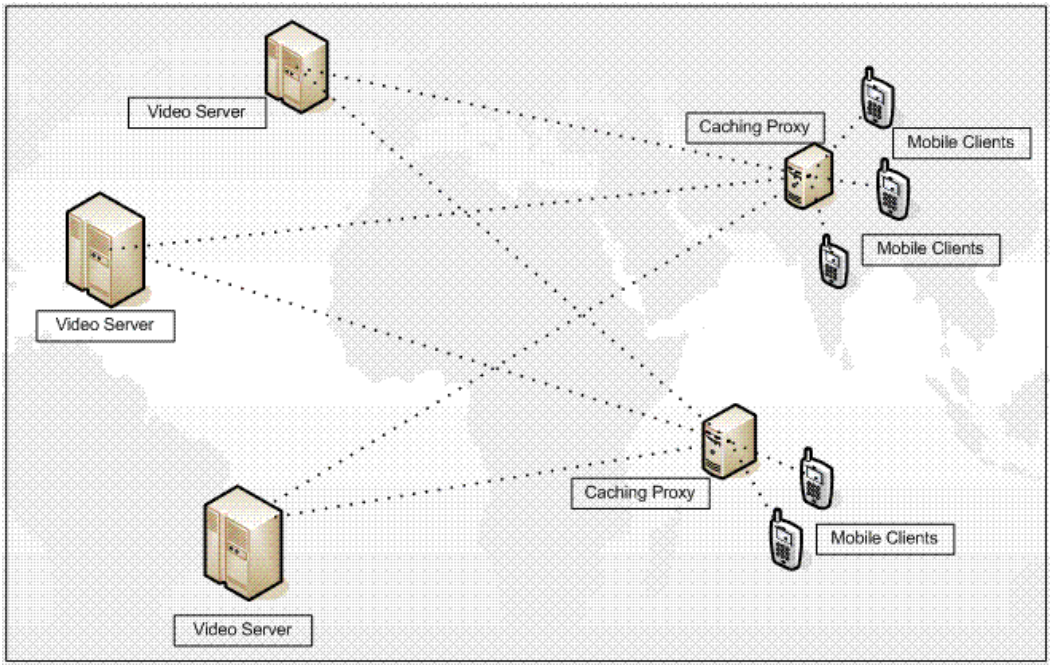

Figure 1: *Multi-client Caching Proxies and Multiple Video Personalization Server architecture*

3.1 Cache Design

The proposed multiple caching proxies serve the following purposes:

I. They act as buffer between multiple VPS(s) and multiple mobile clients, in order to reduce the overall latency observed by the clients.

II. They simulate generative proxy video servers that generate videos of different visual quality in order to increase efficiency of the cache replacement policy.

III. They perform semantic client request aggregation to reduce the request processing load and hence increase the cache replacement performance.

3.2 Personalized Video Segment Calculation

The caching proxies maintain meta-data of the semantically indexed video segments across all the VPSs.

3.2.1 Relevance Value of a Video Segment and its Summary

Video segments are indexed using semantic terms. Each video segment is assigned a relevance value based on the client's preference with regard to video content. Assume video segment S_i is indexed by a semantic term T_i. In its request, the client specifies a preference for video content using a descriptive term labeled as P. The relevance value V_i assigned to the video segment S_i is then given by:

$$V_i = similarity(T_i, P), 0 \le V_i \le 1 \qquad (1)$$

In the current implementation the *similarity* is evaluated using the *lch* semantic similarity measurement algorithm [12].

Each indexed video segment is summarized at multiple levels of abstraction using content-aware key frame selection and motion panorama computation algorithms [24]. Each video summary consists of a set of key frames and motion panoramas. For each video segment, its original version is assumed to contain the greatest amount of detail; whereas its summary at the highest level of abstraction is assumed to contain the least amount of detail. It is reasonable to assume that the amount of information contained within a video summary (relative to original version) is related to its duration, i.e.

$$v_i = v_{i0} \cdot f(L_i / L_0) \qquad (2)$$

where, v_{i0} is the relevance value of the original video segment, and L_0 and L_i are the time durations of the original video segment and the video summary respectively. Typically, the amount of information contained within a video summary (relative to original version) does not necessarily increase linearly with its relative duration.

3.2.2 MMKP-based Video Personalization

The objective of video personalization is to present a customized or personalized video summary that retains as much of the semantic content desired by the client as possible, but within the resource constraints imposed by the client. The client typically wants to retrieve and view only the contents that match his/her content preferences. In order to generate the personalized video summary, the client preferences, the client usage environment and client-side and system-level resource constraints need to be considered. The personalization engine selects the optimal set of video contents (i.e., the most relevant set of video summaries) for the client within the resource constraints imposed by the client. In our work we are implementing the MMKP-based video personalization strategy to generate a customized response to the client's

request while satisfying multiple client-side and system-level resource constraints. Compared to the 0/1 Knapsack Problem (KP)-based and the Fractional Knapsack Problem (FKP)-based video personalization strategies presented in [14], [19] and [20], MMKP-based video personalization strategy is shown to include more relevant information in its response to the client's request. The MMKP-based personalization strategy is also shown to support multiple client-side constraints, in contrast to the 0/1KP-based and the FKP-based personalization strategies which can support only a single client-side resource constraint at a time.

In many applications, it is desirable to provide the client with as much information as possible. In such cases it may be preferable to include two shorter video summaries in the response rather than a single video segment of longer duration that contains more details. For example, if a client needs to browse the sports news of the day, it might be helpful to provide him/her with multiple, though short, sports news summaries rather than a single long and detailed video segment containing news of a specific sport. The implemented Multiple-Choice Multi-Dimensional Knapsack Problem (MMKP)-based video personalization strategy [15], [20] performs around the above mentioned technique.

3.3 Multi-Stage Client Request Aggregation

The client requests arriving at the caching proxy can be distinguished based on; arrival time, video content preference and client-side resource constraints. This implementation uses a novel approach to reduce the content generation time at the caching proxy by semantically grouping the client requests using the above mentioned features. The grouping of several client requests into one request increases the proxy cache hit percentage and eventually reduces the average latency at the client-side. The semantic aggregation is performed in three stages. The

first stage involves batching client requests arriving within a pre-specified time window. The resulting group is termed a *batch group*. In the second stage, client requests within a *batch group* with similar video content preferences are clustered into *batch service groups*. Let q denote the ordered set of semantic terms used to index video segments, and Q denote the ordered set of semantic terms that the clients can use in their requests to specify their video content preferences, such that $q \subseteq Q$. Also assume that a similarity matrix $S_{size(q)\ x\ size(Q)}$ used to define the similarity of semantic terms, where $0 \leq S_{ij} \leq 1$ represents the semantic similarity of term $t_i \in q$ and term $t_j \in Q$. The value of S_{ij} is computed using the *lch* algorithm described in [12]. Let T be the semantic term used in the client's video content preference list, and let *k* be the index of term T in the ordered set Q, then the client *content preference vector*

$$P_c = (s_{1k} , s_{2k} , \ldots s_{size(q)k}) \qquad (3)$$

where, S_{ik} , $1 \leq i \leq size(q)$ is the semantic similarity between term $t_i \in q$ and $t_k \in Q$, and can be obtained from the similarity matrix $S_{size(q)\ x\ size(Q)}$. The *preference clustering* algorithm uses the cosine similarity measure between a pair of client query content preference vectors P_{c1} and P_{c2} to represent the distance between them. The *k-means* clustering algorithm is used to cluster client requests with similar content preference values into a group, termed as a *content service group*. The client requests within a *content service group* are further clustered based on client-side system-level resource constraints to generate a set of *service groups*. In our experiments, client requests with similar values for the video viewing time limit are clustered together using the *k-means* clustering algorithm.

3.4 Creating Different Layers of the Video

From each video segment that the cache receives for storage, two additional versions, or layers, of the same video are created. The original video segment, or layer, is designated as V_{org}. V_{mid} is an intermediate layer video of lower visual quality (and smaller file size) than V_{org} whereas V_{base} is the base layer video of lowest visual quality and smallest file size. The lower quality video layers V_{mid} and V_{base} are used to design an efficient cache replacement policy.

The different video layers are essentially transcoded versions of the original video at different levels of visual quality with file sizes that are significantly smaller than the file size of the original video. We have experimented with novel transcoding methods such as Features, Motion and Object Enhanced Multi Resolution (FMOE-MR) video encoding [6] and Ligne-Claire video encoding [12]. Due to its relative simplicity in terms of implementation for the purpose of caching, we have chosen FMOE-MR as the trancoding technique for this paper. In the following subsections, we describe briefly how the two layers, V_{mid} and V_{base}, are created from the original video segment V_{org}.

3.4.1 Creating V_{mid}

The video layer V_{mid} represents an intermediate-level video which has a more compact representation than the original video V_{org}, albeit at the cost of lower visual quality. The video layer V_{mid} is generated using a novel multi-resolution video encoding technique termed as Features, Motion and Object-Enhanced Multi-Resolution (FMOE-MR) video encoding [6]. The FMOE-MR video encoding scheme is based on the fundamental observation that applying a low pass filter in the image color space is equivalent to DCT coefficient truncation in the corresponding DCT (frequency) space [21].

3.4.2 Generating V_{base}

The base video layer V_{base} is generated by a method similar to the Gaussian smoothing performed in the case of FMOE-MR video encoding. The primary difference is that, in the case of the V_{base} video layer, the smoothing operation is performed uniformly over the entire video frame in contrast to FMOE-MR video encoding where the extent of smoothing can vary within a video frame based on the perceptual significance of the region under consideration. This results in further dramatic decrease in the file size, albeit at the loss of video quality. Note that V_{base} is of much lower visual quality than V_{mid} since object-based enhancement is not used.

3.4.3 File Size Overload

V_{mid} and V_{base} are additional files stored in the cache. The total sum of all the files V_{org}, V_{mid} and V_{base} is around 1.5 times that of V_{org}; i.e., the combined size of V_{mid} and V_{base} adds $\approx 50\%$ overhead to storage space required for storing the video segment.

Thus ensuring reasonably good visual quality for V_{mid} and V_{base} makes it possible to discard V_{org} to allow for extra space in the cache, and yet have videos of reasonable visual quality reside in the cache, during a cache replacement.

3.5 Cache Replacement

A novelty in the proposed cache design is that fact that a hit or a miss score depends on the *client-type*. A *client-type* defines the membership status of the client with regard to the video personalization service. Without loss of generality, we have used two categories of clients - either they are *paying* or *subscribing* clients, who are paying for the best quality of service, or they are *non-paying* clients who receive videos of varying quality depending on the availability

of the videos in the cache. For non-paying clients, the best quality video is not guaranteed; however the best quality video is guaranteed for the paying client.

If the requesting client is *non-paying*, then the cache checks if the requested file V_{org} is present in the cache. If it does not exist in the cache, it checks whether a lower quality version of the file, i.e., V_{mid}, is present in the cache. If V_{mid} is absent, then it checks whether V_{base} is present in the cache. If neither V_{mid} nor V_{base} are present in the cache the client request is treated as a cache miss. In contrast, if the requesting client is *paying* then the client request is treated as a cache miss if the requested file V_{org} is absent in the cache. Thus, for a *paying* client, the best quality video is provided, whereas for the non-paying client, the quality of video which is present in the cache is present; it might not be the best quality one.

In the event of a cache miss, whether the client type is a *paying* or *non-paying*, the requested file is relayed from the VPS, and also stored in the cache simultaneously. If there is not enough space available in the cache, a cache replacement policy is enforced to replace existing file(s) in the cache in order to make space for the requested file. If the cache cannot accommodate the requested file, an existing file with the minimum *retention-value* (RV) is discarded. The *retention-value* is essentially a number associated with each file in the cache; the lower the *retention-value*, the less valuable is the file to the cache. Thus, if an existing file is to be discarded from the cache in order to make space in the cache, the one with the lowest *retention-value* is removed.

The computation of the *retention-value* is specific to a cache replacement policy. For the proposed cache, we compute the *retention-value* as follows:

$$retention\text{-}value = NCC/fileSize \qquad (4)$$

where, NCC is the *number of common clients* requesting that particular file, and *fileSize* is the size of the file. This cache replacement policy has been termed NCCS (Number of Common Clients - Size) algorithm. NCCS essentially denotes the number of client requests for this particular file normalized by the file size. In order to assign higher priority to files which are requested by paying clients, the number of clients (NCC) is incremented by 10, instead of 1, for paying clients as opposed to non-paying clients. This information is obtained during the first phase of the communication between the cache and the VPS.

As will see in the following experimental results section, the proposed NCCS cache replacement policy that uses the proposed *retention-value* as the replacement criterion is better suited for caching of personalized video than other standard cache replacement algorithms.

Recall that in the case of a miss, the file with the least *retention-value* in that category is discarded. The file replacement process is as follows. First, the video file with the minimum *retention-value* is identified. After that, an attempt is made to remove the original video segment; i.e., V_{org} corresponding to that video file. If V_{org} had already been removed previously, then V_{mid} is removed; if V_{mid} has also been removed, then V_{base} is removed. This top down approach ensures that the layer which takes up the largest amount of space in the cache is removed, so that space for more popular video segments, can be made.

CHAPTER 4

CLIENT-CACHE-SERVER COMMUNICATION

The client-cache-server communication is done in two phases:

Phase 1:

Multiple clients send their queries or requests to the caching proxy, which uses multi-stage client request aggregation algorithm to generate a sub-request of incoming client requests. Each request video preference essentially consist of one category out of six possible categories (*News Anchor*, *News*, *Sports News*, *Commercial*, *Weather Forecast* and *Program Header*), and the available battery time in the client device as a resource constraint. The cache generates a list of video segments based on the client video preference and resource constraints, and sends the names of the video segments to be downloaded (in a specified order), as metadata back to each client.

Phase 2:

Each client now makes a series of requests to the caching proxy for files belonging to one of the four categories. The files can be streamed, or can be progressively downloaded, depending on the type of service available at the caching proxy. In either case, recall that the names of the files, and the order they appear in, have been specified in the metadata obtained in *Phase 1* of the communication. When a client requests a video segment file from the cache, it is scored as either a *hit* or a *miss*. **Figure 2** depicts the two phases of communication described above.

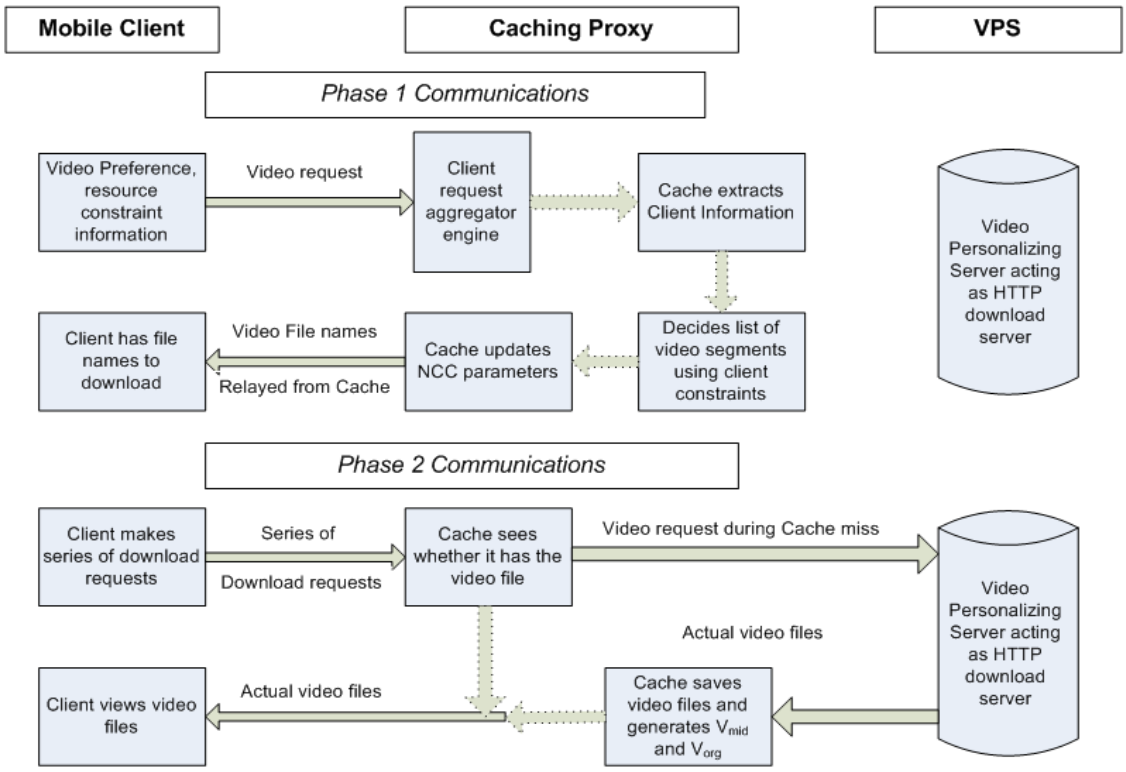

Figure 2: *The client-cache-server communication protocol..*

CHAPTER 5

EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of the proposed caching proxy. First, we describe the experimental setup, and then discuss the results.

5.1 Experimental Setup

Our simulation consists of 1 video personalization server and 3 proxy caches. The server and caches were deployed on Pentium-4, 2.79 GHz Machines with 2 GB of RAM running on Linux Red Hat 4 Operating Systems. The number of clients varied between 50 and 200. However, unless specified otherwise, the results presented in this paper are for simulations involving 150 clients. It is assumed that around 40% of the clients are paying clients; the rest are non-paying. The time-window size for the first stage of client-request aggregation at the caching proxies was set to 3000ms. In order to simulate the real world scenario the mobile clients contact the HTTP server proxy which redirects the clients to one of the 3 proxy caches. So we have the Video Personalizing Servers and the HTTP server proxy running on different ports on Machine A, Caching Proxies running on different ports on Machine B and the clients running on separate ports on Machine C. During the experiments, the Internet network distance [17] between the VPSs and client is assumed to be several folds higher than between Caching Proxy and client to simulate for remoteness. The assignment of proxy caches implements the Global Network

Positioning [16] approach to calculate Internet network distance [17] which is discussed later in this section.

Our experiments involved multiple sessions of client-cache-server communication. Each session consists of multiple clients, each performing the preference-request once. All the proxy caches start empty (no cached video files) for each session. During each session the clients contact the HTTP server proxy with time lag determined by a random function generator which follows Poisson Distribution with a max-delay of 4 seconds and variance of 2 seconds. The client preference value is generated using a random uniform distribution generator over the defined client preference vector. The resource constraints namely viewing-time limit and bandwidth are generated using Normal Distribution generators. In order to simulate mobile devices with different viewing time limits, the values for viewing time limits are modeled as a mixture of 2 normal distributions $N_1(\mu_1, \sigma_1^2)$ and $N_2(\mu_2, \sigma_2^2)$ where,

$\mu_1 = 50$ seconds, $\sigma_1 = 70$ seconds and

$\mu_2 = 150$ seconds, $\sigma_2 = 70$ seconds.

The Normal Distribution generator for bandwidth, $N_3(\mu_3, \sigma_3^2)$ has,

$\mu_3 = 180$ KBps and $\sigma_3 = 40$ KBps.

Each caching proxy maintains a cost-table which assigns a cost metric to all the files on the VPS. The cost-metric is generated using the Global Network Positioning [16] approach. For the set of initial Landmarks [16] we use the VPS to provide the set of reference coordinates. The Caching Proxies derive their positioning coordinates using the VPS as the reference. The HTTP server proxy calculates the network positioning coordinates using the Caching Proxies as the landmark. The clients in turn derive their Global Network Positioning coordinates using the HTTP server proxy as the landmark. This hierarchical approach to derive the network distances

between the clients, caching proxies and video servers provides a flexible and scalable implementation. As the clients are mobile, the network positioning coordinates could change during the communication process. The Global Network Positioning approach provides with faster and efficient Internet network distance calculation approach compared to other existing approaches based on IDMaps [17].

We have plotted the results for the cache performance metrics with total size of caching proxies. The cache size is taken as a percentage of the total VPSs size. The performance metrics used for comparison are:

- *Average Client Latency*. The client latency is calculated as the time lapsed after the client makes the request during first phase of communication and when it receives the first byte of first video requested during the second phase of communication.
- *Hit Ratio;* The number of requests satisfied by the proxy cache as a percentage of total requests.
- *Byte Hit Ratio*; this value is calculated by taking the ratio of hit bytes (the file size of hit files) and number of hits for each cache. The final value is obtained by taking a weighted average of all the values obtained above using number of requests as the weight.
- *Bytes per Request* transferred by the VPS as a percentage of no-caching implementation.

5.2 Cache Replacement: Retention Function

We have considered four other standard cache replacement policies in order to do a performance comparison with the proposed NCCS cache replacement algorithm. At any time in its life in the cache, the *retention-value* of each video segment may depend on the following parameters:

- *nClients*: The number of clients which are requesting this file. This information is derived during Phase 1 of the communication when the clients make initial requests for the files to the cache. If a requesting client type for a file is non-paying, *nClients* is incremented by 1. If the client is a paying client, then *nClients* is incremented by 10 (the value 10 is empirically chosen). This scheme can accommodate multiple grades of service; for example, for even more special clients, the value of *nClients* can be incremented by 20, and so on.
- *nMisses*: The number of misses that a video segment has experienced so far. The first time a video file is requested is obviously scored as a miss; as a result, the minimum value of *nMisses* is 1.
- *nHits*: The number of hits that a video segment has experienced so far.
- *Size*: The file size on hard disk of the video segment. Typically, larger files are not preferred in the cache because the general philosophy is to have more, smaller-size files, instead of one, large file.
- *Latency*: Latency incurred while fetching the video segment file. If the video segment files are distributed over multiple video personalizing servers, the latency is an important criterion since it is typically not desirable to fetch files that are far away in terms of network distance.
- *hitTime*: The time (in nanoseconds) at which the latest hit to the video segment was made.

The replacement algorithms, with their corresponding *retention-functions*, are as follows:

GDS (Greedy-Dual-Size): The GDS cache replacement algorithm uses a retention value given by,

Retention_value = Latency/Size

In addition to removing files with the minimum retention value during a cache replacement, the GDS algorithm also subtracts this minimum *retention-value* from the *retention-value* for each of the other files in the cache. When a file is scored as a hit, the original *retention-value* of the file is recomputed.

GDSF (Greedy-Dual-Size-Frequency): The GDSF cache replacement algorithm [14] uses a retention value given by,

Retention_value = nHits × Latency/ Size

Similar to GDS, GDSF subtracts this minimum *retention-value* from all the *retention-value* in each of the other files in the cache. When a file is hit, the original *retention-value* of the file is recomputed.

LRU (Least Recently Used): The retention value is given by,

Retention_value = hitTime

This simple retention value is used to replace files that were accessed farthest in the past.

LFU (Least Frequently Used): The retention value is given by,

Retention_value = nHits + nMisses

In other words, the *retention-value* is the number of accesses to the file, which can be simply used as a measure of the frequency of access.

5.3 Simulation Results

In the first experiment (Experiment #1), the origin server contains 48 video files. It is assumed that each file is 3000 KB. The number of clients in the environment is 150. We vary the sizes of the proxy caches from 5% to 45% of the storage available on the origin server. The results are shown in Figures 3, 4, and 5. Figure 3 displays the hit rates of various cache replacement strategies at various cache sizes. Figure 4 shows the corresponding byte hit ratios. As figures show, NCCS yields the highest hit rates at all cache sizes. The improvements in cache hit rates translate into reductions in client latencies as depicted in Figure 5.

In the second experiment (Experiment #2), the VPS contains 41 files. However, unlike the first experiment, the sizes of files vary and are drawn uniformly from the range (2200 KB, 5500 KB). The rest of the configuration remains identical to the first experiment. Figure 6, Figure 7 and Figure 8 show the results of the experiments. Again we see NCCS performs best in terms of all three metrics.

We surmise that the reason proposed NCCS cache replacement policy outperforms other cache replacement schemes is because of the fact that, in this particular case, due to the two-phase communication initiated between the clients and VPS, the NCCS scheme knows in advance which files will be asked for by the clients. Note that this is a special situation that can arise only in the case of the above two-phase communication. Thus, the cache captures the popularity of a file by computing the commonality of that particular file (video segment) amongst all the clients who will request the file in the near future. This fact is captured by the NCC statistics (number of common clients). In addition, the Size factor in the NCCS cache replacement policy makes NCCS prefer more, smaller files, compared to fewer, larger files,

which should be the case for a good caching scheme. As a result, the proposed NCCS cache replacement policy is observed to be the most successful.

In the next set of experiments (Experiment #3), we study the benefits of the multi-stage client request aggregation (MSCRA) technique. The simulation setup comprises of 150 clients, 41 files at the origin server with file sizes varying in the range (2200 KB, 5500 KB). We compare the scenarios wherein the proxy caches implement NCCS replacement strategy with and without MSCRA technique. Figure 9 shows the byte hit rates of the two cases when cache size varies from 0% to 100% of the storage at the origin server. Figure 10 shows the bytes transferred from the origin server and Figure 11 depicts the average client latency values. The results demonstrate that MSCRA results in substantial reduction in server loads and client latencies. The figures clearly show that the performance of NCCS outdoes all the other cache replacement policies.

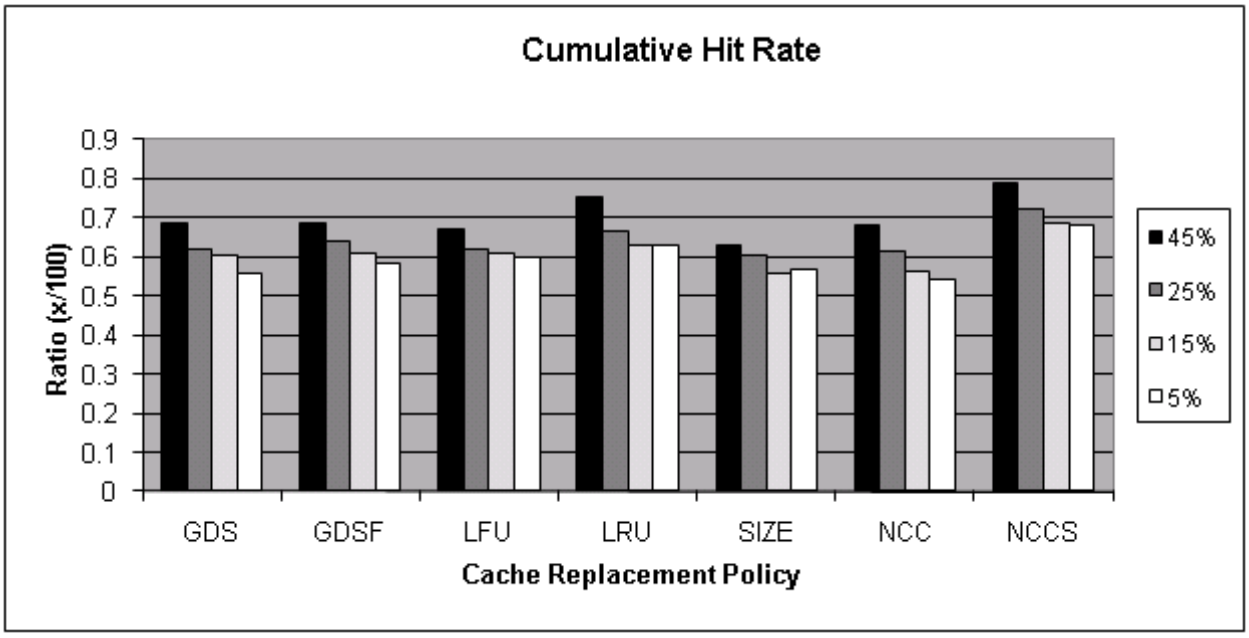

Figure 3: *Cumulative cache hit rate – Experiment #1.*

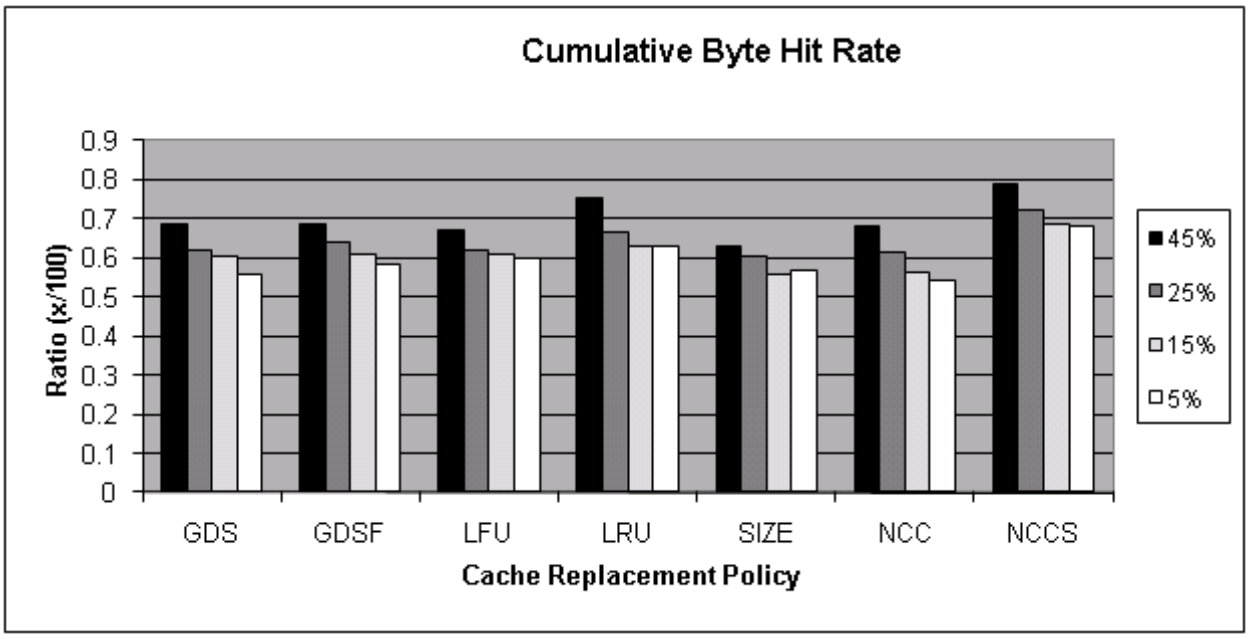

Figure 4: *Cumulative cache byte hit rate – Experiment #1.*

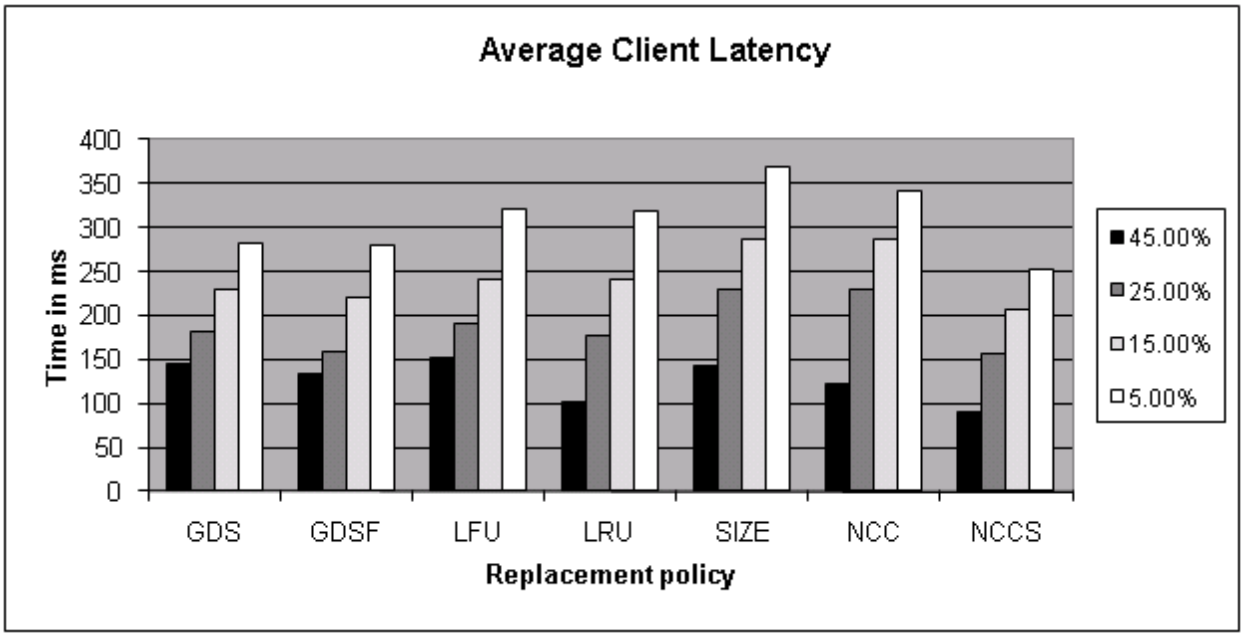

Figure 5: *Average client latency – Experiment #1.*

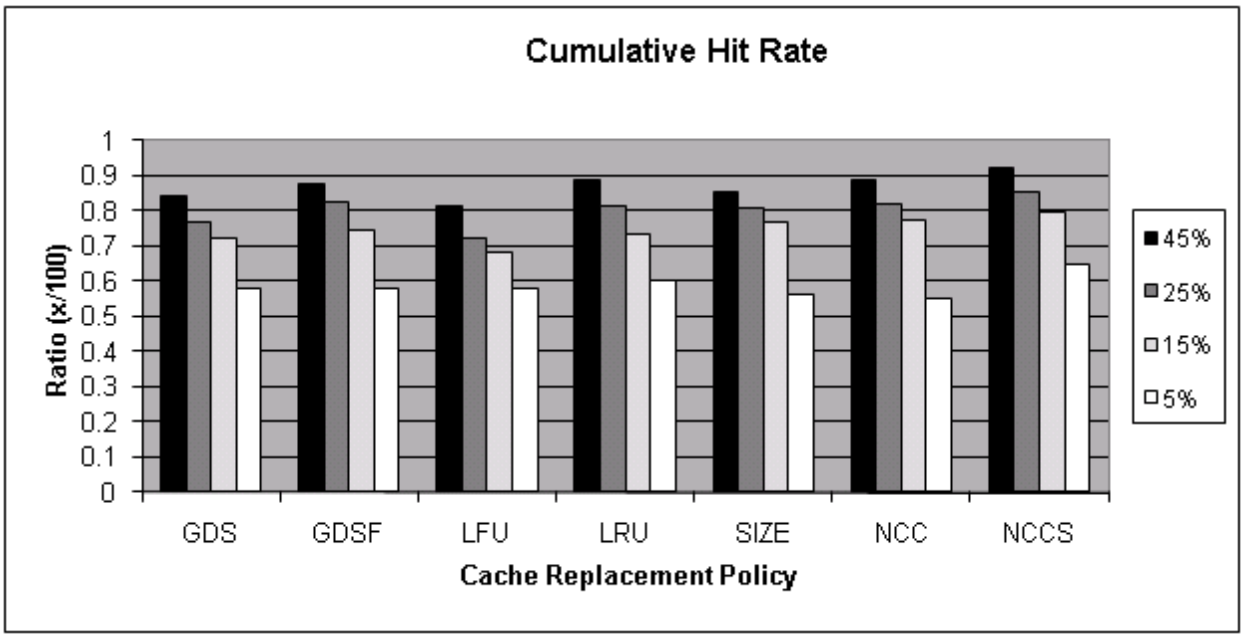

Figure 6: *Cumulative cache hit rate – Experiment #2.*

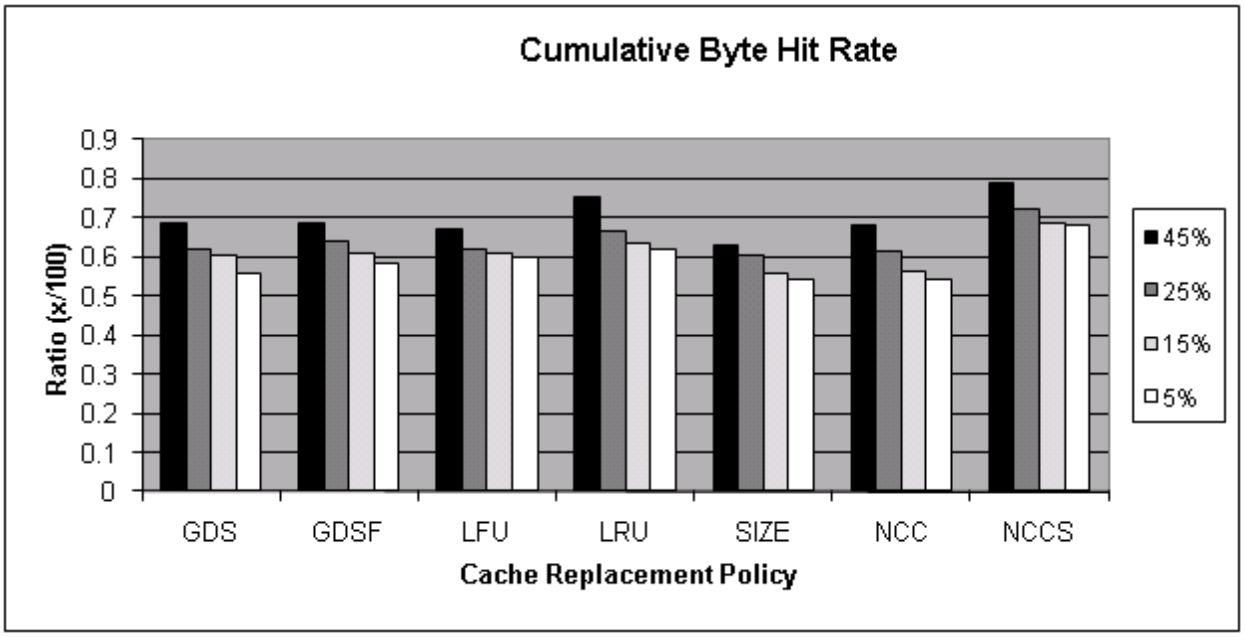

Figure 7: *Cumulative cache byte hit rate – Experiment #2.*

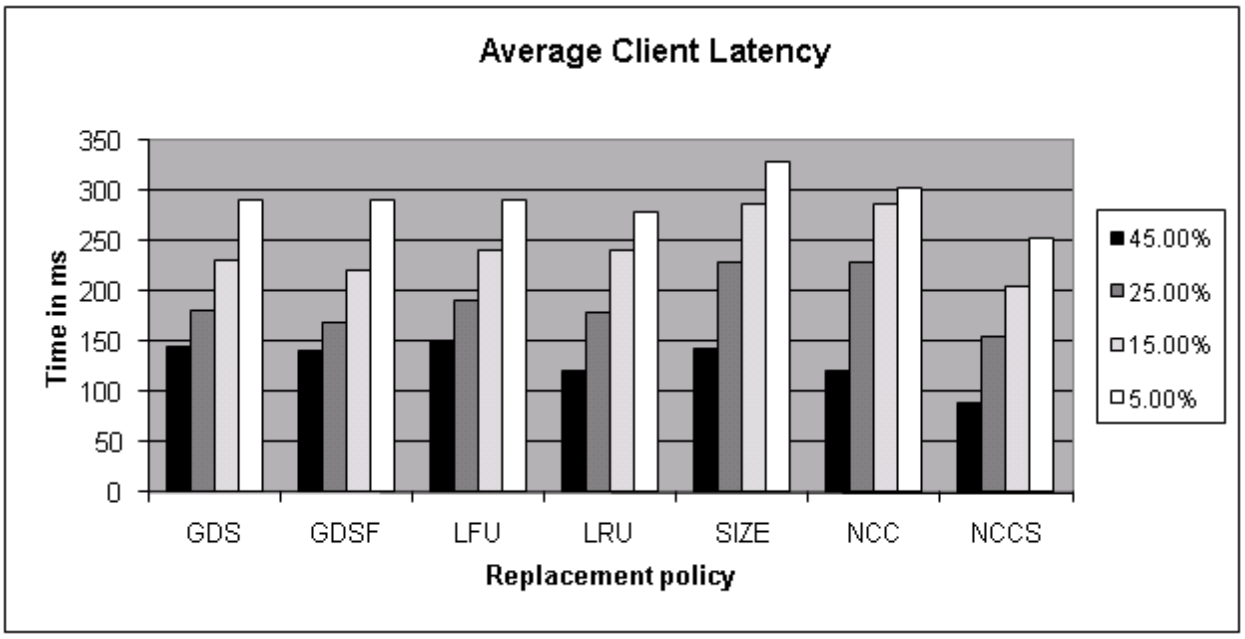

Figure 8: *Average client latency – Experiment #2.*

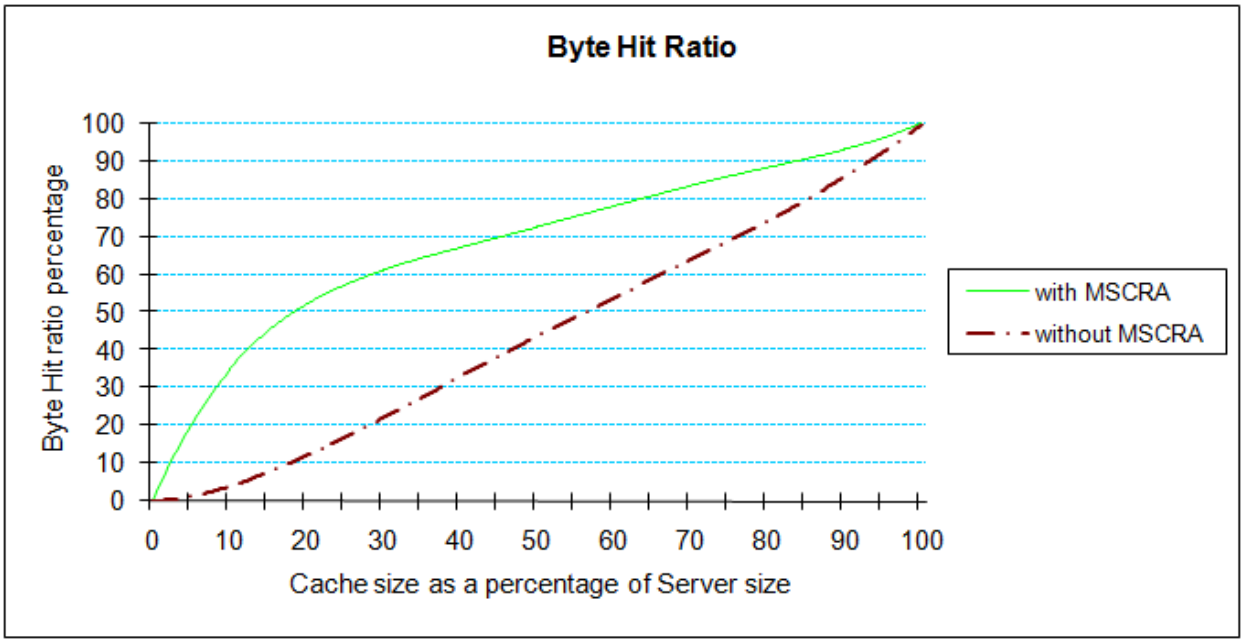

Figure 9: *The Byte Hit Ratio calculated as a weighted average of all the caching proxies – Experiment #3.*

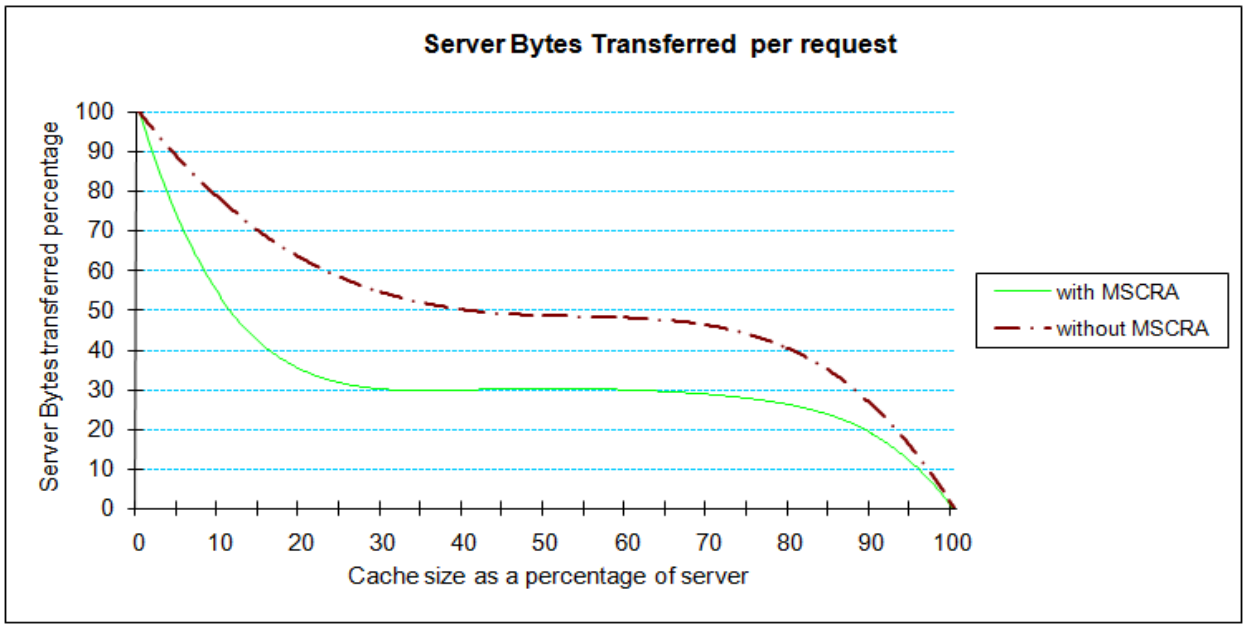

Figure 10: *The Server Bytes transferred per request calculated as a percentage of no-caching implementation – Experiment #3.*

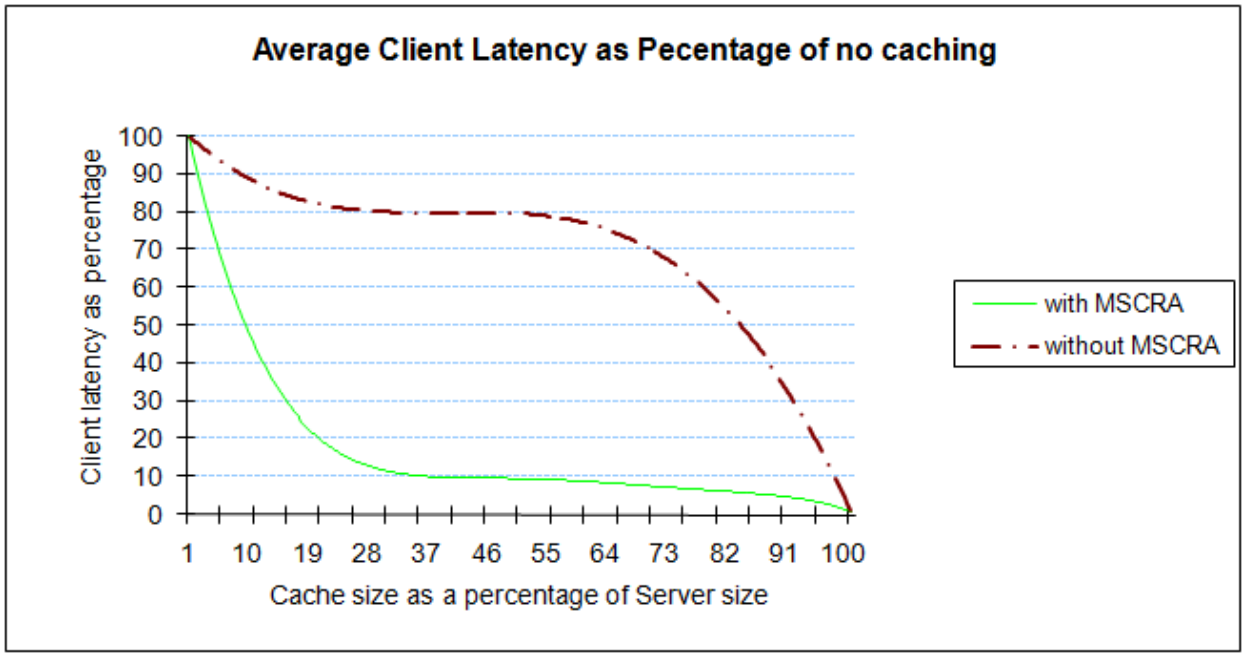

Figure 11: *The average client latency calculated as the percentage of no-caching implementation with increasing total size of caching proxies as a percentage of total size of VPS – Experiment #3.*

The next set of experiments (Experiment #4, Experiment #5 and Experiment #6) simulates the client-cache hot-spot scenario, where proxy caches are hot-spots/popular for some specific subset of file requests. The configuration is similar to Experiment #2, but for these experiments, we have divided clients in 3 non-overlapping groups based on file requests and each group is assigned a proxy cache. Figures 12, 13 and 14 (Experiment #4) display the results for scenario where 33% of the clients from each group make requests to their respective proxy cache and the rest 67% are uniformly distributed among the rest of the proxy caches.

Similarly, Figures 15, 16 and 17 (Experiment #5) and Figures 18, 19 and 20 (Experiment #6)show results for 50% and 75% respectively of each group's requests made to their respective cache and the rest 50% and 25% respectively are uniformly distributed to other proxy caches.

The results displayed in Figures 21-29 (Experiment #7, Experiment #8 and Experiment #9), have the experiment configuration similar to Experiment #3, but the client requests followed the above mentioned schema(Experiment# 4, Experiment #5 and Experiment #6 respectively). The results again display the performance of MSCRA in reducing server loads and client latencies.

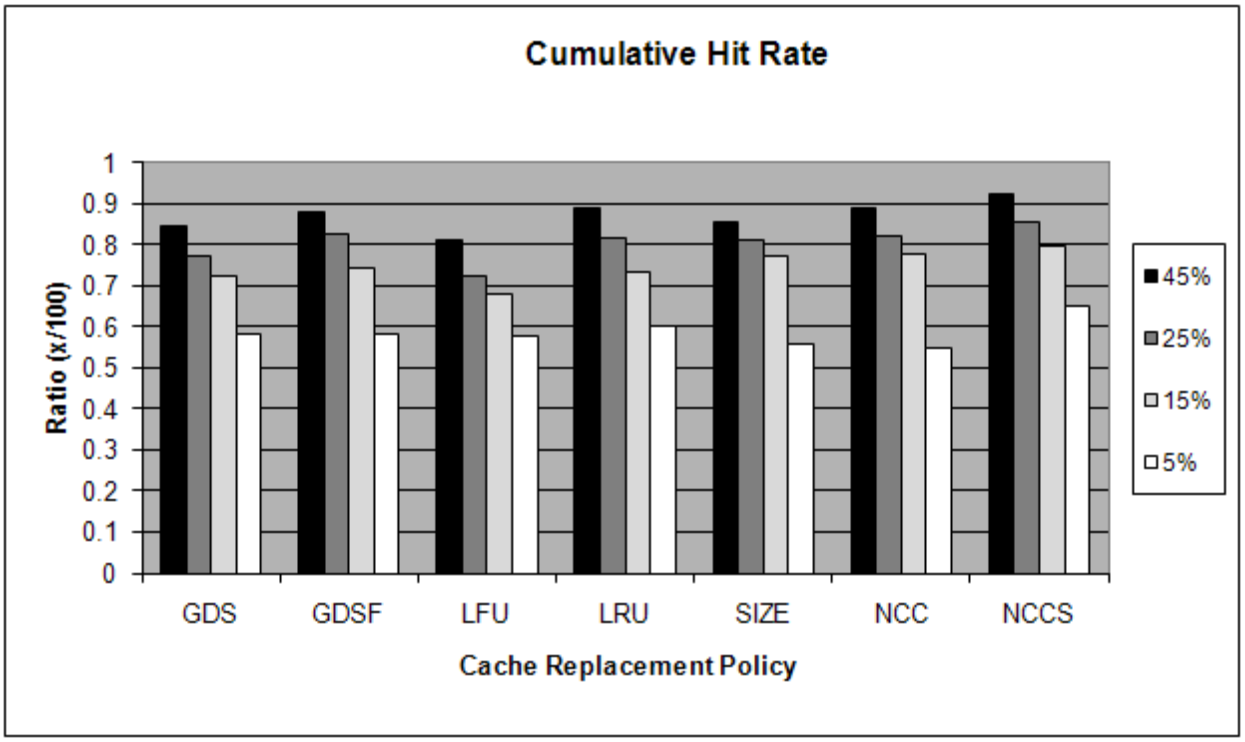

Figure 12: *Cumulative cache hit rate – Experiment #4.*

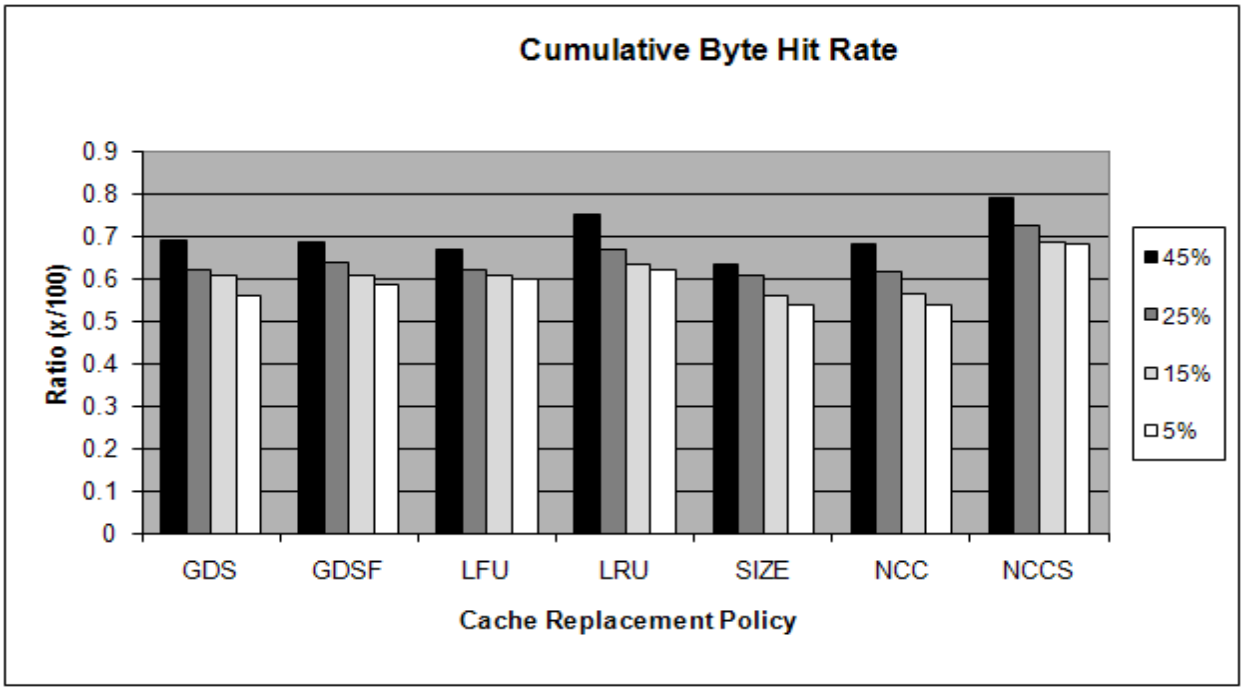

Figure 13: *Cumulative cache byte hit rate – Experiment #4.*

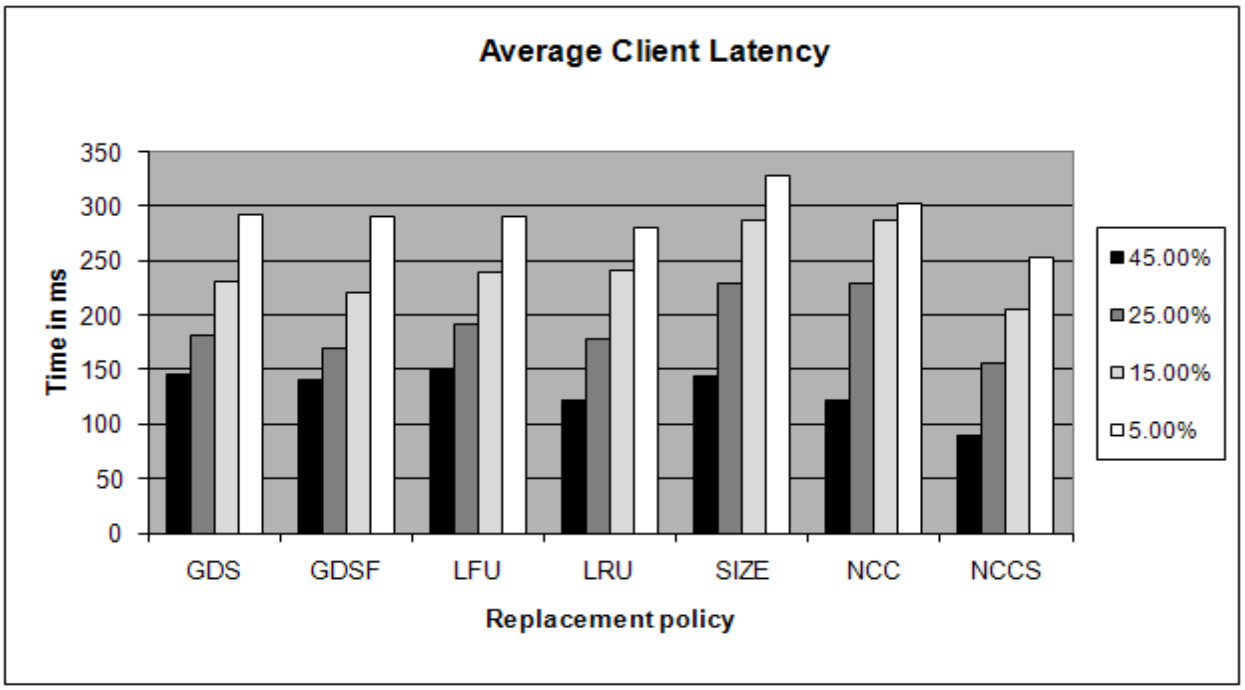

Figure 14: *Average client latency – Experiment #4.*

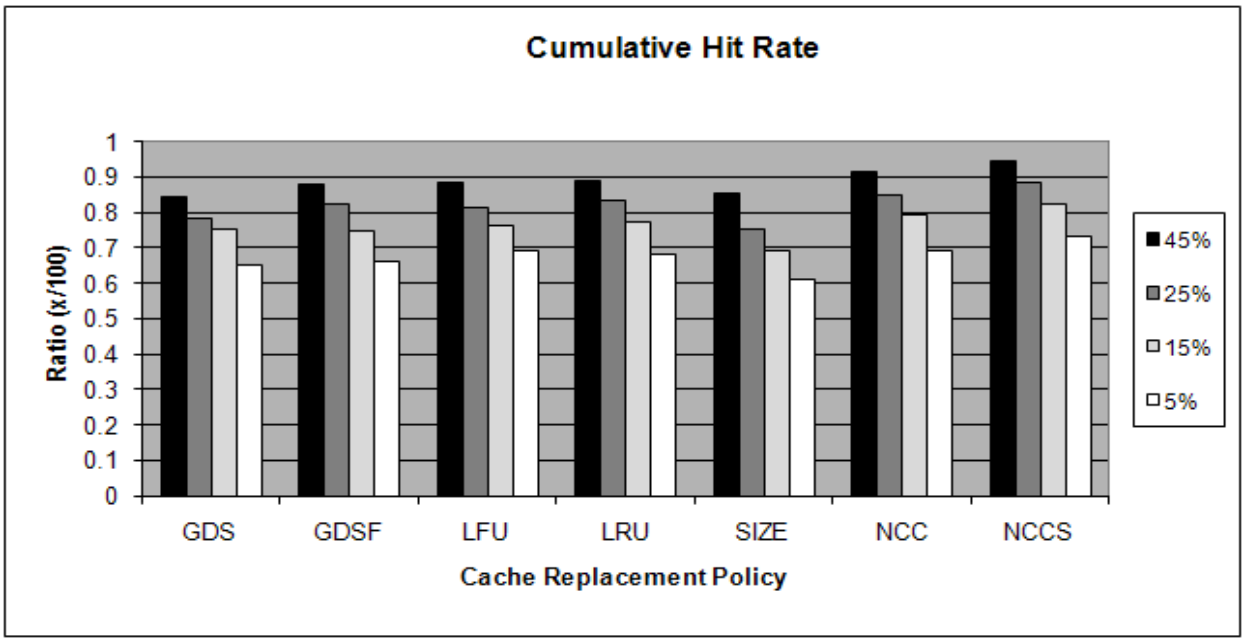

Figure 15: *Cumulative cache hit rate – Experiment #5.*

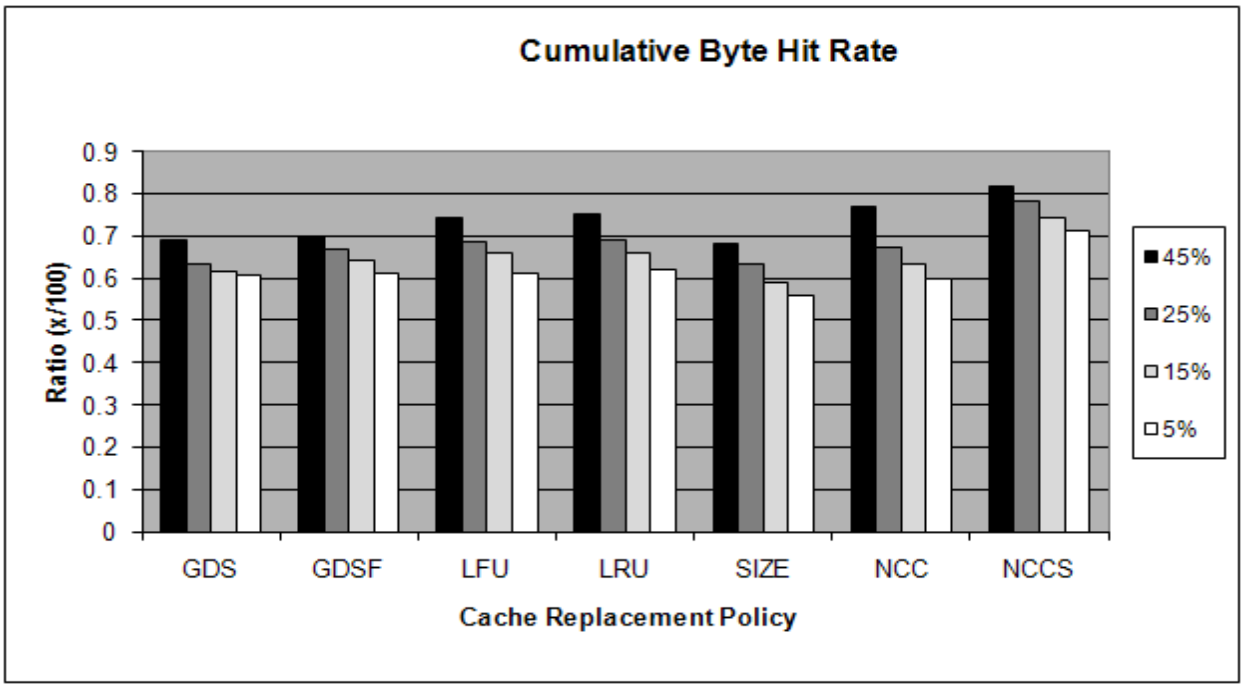

Figure 16: *Cumulative cache byte hit rate – Experiment #5.*

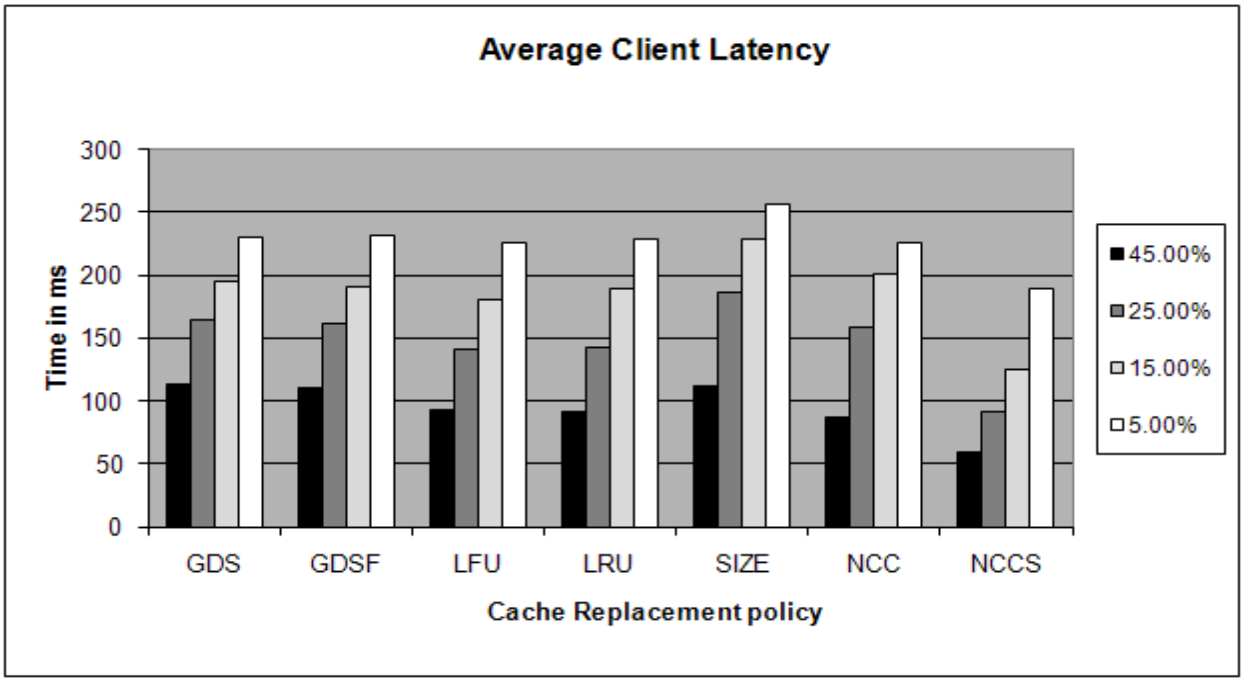

Figure 17: *Average client latency Experiment #5.*

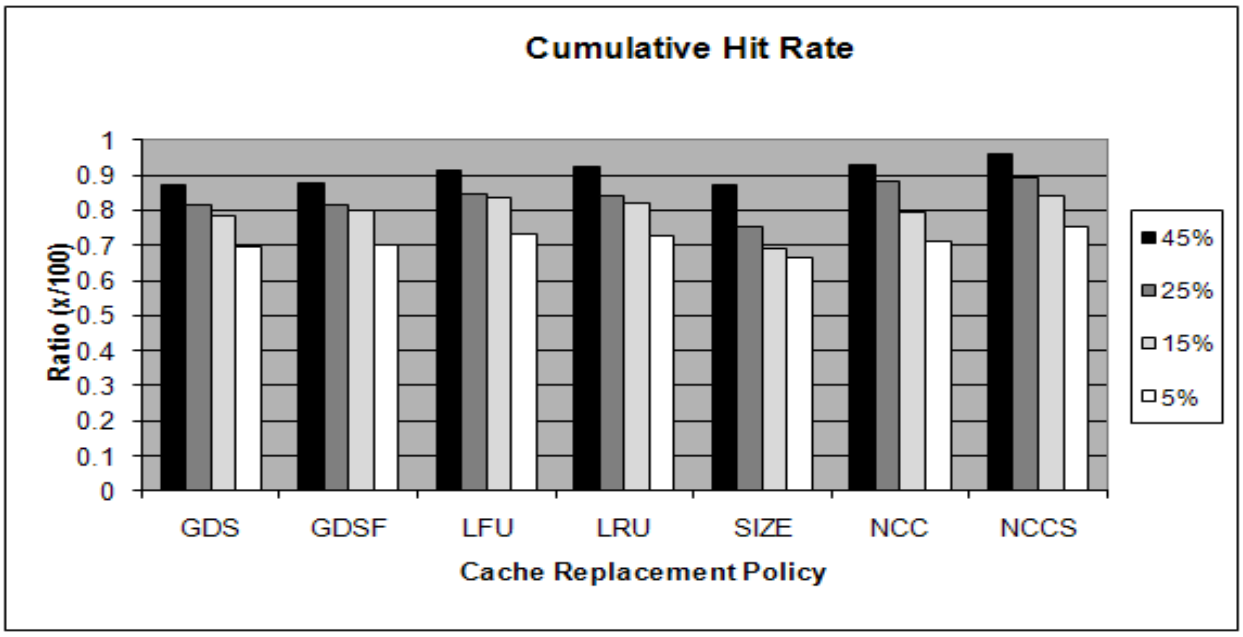

Figure 18: *Cumulative cache hit rate – Experiment #6.*

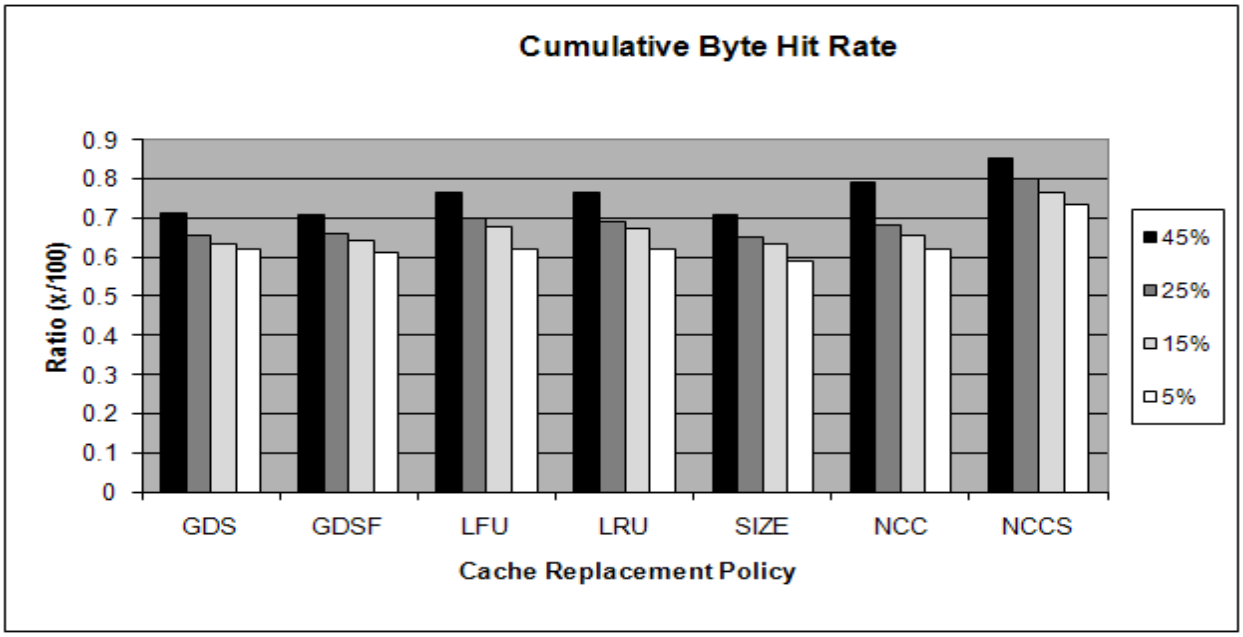

Figure 20: *Cumulative cache byte hit rate – Experiment #6.*

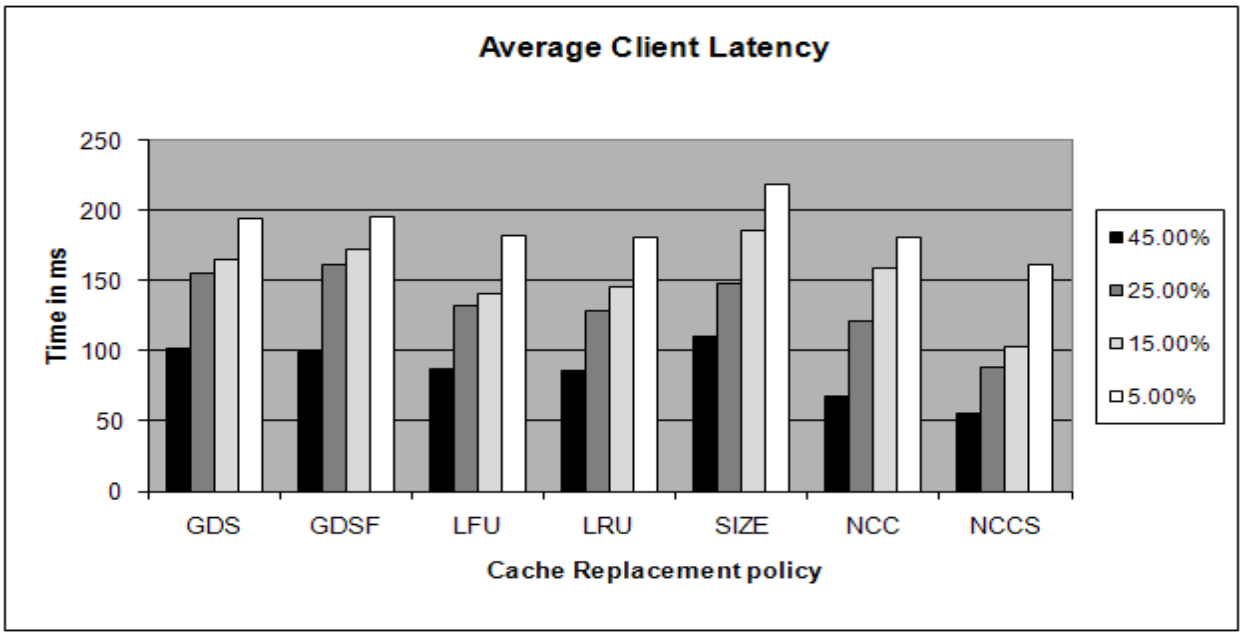

Figure 19: *Average client latency – Experiment #6.*

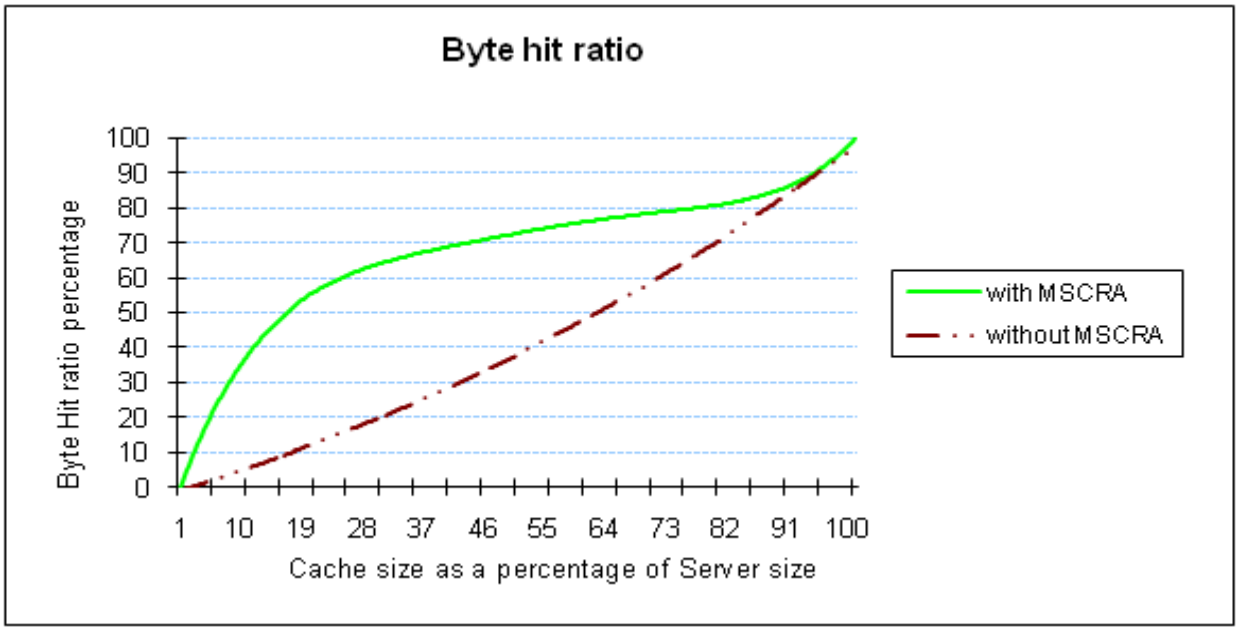

Figure 21: *The Byte Hit Ratio calculated as a weighted average of all the caching proxies – Experiment #7.*

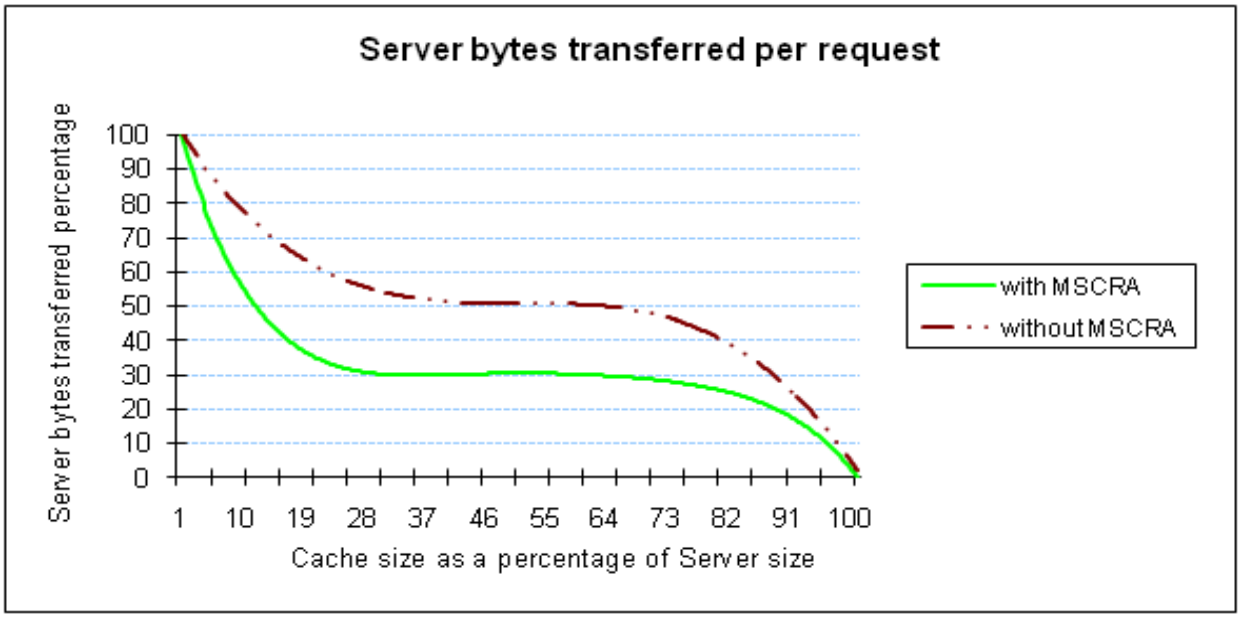

Figure 22: *The Server Bytes transferred per request calculated as a percentage of no-caching implementation – Experiment #7.*

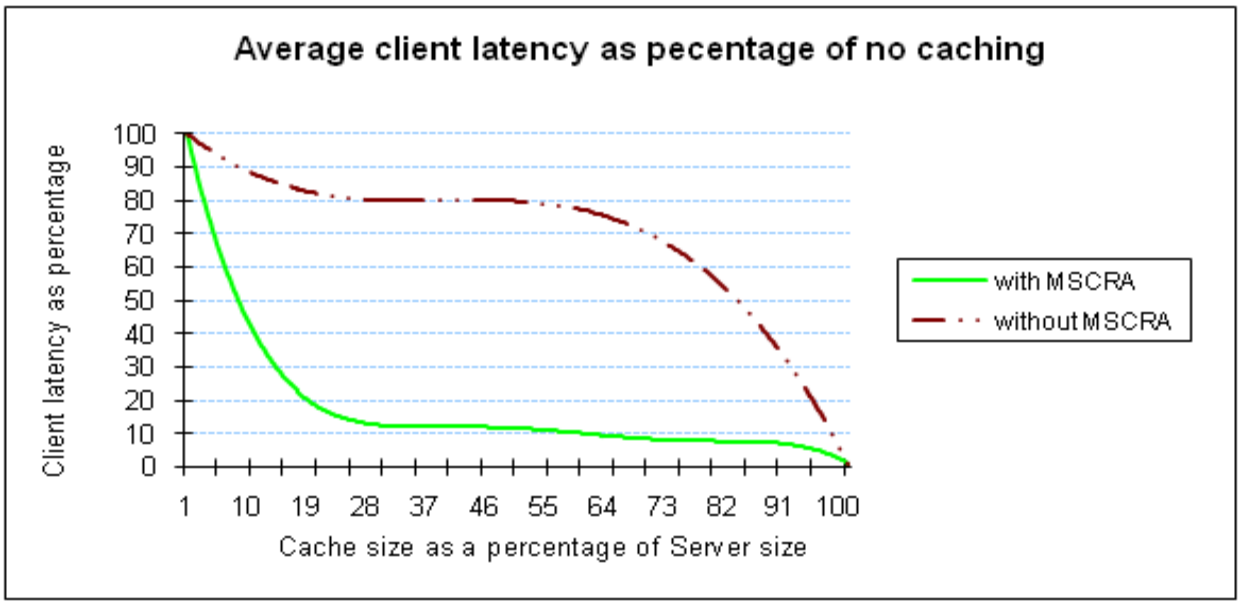

Figure 23: *The average client latency calculated as the percentage of no-caching implementation with increasing total size of caching proxies as a percentage of total size of VPS – Experiment #7.*

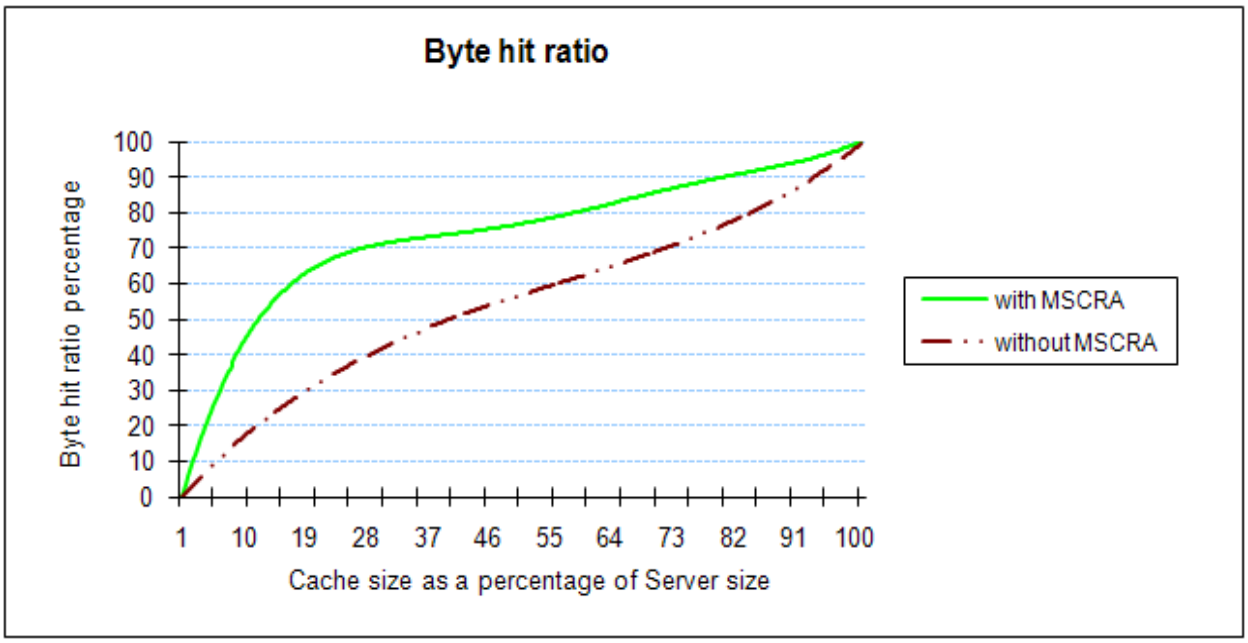

Figure 24: *The Byte Hit Ratio calculated as a weighted average of all the caching proxies – Experiment #8.*

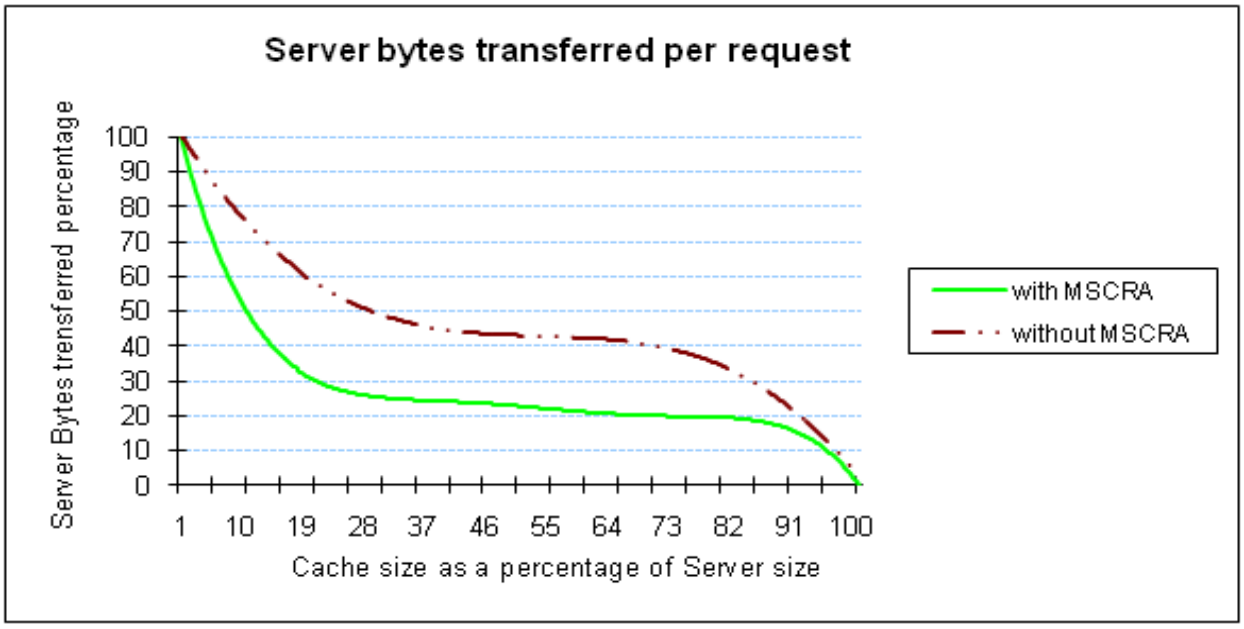

Figure 25: *The Server Bytes transferred per request calculated as a percentage of no-caching implementation – Experiment #8.*

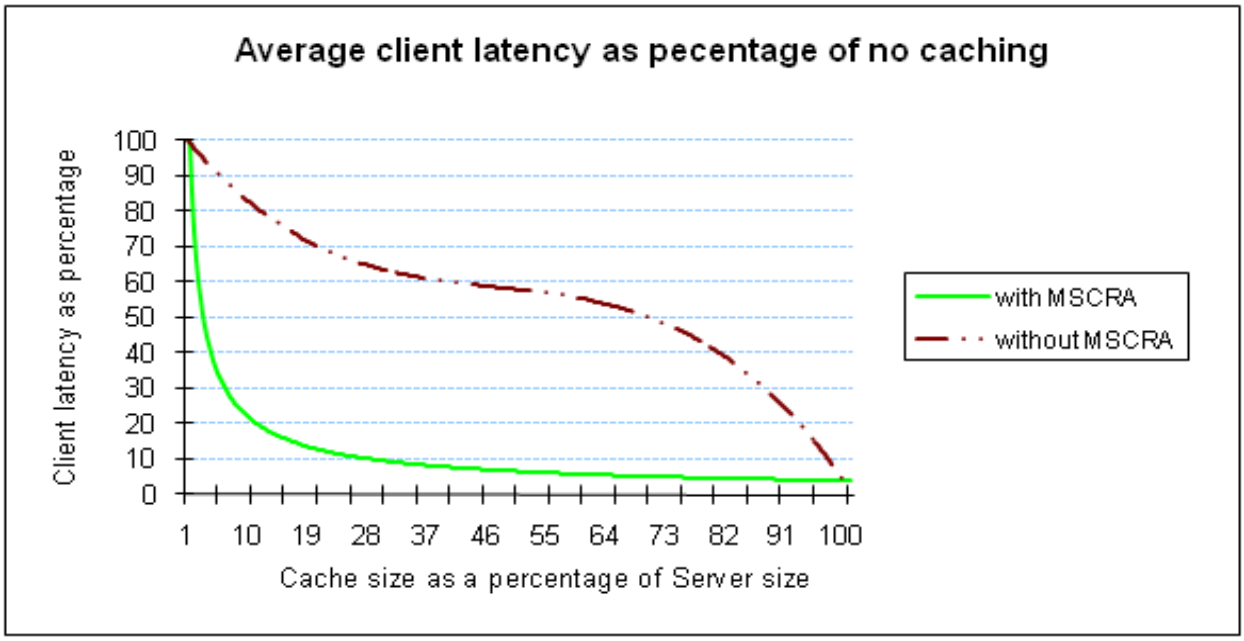

Figure 26: *The average client latency calculated as the percentage of no-caching implementation with increasing total size of caching proxies as a percentage of total size of VPS – Experiment #8.*

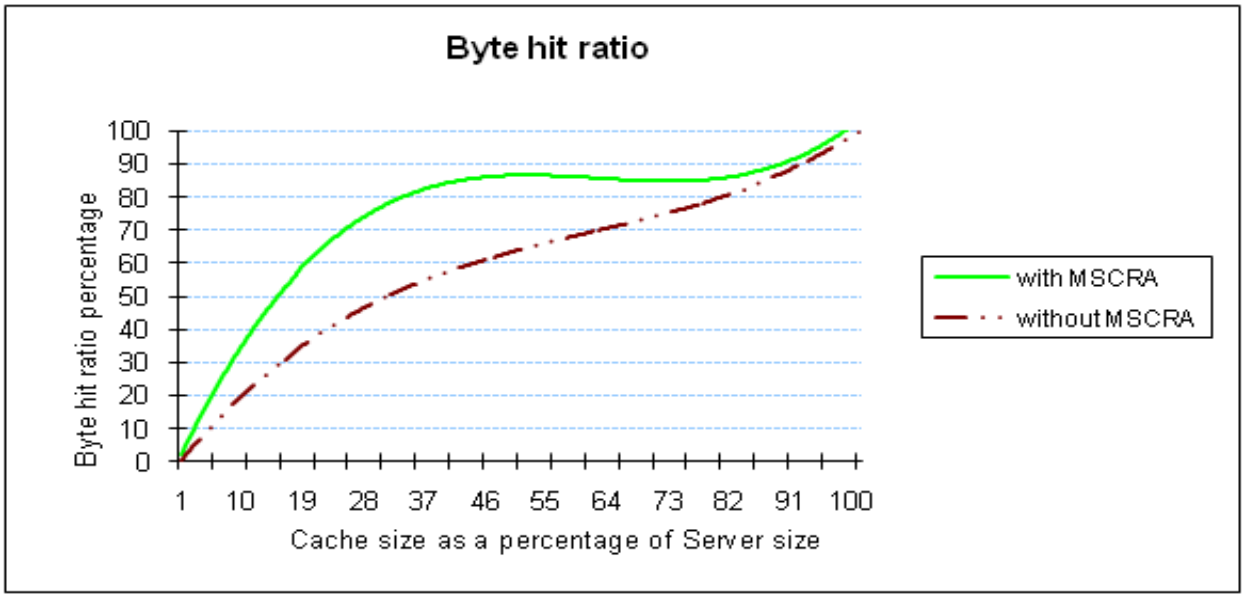

Figure 27: *The Byte Hit Ratio calculated as a weighted average of all the caching proxies – Experiment #9.*

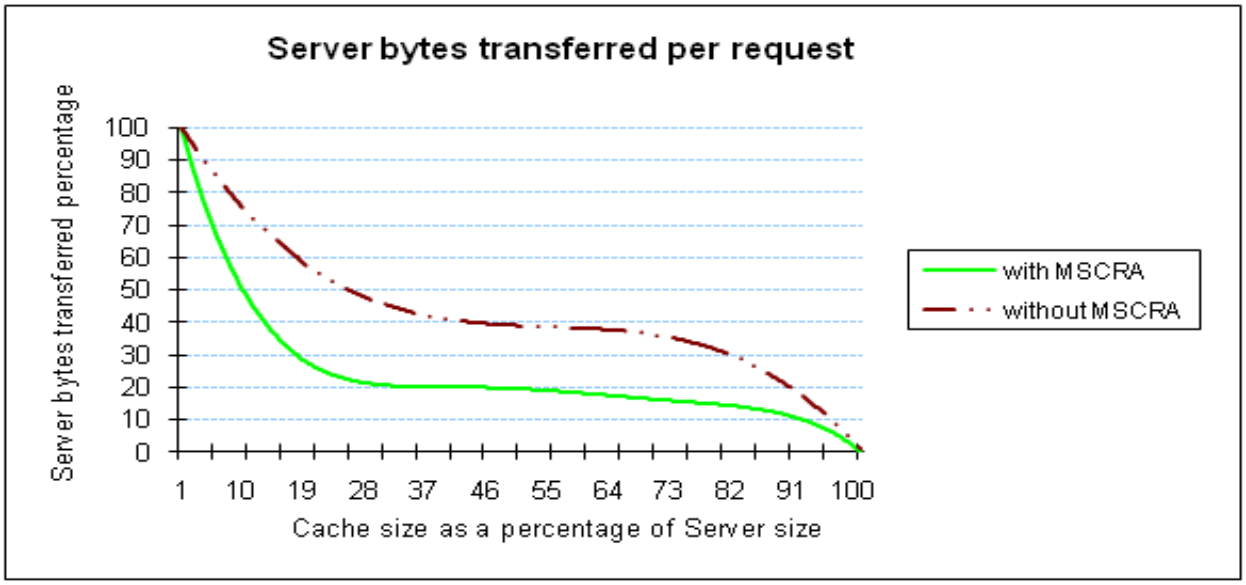

Figure 28: *The Server Bytes transferred per request calculated as a percentage of no-caching implementation – Experiment #9.*

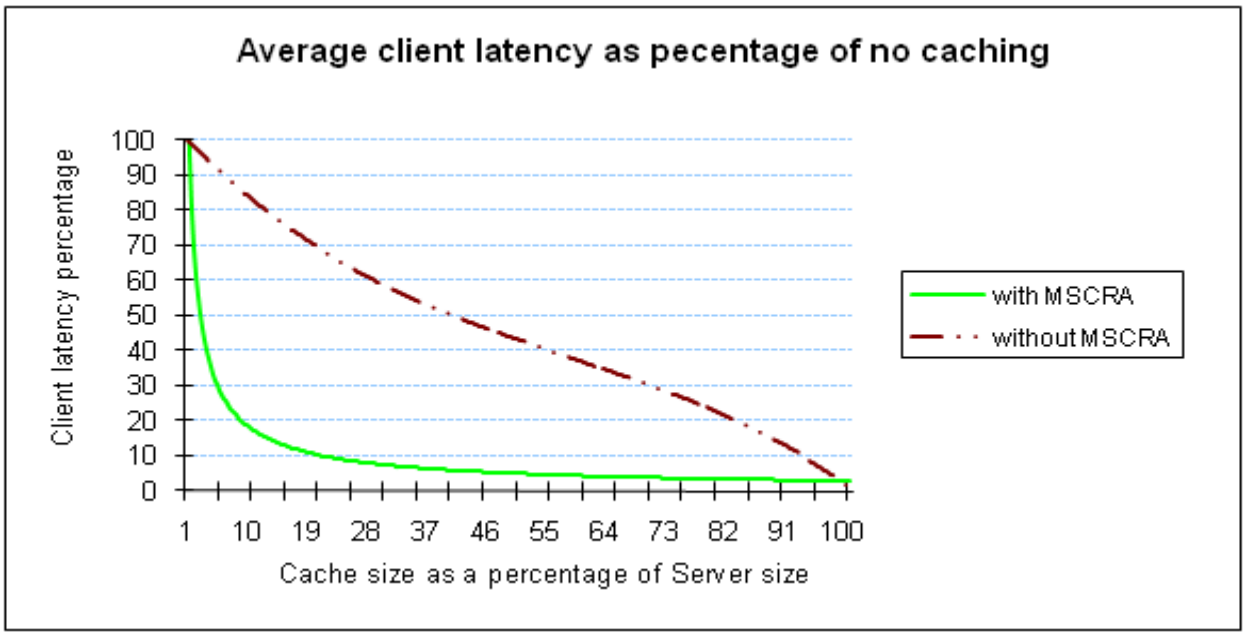

Figure 29: *The average client latency calculated as the percentage of no-caching implementation with increasing total size of caching proxies as a percentage of total size of VPS – Experiment #9.*

The results in Figure 30-35 display the performance of the *GDSF*, *LFU*, *NCC* and *NCCS* replacement policies. The results are extracted from Experiments #4– 6, to show the performance of the above mentioned replacement policies with changing client-group percentage for two proxy cache sizes, 5% and 45%. The results clearly show that the performance of the proxy caches is still remarkable even with the smaller percentage of similar client-requests.

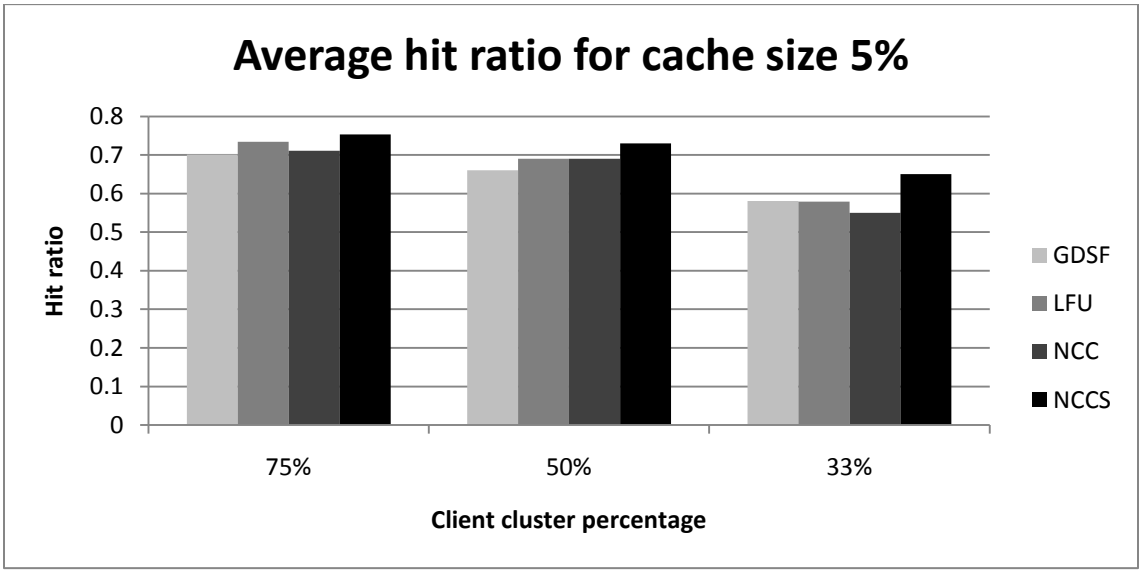

Figure 30: *Average hit ratio – Experiments #4-6.*

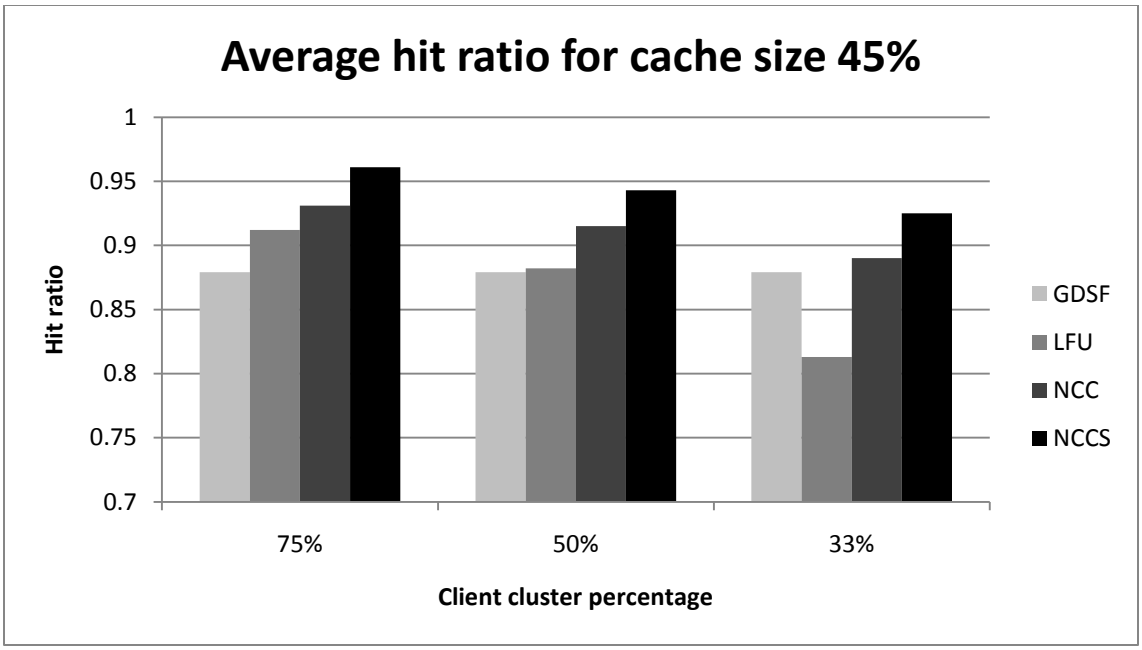

Figure 31: *Average hit ratio – Experiments #4-6.*

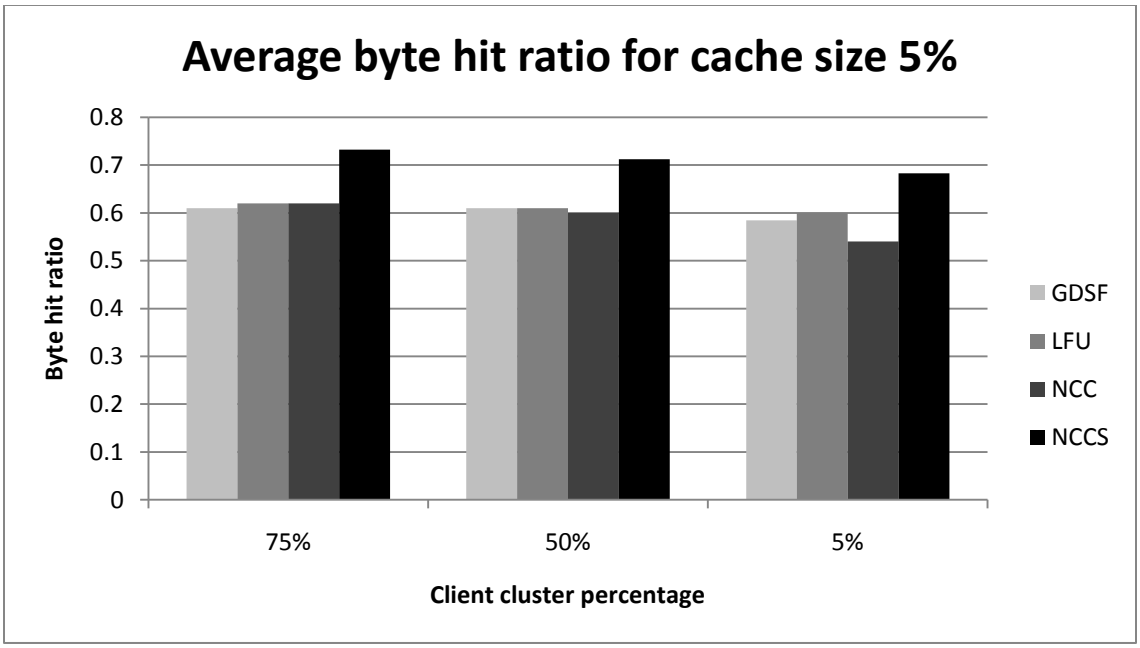

Figure 32: *Average byte hit ratio – Experiments #4-6.*

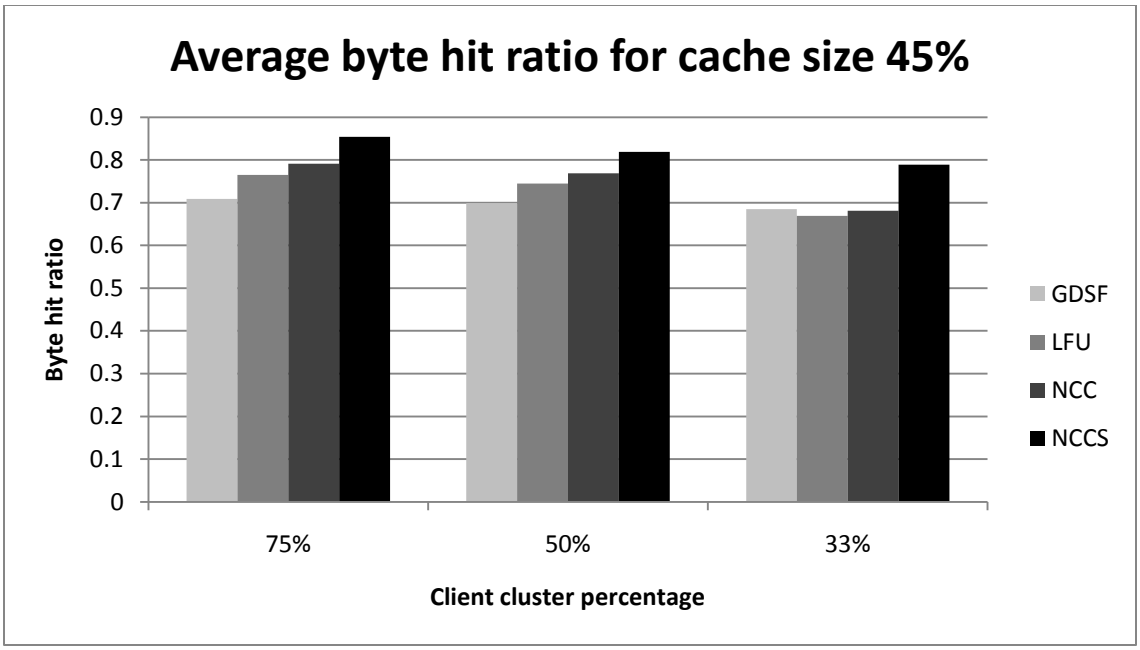

Figure 33: *Average byte hit ratio – Experiments #4-6.*

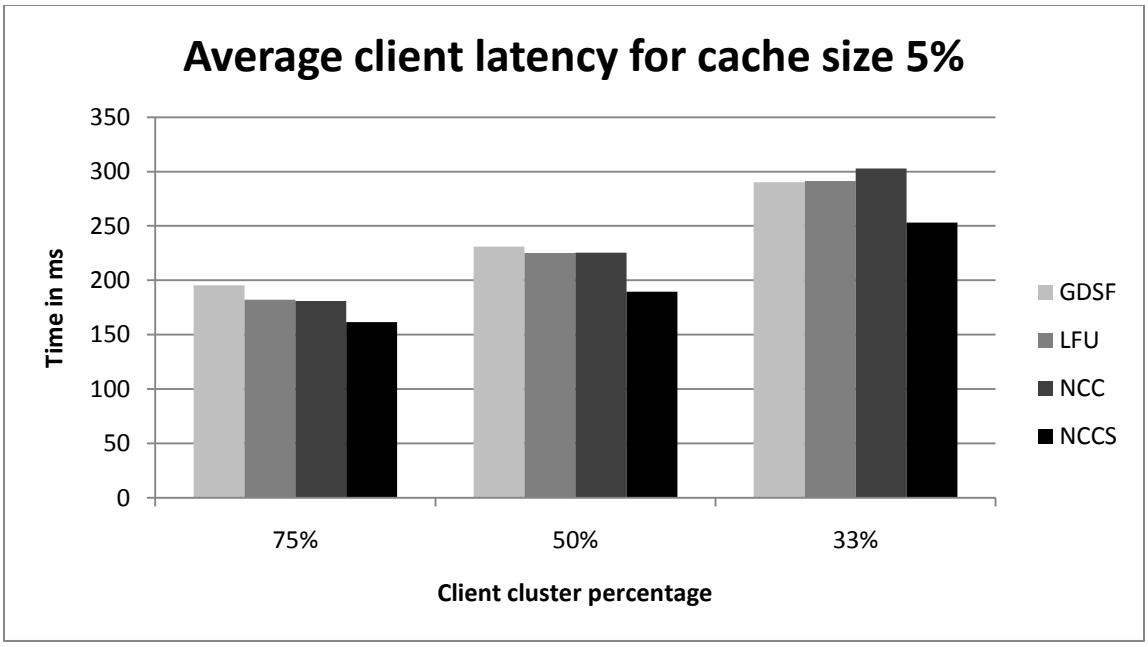

Figure 34: *Average client latency – Experiments #4-6.*

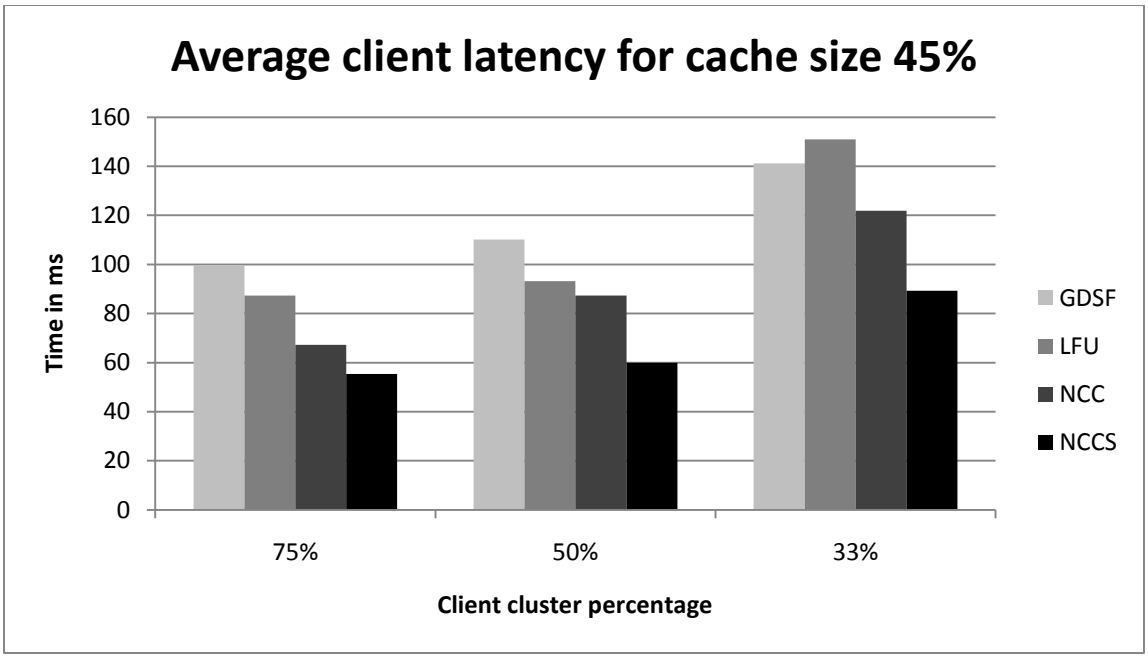

Figure 35: *Average client latency – Experiments #4-6.*

As a side note, we also observed that a cache running on a 2.0 GHz Pentium Processor with 2 GB RAM and 2 MB L2 cache, could create V_{mid} and V_{base} in real time; i.e., around 30 frames per second.

CHAPTER 6

RELATED WORK

Proxy caches have been widely employed to reduce the response time delay and improve the network utilization. Recently, proxy cache has been extended to seamless video services over network. Since it is impossible to store all data at proxy cache, due to storage limitations, it is important to determine which parts of video must be stored at proxy cache. Most proposals for proxy caching mechanisms for video streaming architectures focusing on using the cache buffer and the bandwidth efficiently [10] [18] divide the video stream into blocks. Division of a video stream into blocks and caching of initial block, i.e. *prefix caching* [10], is efficient to allow the client to watch the video immediately. [10] considers an exponential division of a video stream into blocks to reduce the number of blocks. Blocks at the beginning are more important, and they are made smaller in the scheme. Some schemes have also used video frame selection methodology in terms of bandwidth requirements for different options (levels) of video. The caching proxy can dynamically adjust the cached video data by choosing an appropriate level based on network condition or the popularity of the video [13].

Many papers have considered quality adaptation of cache video block at a proxy to support heterogeneous clients in terms of the available bandwidth, end-system performance, and user preferences on the perceived video quality [10]. [10] also considers a layered video encoding algorithm.

Providing cooperation among proxies or clients is also considered in some papers to provide further effective and high quality video streaming services. The research has been mainly focused on studying the benefits of cooperation among proxy caches, designing cache-sharing protocols and cooperative architectures and developing various alternatives for optimizing communication costs and other performance parameters.

Early research on cache replacement policies showed that a straightforward least recently used (LRU) policy was not always the best for maximizing standard performance metrics such as the cache hit ratio. Other work has tried to reduce the latency of retrieving resources by including past retrieval times in a cost metric for each cached resource. Many algorithms have been proposed and found effective for web proxy caching. These algorithms range from simple traditional schemes such as Least-Recently Used (LRU), Least Frequently Used (LFU), First-In-First-Out (FIFO), and various size based algorithms, to complex hybrid algorithms such as LRU-Threshold, which resembles LRU with a size limit on single cache elements, Lowest-Relative Value (LRV), which uses cost, size and last reference to calculate its *utility*, and Greedy Dual which combines locality, size and cost considerations into a single online algorithm. Several studies have proposed and empirically evaluated variants on Greedy Dual that incorporate a combination of long-term popularity and frequency of access.

CHAPTER 7

CONCLUSIONS

Server-Cache architecture has been proposed and implemented, which can disseminate personalized video contents to resource-constrained devices. The proposed video personalization server (VPS) performs automatic video segmentation and video indexing based on semantic video content. A novel caching mechanism, dedicated to cache video segments generated by VPS, implementing the Multi-Dimensional Knapsack Problem [1],[9] (MMKP)-based video personalization strategy to calculate personalized video segments based on client preference and resource constraints has been proposed. The proposed caching mechanism uses an ingenious multi-stage client request algorithm and exploits the commonality of files across clients to use a novel cache replacement mechanism termed as NCCS (Number of Common Clients Size). The multi-stage client request algorithm uses semantic knowledge to group client requests and helps reduce the request load on the VPS as well as increase the efficiency of replacement algorithm. The cache also generates two lower quality versions of each video file. These versions aid in the dissemination of video files to multiple clients entitled to different levels of service based on whether they are paying for the video service or not. Results show that the proposed cache implementation using multi-stage client request aggregation, in conjunction with the proposed NCCS cache replacement policy, outperforms standard cache implementations. Thus, the video-cache architecture is suitable for personalized video dissemination to resource-constrained

mobile devices such as mobile phones, PDAs, Pocket PCS and laptop computers operating in battery mode.

REFERENCES

[1] Akbar M.D., Manning E.G., Shoja G.C. and Khan S., "Heuristic Solutions for the Multiple-Choice Multi-dimension Knapsack Problem", *Proc. Intl. Conf. Computational Science*, 2001, pp. 659-668.

[2] Chattopadhyay S., Luo X., Bhandarkar S. M. and Li K., "FMOE-MR: content-driven multi-resolution MPEG-4 fine-grained scalable layered video encoding", *Proc. ACM Multimedia Computing and Networking Conference (ACM MMCN' 07),* San Jose, California, January 2007.

[3] Chattopadhyay S., Bhandarkar S.M. and Li, K., "Ligne-Claire Video Encoding for Power Constrained Mobile Environments", *Proc. ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 1036-1045.

[4] Chattopadhyay S., Bhandarkar S.M. and Li K., "A Framework for Encoding and Caching of Video for Quality Adaptive Progressive Download", *Proc. ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 775-778.

[5] Cherkasova L., "Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy", HP Laboratories Report No. HPL-98-69R1, April, 1998.

[6] Eickeler S. and Müller S., "Content-based Video Indexing of TV Broadcast News using Hidden Markov models", *Proc. Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Processing*, March 1999, pp. 2997-3000.

[7] Flickner M., Sawhney H., Niblack W., Ashley J., Huang Q., Dom B., Gorkani M., Hafner J., Lee D., Petkovic D., Steele D., and Yanker P., "Query by Image and Video Content: The QBIC System", *IEEE Computer Magazine*, September 1995, pp. 23-32.

[8] Geusebroek J.M., Smeulders A.W.M. and Van De Weijer J., "Fast Anisotropic Gaussian Filtering", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, March 2001.

[9] Hernandez R.P. and Nikitas N.J., "A New Heuristic for Solving the Multichoice Multidimensional Knapsack Problem", *IEEE Trans. System, Man and Cybernetics*, Part A, Vol. 35, No. 5, 2005, pp. 708-717.

[10] J. Song, "Segment-based proxy caching for distributed cooperative media content servers", *ACM SIGOPS Operating Systems Review,* vol. 39, no. 1, Jan. 2005, pp. 22-23

[11] Jin S. and Bestavros A., "Popularity-aware greedy dual-size web proxy caching algorithms", *Proc. 20th IEEE Intl. Conf. Distributed Computing Systems (ICDCS),* Taipei, Taiwan, Apr. 2000, pp. 254-261.

[12] Leacock C. and Chodorow M., "Combining Local Context and WordNet Similarity for Word Sense Identification", *WordNet: An Electronic Lexical Database*, C. Fellbaum (Editor), MIT Press, Cambridge, MA, 1998, pp. 265-283.

[13] Ma W. and Du H. C., "Frame Selection for Dynamic Caching Adjustment in Video Proxy Servers", *ACM Multimedia Tools and Applications,* vol 22, no. 1, Jan. 2004, pp. 53-73

[14] Merialdo B., Lee K.T., Luparello D. and Roudaire J., "Automatic Construction of Personalized TV News Programs", *Proc. ACM Conf. Multimedia*, (Orlando, FL, Sept. 1999), pp. 323-331.

[15] Sikora T., "Trends and perspectives in image and video coding," *Proceedings of the IEEE*, vol. 93, no. 1, Jan. 2005, pp. 6-17.

[16] T. S. Eugene Ng and Hui Zhang, "Towards Global Network Positioning", *Extended Abstract, ACM SIGCOMM Internet Measurement Workshop 2001*, San Francisco, CA, November 2001

[17] T. S. Eugene Ng and Hui Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", *INFOCOM'02*, New York, NY, June 2002

[18] Taniguchi Y., Wakamiya N. and Murata M., "Quality-Aware Cooperative Proxy Caching for Video Streaming Services", *Journal of Networks*, vol. 3, no. 8, Nov. 2008

[19] Tseng B.L. and Smith J.R., "Hierarchical Video Summarization Based on Context Clustering", *Proc. SPIE*, Vol. 5242, Nov. 2003, pp. 14-25.

[20] Tseng B.L., Lin C.Y. and Smith J.R., "Video Personalization and Summarization System for Usage Environment", *Jour. Visual Communication and Image Representation*, Vol. 15, 2004, pp. 370-392.

[21] Vanderbei R. J., "Linear Programming: Foundations and Extensions", *Kluwer Academic Publishers*, 1997.

[22] Wei Y., Bhandarkar S.M. and Li K., "Client-centered Multimedia Content Adaptation", *ACM Trans. Multimedia Computing, Communications and Applications (ACM TOMCCAP)*, in press.

[23] Wei Y., Bhandarkar S.M. and Li K., "Video Personalization in Resource-Constrained Multimedia Environments", *Proc. ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 902-911.

[24] Wei Y., Bhandarkar S.M. and Li K., “Semantics-based Video Indexing Using a Stochastic Modeling Approach”, *Proc. IEEE Intl. Conf. Image Processing (ICIP)*, San Antonio, TX, Sept. 2007.

[25] Wheeler E.S., “Zipf's Law and Why it Works Everywhere”, *Glottometrics*, vol.4, 2002, pp. 45-48.