Chapter One

Introduction

At the beginning the Internet was a medium for text-based applications such as email and file sharing. Recently it has become a tool for major interaction between users and for providing different types of services, including shopping, banking, entertainment, etc. As network technologies improved, network bandwidth increased and service cost decreased causing an increase in the number of Internet users. Such rapid growth causes network congestion and has increased the load on servers, resulting in an increase in the access times of the WWW (World Wide Web) documents [1]. To overcome this situation, caching provides an efficient solution to the latency problem by bringing documents closer to clients.

Caching can be deployed near the server that retrieves resources on behalf of a client from one or more servers or it can be within the client browser. These resources are then returned to the client as though they originated from the proxy server itself to reduce the server load. A proxy server is a computer that is often placed near a gateway and provides a shared cache to a set of clients. All clients send their requests to the proxy regardless of requested service. The proxy can serve these requests using previously cached responses or bring the required documents from the original server. It optionally stores the responses in its cache for future use. One objective of proxy caching is to reduce the amount of external traffic that is transported over the wide-area network mainly from servers to clients. This also reduces the access latency for a document as well as the user’s perceived latency. And because the proxy caches have limited storage it is required to store the popular documents that users tend to request more frequently [2] Caching for streaming di.ffers from caching web objects. The prime aim of caching for streaming is that it aims at decreasing the required transport capacity on the distribution network as much as possible. The dynamicity of the video library results in a different behavior in video popularities. When videos are introduced, they are very popular, and then over time they deteriorate in popularity. As a result traditional web objects are requested more or less uniformly over prolonged periods but a video object is consumed over a relatively short time span. Moreover, video objects are usually much larger than traditional web objects. For these reasons it is very important in video streaming to store the right content at the right time in caches.

A key component of a cache is its replacement policy, which chooses the victim video that will be evicted from the cache to make room for a new video. The best cache replacement algorithm is the algorithm which dynamically selects a suitable subset of videos for caching. It also maximizes the cache hit ratio, which is the fraction of requests served from the cache, by attempting to cache the videos which are most likely to be requested in the future [2] . This project simulates a video service that stores videos in caches where the contents of caches are updated using a number of cache replacement algorithms. By applying the popularity distribution of content, the popularity of videos is determined and popular ones are entered into the cache. When the cache is full, the different replacement algorithms are simulated to choose the video to be evicted from the cache. Finally the simulation calculates output parameter values for comparison.

1.1Research Problem

A significant amount of the web traffic in the Internet is caused by redundant users request for the same content. By using caches some of the redundant user requests are served more quickly and so reduces download latency. However this number of requests being served from caches are not quite enough and there is still a lot of amount of congestion and delay in the Internet. As in Web caching, capacity is the main source of congestion because of the limited size of cache[3].Applying an effective replacement algorithm that is most suitable for video library dynamicity is required to get the best use of the cache limited size. There is a need to evaluate and compare cache replacement algorithms to determine the suitability of each under different variations including the size of the cache, video library and number of user requests.

1.2 Research Objectives

Video is considered a rich media type that causes a significant increase in Internet traffic. Therefore, video caching must be efficient and the employed cache replacement algorithm should be the one that increases the cache hit ratio and reduces the cache misses as much as possible. In order to fulfill the overall goal of this work, the following objectives were set:

1- Evaluate a set of replacement algorithms under different number of videos using video popularities generated by a Zipf distribution to find the most efficient replacement algorithm that works the best for video streaming.
2- Investigate the influence of apply different cache sizes; the goal here is to find the best cache sizes that should be used with each algorithm.

1.3 Research Scope

This work evaluates a number of cache replacement algorithms in term of achieved cache hit ratio with respect to a number of parameters including the size of video library, user request rate and cache size. The study considers video popularity to follow a Zipf distribution. It employs simulation to obtain evaluation results. In this work we choose the three well-known replacement algorithms which are the (Fist In First Out Algorithm (FIFO), Least Recently Used algorithm (LRU) and the Least Frequently Used (LFU)). These algorithms are considered as famous algorithms that are implemented in the vast majority of research work. This enables easy comparison of this work to other related work in literature. We also added the Optimal algorithm and other two algorithm which are designed especially for videos (The Chunk-based Caching algorithm (CC) and Quality-based video Caching algorithm). These algorithms are chosen based on the fact that they have proven very efficient choices for video replacement algorithms. Other cache replacement algorithms that may have a slightly higher or lower performance compared to the algorithms in this work will result in a performance similar to the ones discussed here. We also include the LRU-k algorithm as one example of the LRU improved algorithms, as the LRU and LFU improved algorithms result in marginal improvements over the original LRU and LFU algorithms.

1.4 Programming language and tool

1.4.1 Java

Java is free and Easy to learn object oriented programming language (OOP). It has a rich Application Programming Interface (API) with a great collection of open source libraries. It is suitable for implementing the simulation developed in this work[4].

1.4.2 Microsoft Excel

Microsoft Excel is a spreadsheet developed by Microsoft for Windows, Mac OS X, and iOS. It features calculation, graphing tools and pivot tables[5]. It .was used in this work to organized outputs and generates figures

1.5 Thesis Organization

This thesis is organized as follows: the introduction is in Chapter 1. Chapter 2 is an overview of caching and cache architectures. It also highlights video popularity distributions, reviews of some popular cache replacement algorithms and their implementation and algorithm steps and the related work. Chapter 3 is the research methodology. Chapter 4 demonstrates the implementation of the comparisons of the different replacement algorithms and shows and discusses the results. Chapter 5 summarizes the work and suggests possible directions for future work.

Chapter Two

Caching, Cache Replacement Algorithms and Related Work

2.1 Caching and Video Popularity Distributions

In this section, we explain the concept of caching as well as video popularity distributions.

2.1.1 Introduction

In a video service the popularity of videos decays over time due to the release of new videos. As a result, the contents of caches become less popular and must be updated periodically to maintain the most popular videos. A cache replacement algorithm is the process in charge of selecting an item from the cache to be removed and substituted with a more popular item. The main goal of cache replacements is to maximize the cache hit ratio in order to improve other performance measurements.

2.1.2 Cache Parameters

Here are the basic parameters for cache design:

- Cache hit: an incident where the data is found in the cache.
- Caches miss: an incident where the data is not found in the cache.
- Hit time: time to access the cache.
- Miss penalty: time to move data from server to cache.
- Hit ratio: percentage of times the data is found in the cache.
- Miss ratio: percentage of times the data is not found in the cache.
- Cache block size or cache line size: the amount of data that gets transferred on a cache miss[6].

2.1.3 Caching Architectures

There are different architectures used for caching. The most common ones are proposed by Sarmed AL-Najim in[7]and are: Hierarchical Caching and Distributed Caching. Each of these make the best use of multiple caching as many different caches are connected to each other.

2.1.3.1 Hierarchical Caching

A hierarchical cache has a tree-like structure where similar caches are placed on the same network level, and then connected to another level of caches. In hierarchical caching, the caches are grouped together in a certain level within the network topology. A request from the client is made at the bottom of the hierarchy and the request will first be sent to the cache at the lower level. If the request is found then it is returned to the client. If not, then the request is forwarded to the cache at the higher level of caches. This procedure will be followed until a match is found in one of the caches in the hierarchy. If the requested object is not found then the request is sent to the server. The response will then travel back down the hierarchy leaving the object initially requested at each level and the response will finally reach the client at the bottom of the hierarchy.

Hierarchical Caching reflects what is known as parents and children. As a child cache would forward its request to the parent cache, and if the object requested is not found in the parent cache then the request is forwarded by the parent cache to the server.

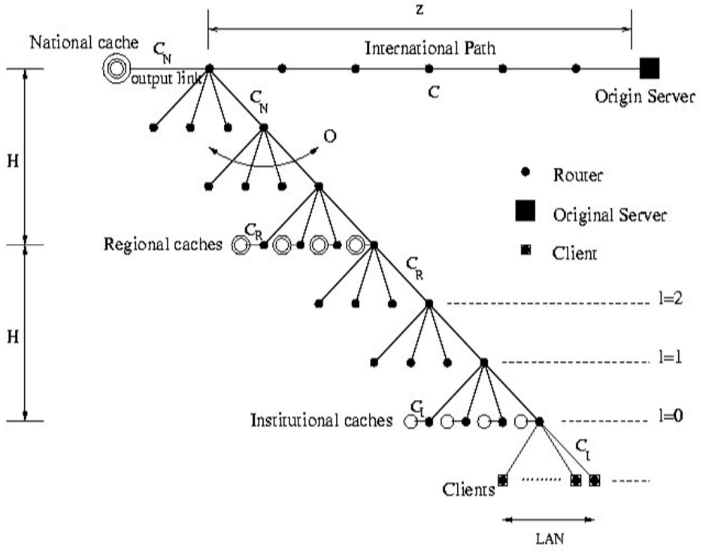

Figure 2-1-1: Hierarchical caching[7].

Figure 2-1-1 above explains the hierarchical cache system. When a request is made at a client browser, the first place to look for the data is in the Institutional caches. If the data is not found there then the higher Regional caches are contacted. If the data is still not in these caches, then the National Cache is contacted, and finally if the objects required are still not found, then the server is contacted by the National Cache and the resulted items are brought back downwards through this route and stored at each level until at the end it reaches the client[7].

The short connection and low bandwidth usage is an advantages of using hierarchical caching. However it is hard to implement, since it is required to configure neighbor caches and cache misses which causes extra delays. In Hierarchical caching the caches in the higher levels must be very efficient and very powerful to produce good performance.

2.1.3.2 Distributed Caching

In distributed caching there is only one level of caches, namely the Institutional level. All the caches in this lowest level communicate with each other and work to serve each other's clients. When a browser makes a request, the data will be looked up in the browsers institutional cache. If the data is not there, then other institutional caches are contacted. Only if the result obtained is still a miss, then the server would be contacted directly. In each Institutional cache there is a meta-data that makes it easier to find the requested data from the huge number of Institutional caches, as it is a directory of all cache contents of other Institutional caches.

In comparison with hierarchical methods, distributed caching does not require additional disk space for Intermediate and Higher level caches[7]. An advantage of Distributed caching is that the data transmission is easy and accurate because there is less traffic congestion in the low-level network. However In the large distributed cache system and when the transmitted data is not from the neighbor cache but from caches over a long distance, the connection time can be quite slow. Therefore sometimes it might be faster to connect to the server directly.

Finally, the two methods (Hierarchical and Distributed Caching) can be combined to create a hybrid caching architecture. This combination gives the best of both methods and improves performance and efficiency.

2.1.4 Video Popularity Distributions

The popularity of videos follows different distributions and the most popular Internet content popularity is the Zipf distribution and Zipf-like Distribution.

Zipf's law is a famous statistical law that is observed in the behavior of many complex systems of different nature. The law is named after Harvard linguistic professor George Kingsley Zipf (1902-1950). It was originally applied to the relationship between words in a text and their frequency of use.

The basic Zipf's law and Zipf-like law govern many features of the WWW such as Web objects access distribution, the number of pages within a site, the number of links to a page and the number of visits to a site[8]It is a . description of the relationship between the frequency of occurrences of an event and its rank, when the events are ranked with respect to the frequency of occurrence. Let the popularity of words used in a given text be denoted by ρ, and their frequency of use be denoted by P, then

$$P \sim \rho{-}\beta$$

With $\beta \approx 1$. More general cases are Zipf-like laws that relate the frequency of symbol use to popularity rank via a power-law relationship.

Applied to the Web, Zipf-like distribution states that the relative probability of a request for the i'th most popular page is proportional to 1/ i^α, for some constant α between 0 and 1. Zipf's Law is considered as a particular case, with 1 = α. In a popularity distribution of objects that conforms to Zipf's Law, the most popular Web object is twice as popular as the second most popular object, and three times as often as the third most frequent object.

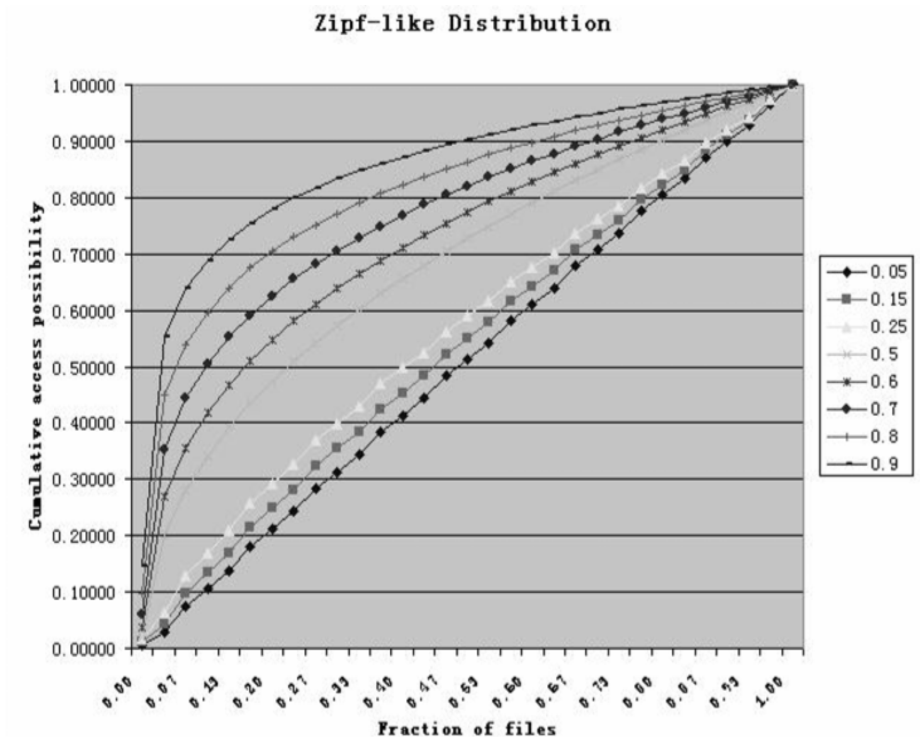

Figure 2-1-2: Zipf-like distributions[3].

Figure 2-1-2 shows a series of Zipf-like distributions with the value of α varying from 0.05 to 1. When 0 = α, it’s a uniform distribution, and objects are receiving equal attention. As α approach 1, popular objects receive greater fraction of requests[3].

2.2 Cache Replacement Algorithms

This section overviews the cache replacement algorithms that are implemented in this work.

2.2.1 Introduction

Caching video objects at proxies close to clients has attracted a lot of attention in recent years. Network based video proxy servers can store the videos in order to minimize initial latency and network traffic significantly. However, due to the limited storage space in video proxy servers, an appropriate video selection method is needed to store the videos which are frequently requested by clients and so cache replacement algorithms are used to evict videos when the cache is full.

2.2.2 Cache Replacement Algorithms

The main goal of cache replacements is to maximize the cache hit ratio in order to improve other performance measurements. Cache replacement algorithms differ in the parameters used to select the item to be evicted from the cache and the way these parameters are applied. Following is an over view of the most popular cache replacement algorithms.

2.2.2.1 First In First Out

First In First Out (FIFO) replacement algorithm always replaces the oldest video. In other words, it replaces the video that has been in the cache for the longest time. Videos are inserted in a queue, with the most recent arrival at the back, and the oldest arrival in the front. When a new video needs to be replaced, the video at the front of the queue (the oldest one) is selected. The FIFO disadvantage is that the oldest videos may be needed again soon, as some important pages may frequently be requested over a long time period. As a result replacing them will cause an immediate Page Fault, and therefore, it is not a very effective algorithm However, it is useful too consider it in our work for comparison purposes. The FIFO cache replacement algorithm steps are shown in Figure 2-2-1.

```
Repeat
        IF (queue (cache) in not full)
                    {Insert video at the end of the queue}
        Else
                    {Delete the video at the front of the queue
                     Insert video at the end of the queue
                     Increment fault}
Until
        End of all requests
Output the number of fault
```

Figure 2-2-1: The FIFO cache replacement algorithm steps

Figure 2-2-2 shows the implementation of the FIFO replacement algorithm. The figure shows the numbers of page faults for a given set of items. In the figure we have data of 15 video request and a cache size=3 presented as F1, F2 and F3 for every data, the appearance of the (*) symbol denotes that a miss accrues.

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
F2		0	0	0	0	3	3	3	2	2	2	2	2	1	1
F3			1	1	1	1	0	0	0	3	3	3	3	3	2
	*	*	*	*		*	*	*	*	*	*			*	*

FIFO number of misses = 12

Figure 2-2-2: the implementation of the FIFO replacement algorithm

2.2.2.2 Least Recently Used

The Least Recently Used (LRU) algorithm replaces the least recently used items first. It requires keeping track of which items was used and when, and it is costly to make sure that the algorithm always discards the least recently used item. General implementation of this technique requires keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes[9].

To fully implement LRU, it is necessary to maintain a linked list of all items in the cache, with the most recently used item at the front and the least recently used item at the rear. The difficulty is that the list must be updated on every item reference. Finding an item in the list, deleting it, and then moving it to the front is a very time consuming operation[10].

One advantage of the LRU algorithm is that it is amenable to full statistical analysis. On the other hand, LRU's weakness is that its performance tends to degrade under many common reference patterns. For example, if there are N pages in the LRU pool, an application executing a loop over an array of N + 1 pages will cause a page fault on each and every access[10]. The LRU cache replacement algorithm steps are shown in Figure 2-2-3.

```
Repeat
    IF (current requested item is in cache)
            Get its index
            Count to zero (indicate it is used very recently, higher the count of the most least recently used item)
    Else
        IF (cache is full)
            Get item with maximum count (LRU item)
            Replace it with new item
            Reset count to zero
            Increment fault
        Else
            Add new item to end of cache
            Increment the fault
            Increment top of cache
    Increment all the counts
Until
    End of all requests
Output the number of faults
```

Figure 2-2-3: The LRU cache replacement algorithm steps

Figure 2-2-4 shows the implementation of the LRU replacement algorithm showing the number of page faults for a given set of items

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1
F2		0	0	0	0	0	0	0	3	3	3	3	3	3	3
F3			1	1	1	3	3	3	2	2	2	2	2	2	2
	*	*	*	*		*		*	*	*	*			*	

LRU number of misses = 10

Figure 2-2-4: The implementation of the LRU replacement algorithm

2.2.2.3 Least Frequently Used

Least Frequently Used (LFU) is a famous cache replacement algorithm. The standard characteristic of LFU is to track the number of times a video is referenced. When the cache is full the algorithm will evict the video with the lowest reference frequency.

A simple method to employ an LFU algorithm is to assign a counter to every video that is loaded into the cache. Each time a reference is made to that video the counter is increased by one. When there is a new video waiting to be inserted and the cache is full, the system will search for the video with the lowest counter and remove it from the cache. The LFU algorithm may seem like an intuitive method. However in a scenario where a video is referenced repeatedly for a short period of time and is not accessed again for an extended period of time, due to how rapidly it was accessed its counter increases drastically even though it will not be used again for a decent amount of time. This leaves other videos which may actually be used more frequently susceptible to eviction simply because they were accessed through a different method[3]t to being Also, new videos that just entered the cache are subjec . removed very soon because they start with a low counter, even though they .might be used very frequently after that The LFU cache replacement algorithm steps are shown in Figure 2-2-5.

```
Take inputs
Initialize Frame and Frequent array to -1
IF   (page miss)
        {Find the least frequently used page from the pages in
FRAME.
                Replace page in frame by current page.
                Create array of page counts and store it in 'count' array}
Increment counter
Print FRAME
```

Figure 2-2-5: The LFU cache replacement algorithm steps

Figure 2-2-6 show the implementation of the LFU replacement algorithm. It shows the number of page faults for a given set of items

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1	7	7	7	2	2	3	3	4	2	2	2	2	2	1	2
F2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
F3			1	1	1	1	1	1	1	3	3	3	3	3	3
	*	*	*	*		*		*	*	*				*	*

LFU number of misses = 10

Figure 2-2-6: The implementation of the LFU replacement algorithm

2.2.2.4 The Optimal Algorithm

The Optimal Page Replacement Algorithm is also known as OPT or MIN. In this algorithm, the video that will not be used for the longest period of time in the future is replaced. It involves the knowledge of future requests to predict which item in the cache will be needed again. The Optimal algorithm has the lowest page fault rate, but it is difficult to implement because it needs knowledge of future requests[12].

The Optimal cache replacement algorithm steps are shown in Figure 2-2-7.

```
Take array n of videos
Initialize fault and cache array to -1
IF   (cache miss)
        IF (cache is full)
              {-Search array n of videos to find the video that will not be used for
                the longest period of time.
                -Replace that video by current video.      }
       Else
             {Insert video to cache                         }
       Increase fault
Output number of faults
```

Figure 2-2-7: The Optimal cache replacement algorithm steps.

Figure 2-2-8 show the implementation of the Optimal replacement algorithm and the number of page faults for a given set of items.

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2
F2		0	0	0	0	0	0	4	4	4	0	0	0	1	1
F3			1	1	1	3	3	3	3	3	3	3	3	3	3
	*	*	*	*		*		*			*			*	

Optimal number of misses = 8

Figure 2-2-8: The implementation of the Optimal replacement algorithm

2.2.2.5 The LRU-K algorithm

The LRU-K page-replacement algorithm is derived from the classical Least Recently Used (LRU). It incorporates both recently and frequency information when making replacement decisions. Since the LRU buffering algorithm drops the page from the buffer that has not been accessed for the longest time when a new buffer is needed, it limits itself to only the time of the last reference. Specifically, LRU does not discriminate well between frequently and infrequently referenced pages until the system has wasted a lot of resources keeping infrequently referenced pages in the buffer for an extended period. It was proven that LRU-K is essentially optimal among all replacement algorithms that are solely based on stochastic information about past references[3].

The basic idea of LRU-K is to keep track of the times of the last K references to popular pages, using this information to statistically estimate the inter-arrival time of such references on a page-by-page basis.

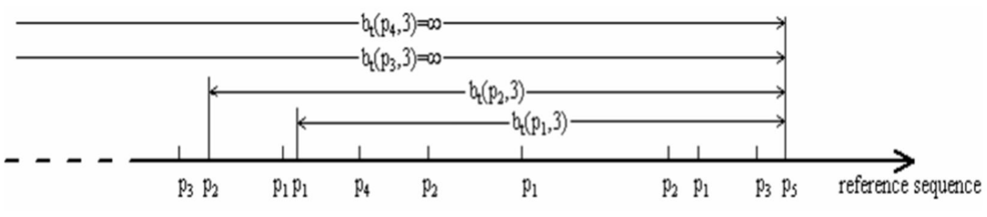

Figure 2-2-9: A simplified example of backward K-distance (K=3)[3].

Figure 2-2-9 shows a simplified example of LRU-3 for a sequence of accesses to pages p1, p2, …, pn. When a request for an absent page p5 arrives and the buffer is full, a victim is chosen based on the backward K-distance from the point of the new access. In the case of this example, both p3 and p4 have the backward K-distance of infinity, so a subsidiary policy is needed to break the tie[3].

```
LRU-K: on request for object p at time t

/* scan cache queue to see if p is already in cache */
q := the object at queue end
hit := false
While (q != null) do
        If (q.url equals p.url) then  // hit
                hit := true
                break
        Endif
        q := next object before q
Enddo
If (hit) then   // hit
        /* update history information of p */
```

```
            If (t-HIST(p,1)> Correlation_Timeout) then // a new,
uncorrelated reference
                              For i =2 to K do
                                         HIST(p,i) = HIST(p,i-1)
                              Endfor
                               HIST(p,1) = t
            Else    // a correlated reference
                               HIST(p,1) = t
            Endif
           hits += 1
Else    // miss
          /* select replacement victims */
       q = the object at the Cache Queue end
        While (Free Space < p.size) do
                   If (t-HIST(q,1) > Correlation_Timeout) then // eligible for
replacement
                             evict victim q from cache
                              Free Space += q.size
                              put HIST(q) into the Evict Table
                   Endif
                   q = next object before q  // object with next max Backward
K-distance
        Enddo
        /* cache the referenced object*/
        fetch p into the cache and append p at the end of Cache Queue
        Free Space -= p.size
         misses += 1
        check the Evict Table for object p
        If (p does not exist) then   // initialize history control block
                allocate HIST(p)
                For i := 2 to K do HIST(p,i) := 0
        Else
                retrieve stored HIST(p)
                For i = 2 to K do HIST(p,i) = HIST(p,i-1)
        Endif
        HIST(p,1) = t
Endif
/* Relocate p in the cache queue with its Backward K-distance and
HIST(p,1)*/
q := next object before p
While (q != NULL && HIST(q,K)
 HIST(p,K)) do
        q := next object before q
Enddo
If (q == NULL) then move p to Cache Queue top
Else move p into the position after q
```

Figure 2-2-10: The LRU-k replacement algorithm steps[3].

Figure 2-2-11 shows the implementation of the LRU-2 replacement algorithm.

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1			1	1	1	3	3	3	2	2	2	2	2	2	2
F2		0	0	0	0	0	0	0	0	0	0	0	0	1	1
F3	7	7	7	2	2	2	2	4	4	3	3	3	3	3	3
	*	*	*	*		*		*	*	*				*	

LRU-2 number of misses = 9

Figure 2-2-11: The implementation of the LRU-2 replacement algorithm.

2.2.2.6 The Chunk-based Caching algorithm (CC)

This algorithm is specifically for streaming video taking into account the dynamicity of the library. The ranking of the algorithm follows the dynamicity of the library (better than traditional algorithms). In addition the algorithm segments each video into chunks and proposes a new algorithm to rank these chunks. After comparing the performance of caching based on this new ranking algorithm with traditional caching algorithms, it is apparent that chunking is most beneficial[13].

Following is a full description of the algorithm as proposed by Dohy Hong[13] -:

The caching algorithm is based on two principles:

1) Scoring videos based on requests for them

2) Segmenting the videos in chunks.

The chunk m+1 of a given video will be requested with a high probability in the near future if chunk m of that video is currently streamed to some user. First let`s consider a simplified version of the caching algorithm without chunking, which makes decisions by ranking videos in their entirety. For the full version of the caching algorithm all videos are segmented in chunks of equal duration, each chunk has a different ranking.

2.2.2.6.1 The simplified version of chunk-based caching algorithm

The simplified version of the algorithm is based on keeping a score Sk for each video k (k=1, 2, …, K). When a new video is requested for the first time its score is initialized to a value B. And every time video k is requested, it score increases by an amount A, and the score of all other videos is decreased by 1. The algorithm re-ranks the videos at each request time based on these scores Sk and the first L ranked videos are cached. At each request one of the following three events can occur

1. The requested video is already found in the cache (a cache hit). The caching algorithm updates the ranking, and no videos are evicted from the cache.

2. The requested video is not stored in the cache and this request means that this video gets upgraded to a rank in the first L positions. Thus the caching algorithm decides to cache the video. The server copies the video into the cache.

3. The requested video did not reside in the cache and it has a rank larger than L. Thus the caching algorithm decides that the requested video does not need to be cached this time. The ranking is updated. The video is served from the origin server.

If two videos have equal Sk values, the video with the lowest k value takes precedence.

2.2.2.6.2 The full version of the Chunk-based Caching algorithm

The full version of the caching algorithm also maintains the values Sk in the same way as described in the simplified version. Each video is segmented in M chunks and each chunk inherits the score Sk from the video it belongs to. For each chunk m of video k a value Nk,m is maintained that accumulates the number of guaranteed hits this chunk will have, knowing which videos are currently watched by the users and assuming that no user aborts watching a video.

Figure 2-2-12 illustrates that the value Nk,m indicates how many times that particular chunk m of video k, will be consumed in the near future (given the current user behavior). This counter Nk,m is maintained as follows:

1. The values Nk,m are increased by 1 for all values of the index m, each time video object k is requested by a user.

2. The value of Nk,m is decreased by 1 after a user watching video object k has consumed chunk m.

3. Note that if before the end of the video object k a user aborts viewing the video (or uses other trick- play commands like “rewind” or “fast rewind”), the values Nk,m need to be updated accordingly. However, the “abort”, “rewind” or “fast rewind” events do not occur in this simulation.

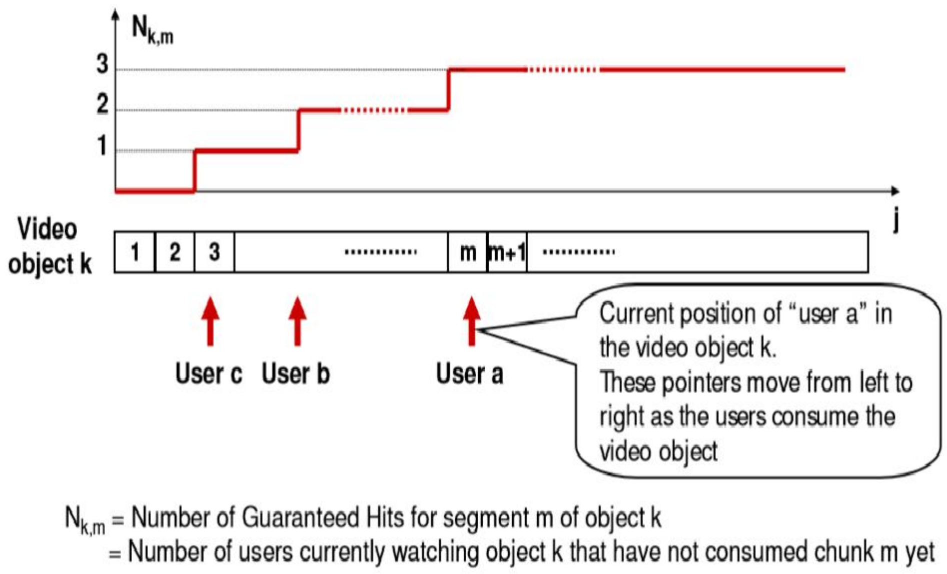
$N_{k,m}$
3
2
1
j
Video object k
1 2 3 ………… m m+1 …………
User c
User b
User a
Current position of "user a" in the video object k. These pointers move from left to right as the users consume the video object
$N_{k,m}$ = Number of Guaranteed Hits for segment m of object k = Number of users currently watching object k that have not consumed chunk m yet

Figure 2-2-12: Maintaining Nk,m for video K [13].

The full version of chunk-based caching operates in a similar way as the simplified version: at each request time for a chunk, one of the three types of events occurs (i.e., a cache hit, a cache miss combined with a cache update or a cache miss without a cache update). In the full version the ranking is based on comparing the values Nk,m. (the higher the value Nk,m the higher the rank of the chunk (k,m)) and the values Sk are used only as tie-breakers. If after ranking chunks based on both Nk,m and Sk there is still a tie, chunks are ranked based on their chunk number.

Figure 2-2-13 show the first part of the CC cache replacement algorithm steps. The input is the video and the outputs are each video with its score SK and its number of guaranteed hits NoT which will be the input for the second part. The output for the second part is the number of hits. As shown in figure 2-2-14. Having n=number of videos, m=number of chunks in each video, SK =score for each video, NoT = Number of guaranteed hits for each chunk.

```
Input Video V
If V mod m=0
        $SK_v = SK_v + a$
        For (i=0 to m)
                $S_{K (v + i)} = SK_v$
                $NoH_v = NoH_v+1$
        For (i=0 to n)
                If (I mod SV =0 && i != V)
                        $SK_v = SK_v - 1$
        Else
            $NoH_v = NoH_v-1$
            For (i=0 to m)
                    If (V mod m=i)
                    S=i
            $SK_v = SK_{(v-s)}$
```

Figure 2-2-13: The Chunk-based Caching algorithm steps for scoring

```
Repeat
  Input video
  If (current video is in cache)
      Update SK, NoH

      If (cache is full)
          Get video with the minimum (NoT) and it compare with currentVideo NoT
          If (currentVideo NoT is greater) then replace the video with the minimum (NoT)
          If (currentVideo NoT is smaller) then no change
          If (currentVideo NoT equal to it) then compare SK value for the 2 Videos
          If (currentVideo SK is greater) then replace the video with the minimum (NoT)
          If (currentVideo SK is smaller) then no change
          If (currentVideo SK equal to it) then put the video with the higher chunk number in
cache
          Page-Fault++
      Else
          Add video to cache
          Page-Fault++
Until end of all requests
Output Page-Fault
```

Figure 2-2-14: The Chunk-based Caching algorithm steps for a number of guaranteed hits

Figure 2-2-15 show the implementation of the Chunk-based Caching replacement algorithm for a given set of items.

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1			1	1	1	1	1	1	1	3	3	3	3	3	3
F2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
F3	7	7	7	2	2	2	2	2	2	2	2	2	2	1	1
	*	*	*	*		*		*		*				*	*

The CC number of misses = 9

Figure 2-2-15: The implementation of the Chunk-based Caching replacement algorithm

2.2.2.7 Quality-based video Caching algorithm

As proposed be Stefan Podlipnig[14]Quality based caching is some sort of , partial caching. Here are two forms to enhance the quality. First form is Quality reduction, where the proxy reduces the quality, which allows simple eplacement strategies. The srecond form is the Quality adaptation, where the proxy reduces and enhances the quality. Although quality adaptation is seen as the more flexible approach it introduces additional complexity. To enhance the quality of a reduced video a cache has to reload specific parts of the video. Furthermore the cache has to implement intelligent adaptive behavior. Because the Quality reduction supports the fact that most of the videos will have a short period of high popularity followed by a decreased popularity, and quality reduction can be coupled with explicit reloading of videos, i.e. a user can trigger a reload if he is not satisfied. Quality reduction can be an effective alternative to complex adaptive behavior.

2.2.2.7.1 Quality Reduction

Quality based video allows quality adaptation. A video should have a number of quality steps that can be obtained through operations on that video. Such quality steps can be realized through layers (base layer, enhancement layers…). Also in quality based there exists a metadata describing the possible quality steps. For each quality step the metadata describes the corresponding operation, such as the resulting size and the resulting quality. The size si and quality factor qi of a video i are in the range 0 < si, qi <= 1.

2.2.2.7.2 Replacement

The following are two types of replacements that are used in the Quality based video caching algorithm

2.2.2.7.2.1 Replacement with repositioning

A quality based replacement strategy chooses the last video and reduces its quality by deleting one quality step. The video will be deleted if the video has only one quality step left. Otherwise the video stays in the cache and is repositioned in the cache list. Then the video at the end of the list is chosen and the above procedure is repeated. For LRU the proposed calculation is modified to incorporate resulting quality and position numbers rather than time. This algorithm is called LRU-R.

2.2.2.7.2.2 Replacement without repositioning

Without a weighted access the last video in the list is chosen for quality reduction successively until it is deleted or the replacement stops. The last video will be deleted in the following replacement round if it is not requested immediately. This behavior is called the vertical replacement. The quality steps of one video are ordered from the top to the bottom. To overcome the strong similarity to the underlying strategy horizontal replacement is proposed. In horizontal replacement the highest layer of all videos is first removed, then the next layer and so on. Furthermore a combination of these strategies is proposed in [14]. This combined replacement with the horizontal pattern is used to remove the upper layers and the vertical pattern to remove the lower layers. The three pattern of replacement are illustrated in figure 2-2-16.

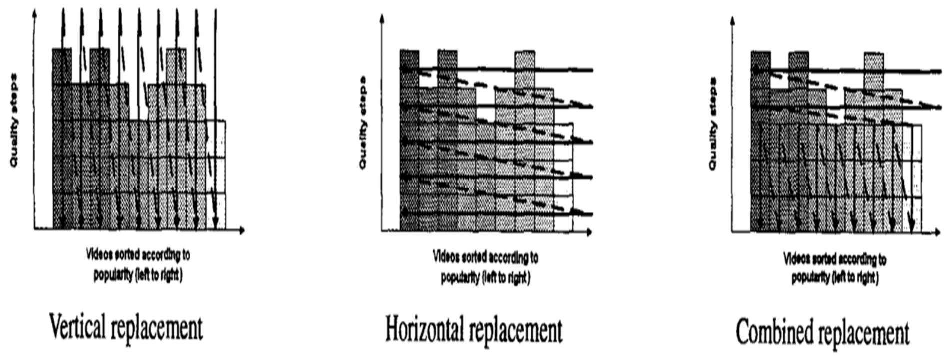
Quality steps
Videos sorted according to popularity (left to right)
Vertical replacement
Quality steps
Videos sorted according to popularity (left to right)
Horizontal replacement
Quality steps
Videos sorted according to popularity (left to right)
Combined replacement

Figure 2-2-16: Replacement patterns of Quality-based video Caching[14].

A given pattern is used in each replacement run. Different videos can have a different number of quality steps. The replacement algorithm tries to follow the given pattern. It is like a matrix traversal where the dimensions are given by the number of videos and the maximum number of quality steps. Note that these patterns can be combined with any original replacement algorithm. The only condition is that the videos are sorted according to their popularity. The popularity can be determined by different video characteristics (for example request recently, request frequency, bitrate, and size).

Figure 2-2-17 show the implementation of Quality-based video Caching replacement Algorithm for a given set of items.

Video reference stream 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2

F1	7	7	7	7	7	7	7	4	4	4	4	4	4	4	2
F2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
F3			1	1	1	1	1	1	1	1	1	3	3	3	3
	*	*	*	*		*		*	*	*		*	*	*	*

QC number of misses = 12

Figure 2-2-17: The implementation of the Quality-based video Caching algorithm

2.3 Related Work

Many researchers have shown interest in web caching as a significant strategy for fast access of formerly retrieved data. We select the main studies in literature that compare web caching replacement algorithms.

2.3.1 Cache Line Replacement Algorithms for Embedded Systems

A study proposed by Gille Damien[15] was accomplished to find the efficient replacement policy in embedded systems. Polices that had been compared are (1-bit, LRU, Modified Pseudo LRU (MRLRU), MRU based Pseudo LRU (PLRUm), Tree-based Pseudo LRU (PLRUt), Random, Round Robin, SIDE). A cache simulator in the study was implemented in a way that allowed applying a detailed investigation of the policies' behavior. The Least Recently Used (LRU) strategy performs well on most memory patterns and that is because of the expense of the hardware requirements and of the power consumption. This work was to evaluate the evaluation the performance of the new algorithms that had been developed. The fast running time of the simulator allowed dealing with numerous replacement proposals across a broad range of embedded applications.

The study results show that the MRU-based pseudo-LRU replacement policy (PLRUm) outperform the LRU algorithm in the low hardware and power consumption requirements.

2.3.2 Performance Improvement of Web caching Algorithms

As proposed by by Dhawaleswar Rao [16], a Response Time Gain Factor (RTGF) is included in this web object replacement algorithm with different sizes for web objects to improve the response speed. The study evaluates the performance by establishing an experimental model that has two kinds of object reference characteristics. The study measured the response time, object-hit ratio, the average object-hit ratio, and evaluated them by comparing the three algorithms LRU, LFU and SIZE algorithm with the proposed algorithm.

The Response Time Gain Factor had been calculated as follows:

Response Time Gain Factor (RTGF) = ((Time Without cache–Time With cache) x 100/ (Time Without cache)

This factor is designed for the average response time gain. This factor gives the amount of advantage in web cache response time.

The results of the study were variable because they depend on the traffic of the network and the diverse object reference characteristics. Further studies can be on the operation method of the cache that considers this diversity dynamically and the division-ratio of storage scope.

2.3.3 Page Replacement Algorithms

Anvita Saxena in [17] compare the page replacement algorithms for virtual memory systems. The researchers consider the traditional algorithms such as Optimal replacement, LRU, FIFO and also study the recent approaches such as Aging, Adaptive Replacement Cache (ARC), CLOCK with Adaptive Replacement (CAR). The study uses a two- level memory hierarchy each consists of a faster and costlier main memory and a slower and cheaper secondary memory.

The results show that CAR and ARC algorithms outperform the basic CLOCK and are promising algorithms. Studies have shown that the benefits of the Page replacement are real although it plays a small part in the performance of applications. They recommend evaluating the implementations of both CAR and ARC in real operating systems.

.

Chapter Three

Research Methodology

The simulation in our work evaluates the replacement algorithms to find the algorithm that works best in video caching.

3.1 Evaluation Model

In our work we generate video requests where requests are used as an input to each replacement algorithm. The output of our model is the hit ratio for each algorithm as shown in figure 3-1.

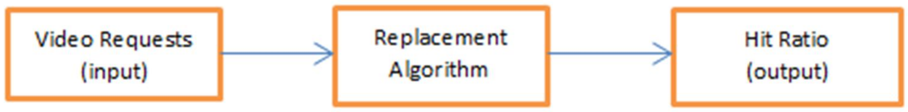

Figure 3-1: Simple view of the model

3.2 Input Data

In our model the input data is a series of requests that are generated randomly using the Zipf distribution. We apply the equation in [3] to define the popularity Pi of the ith object in the rank following a Zipf distribution by:

$$P_N(i) = \frac{\Omega}{i^{\alpha}} \qquad (3\text{-}1)$$

$$\Omega = \frac{1}{\sum_{i=}^{n} \frac{1}{i^{\alpha}}} \qquad (3\text{-}2)$$

Equation (3-1) is used to generate the popularity of videos where the harmonic value (skewness) of the Zipf distribution α =0.75.

The generated video requests are used as an input for the replacement algorithms. To evaluate the replacement algorithms, the same data are input to each algorithm with a specific cache size to compare the algorithms and find the one with the highest hit ratio.

3.3 Replacement Algorithm Flowchart

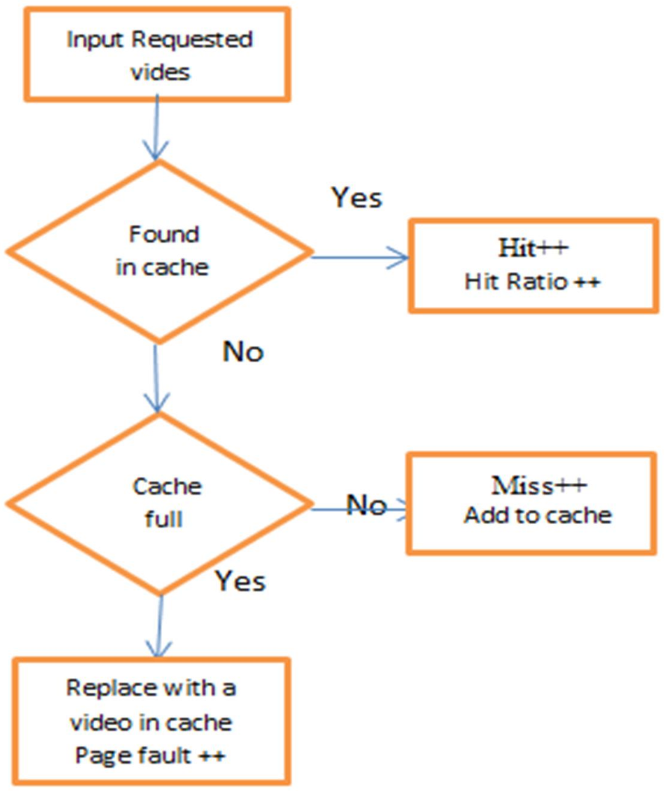

Figure 3-2: Flowchart for replacement algorithms

Figure 3-2 shows the simplified flow chart for all replacement algorithms. The requested video is searched in the cache. If the video is found in the cache the number of hits will increase by one. Otherwise, a miss will occur and the video will be inserted in the cache if the cache is not full. If no place is available in the cache, a video is evicted from the cache (victim video) and replace with the newly requested video. The selection of the evicted video depends on the replacement algorithm. In this work, we develop seven simulation programs for the seven replacement algorithms using java programming language. These simulations are run and the output is the cache hit ratio for each algorithm of the seven algorithms. In our evaluation we also consider the cache size and how it affects the hit ratio. The model is run using different cache sizes to evaluate how each algorithm performs having a small cache size and under large cache sizes.

3.4 Cache Replacement Algorithms

The simulation calculates output parameter values for comparison. The algorithms that are used in this work are Fist In First Out Algorithm (FIFO), Least Recently Used algorithm (LRU), the Least Frequently Used (LFU)),Optimal algorithm (OPT), Chunk-based Caching algorithm (CC) and Quality-based video Caching algorithm (QC).

Chapter Four

Implementation and Results

4.1 Simulation Results and Analysis

A separate simulation code for each of the seven replacement algorithms have been written using java programming language. Input data are randomly generated numbers attained by the zipf distribution that represents video requests. After running the codes the output is a hit ratio for each algorithm of the seven algorithms.

The simulation is run using different numbers of video requests ranging from small values of video request as 200 requests to large values of video requests (2000 and 5000) requests. Each value for total requests is run with different cache sizes.

4.1.1 The hit ratio for 200 video requests with different cache sizes

Observing Table 4-1 and figure 4-1 we can clearly see that as we increase the size of the cache, the hit ratio increases and it is clearly appears that the LRU-2 algorithm and the LFU have a higher hit ratio than the LRU and the FIFO algorithm.

Table 4-1: Hit Ratio for LRU-2, LRU, LFU and FIFO algorithms using different cache sizes

ALGORITHEM	C-Size 10	C-Size 30	C-Size 50
LRU-2	0.64	0.705	0.73

LRU	0.61	0.705	0.73
LFU	0.66	0.705	0.73
FIFO	0.58	0.67	0.725

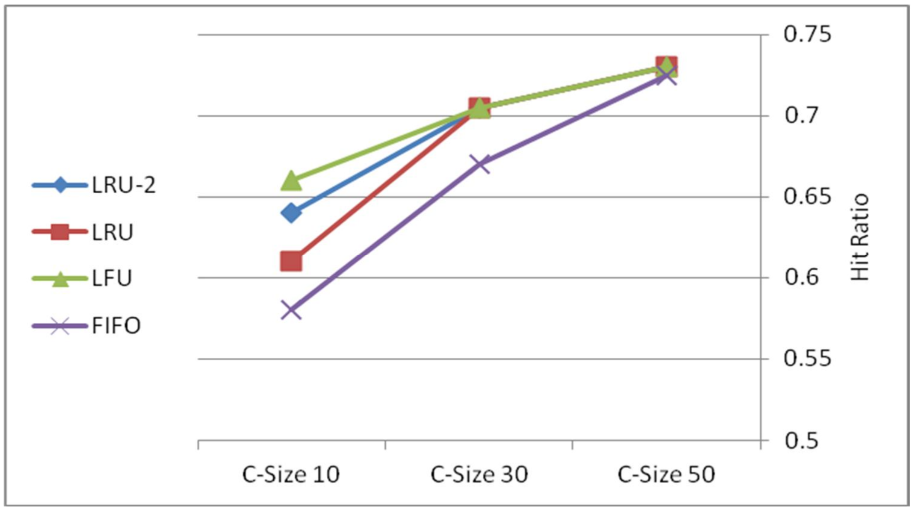

Figure 4-1: Hit Ratio for LRU, LRU-2, LFU and FIFO algorithms using different cache sizes

Following, we compare the other three replacement algorithms OPT, CC and QC. Figure 4-2 shows that the CC algorithm has the highest hit ratio, compared to other algorithms, approaching 0.73 even when the cache size is small. As the cache size reaches 50, all algorithms saturate at a hit ratio of over 0.7, that's can be seen in Table 4-2.

Table 4-2: Hit Ratio for OPT, CC and QC algorithms using different cache sizes

ALGORITHEM	C-Size 10	C-Size 30	C-Size 50
OPT	0.7	0.73	0.73

CC	0.725	0.73	0.73
QC	0.655	0.705	0.73

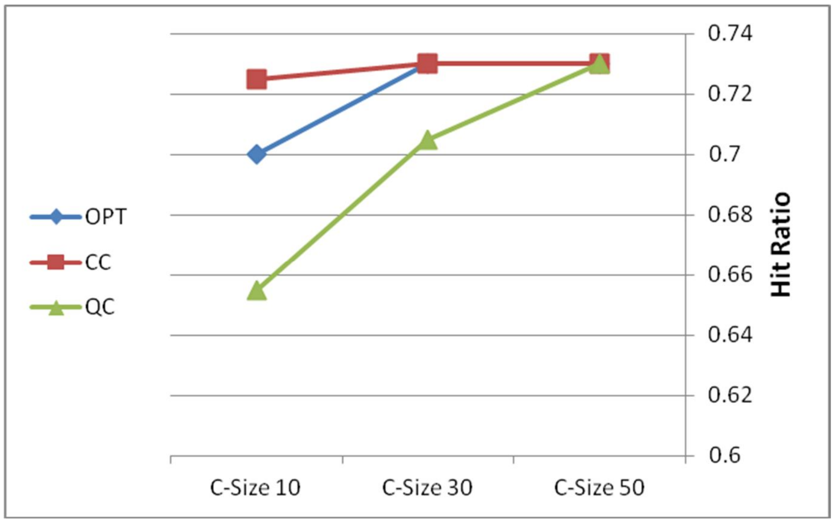

Figure 4-2: Hit Ratio for OPT, CC and QC algorithms using different cache sizes

Another three replacement algorithms QC, LRU-2 and LRU are selected for comparison. We found that the QC algorithm has a higher hit ratio than the LRU-2 and both have a better hit ratio than the third algorithm (the LRU algorithm). Figure 4-3 shows the outcome of the comparison. Table 4-3 shows the exact values of hit ratio for the algorithms.

Table 4-3: Hit Ratio for QC, LRU-2 and LRU algorithms using different cache sizes

ALGORITHEM	C-Size 10	C-Size 30	C-Size 50
QC	0.655	0.705	0.73
LRU-2	0.64	0.705	0.73

LRU	0.61	0.705	0.73

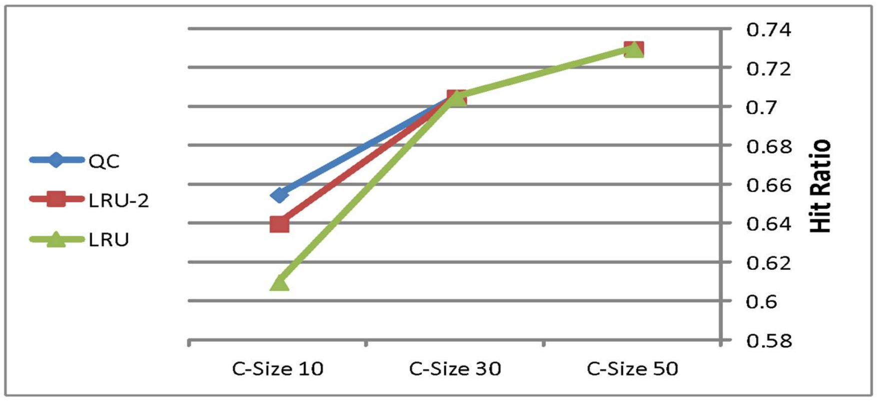

Figure 4-3: Hit Ratio for QC, LRU-2 and LRU algorithms using different cache sizes

Below in figure 4-4 and Table 4-4 we present a comparison of all evaluated algorithms (the seven algorithms).

Table 4.4: Hit ratio for all evaluated algorithms

ALGORITHEM	C-Size 10	C-Size 30	C-Size 50
OPT	0.7	0.73	0.73
CC	0.725	0.73	0.73
QC	0.655	0.705	0.73
LRU-2	0.64	0.705	0.73
LRU	0.61	0.705	0.73
LFU	0.66	0.705	0.73
FIFO	0.58	0.67	0.725

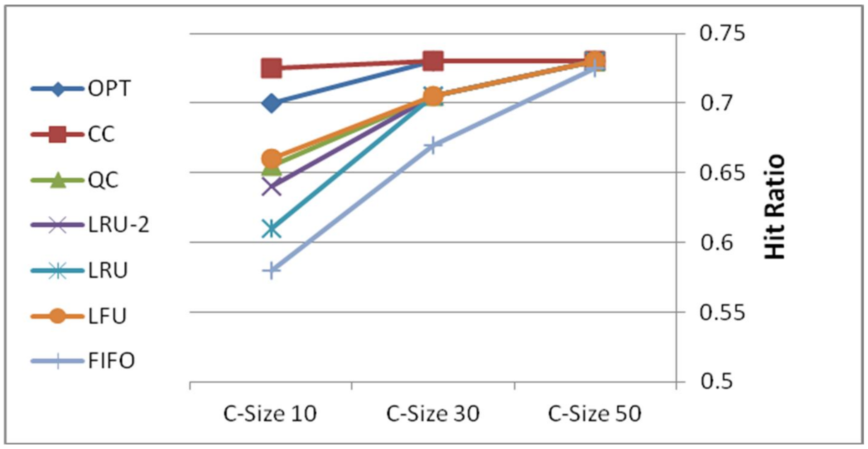

Figure 4-4: Hit ratio for all evaluated algorithms

Looking at figure 4-4 and Table 4-4 we can state the following:

- The CC algorithm outperforms all other algorithms followed by the OPT algorithm.
- The QC algorithm has a lower hit ratio compared to CC and OPT, but the hit ratio becomes similar to the others when the cache size increases to 50.
- Following in term of hit ratio are the LFU, LRU and LRU-2. These algorithms cannot be ranked in a specific order, as the difference in the values of hit ratio is marginal. One algorithm may slightly precede the others under certain conditions and achieve a slightly lower hit ratio under other conditions. The resulting hit ratio depends on the popularity of videos and the number of requested videos.
- The algorithm with the smallest hit ration is the FIFO.

4.1.2 The hit ratio for 2000 video requests with different cache sizes

Here we evaluate all seven algorithms using an input data of 2000 video requests with different cache sizes (50, 120, 300 and 450). The resulting hit ratios for the algorithms are shown in Table 4-5 and presented in figure 4-5 and 4-6.

Table 4-5: Hit ratios of different replacement algorithms for 2000 video request

ALGORITHEM	C-Size 50	C-Size 150	C-Size 300	C-Size 450
OPT	0.697	0.7135	0.7135	0.7135
CC	0.7085	0.7135	0.7135	0.7135
QC	0.5905	0.651	0.6875	0.7105
LRU-2	0.6115	0.658	0.6935	0.7075
LRU	0.582	0.6485	0.6915	0.7095
LFU	0.655	0.6565	0.684	0.706
FIFO	0.5655	0.634	0.6765	0.702

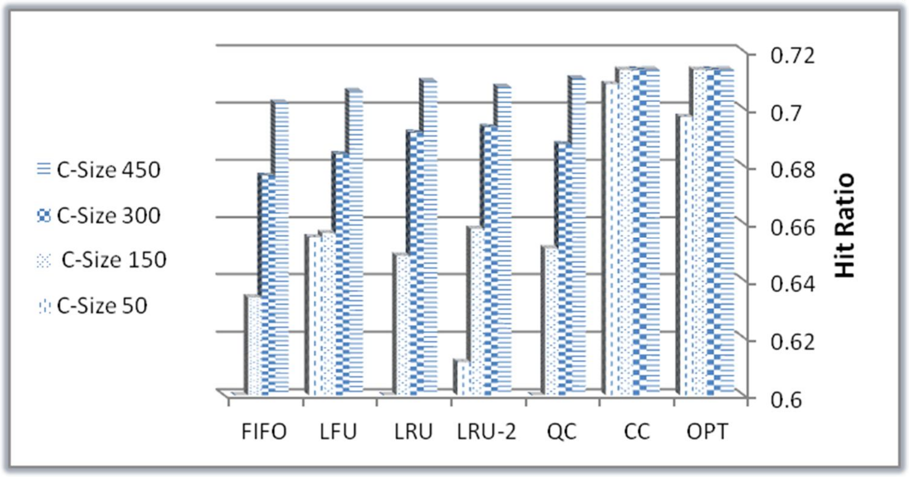

Figure 4-5: Hit Ratios of different replacement algorithms for 2000 video request

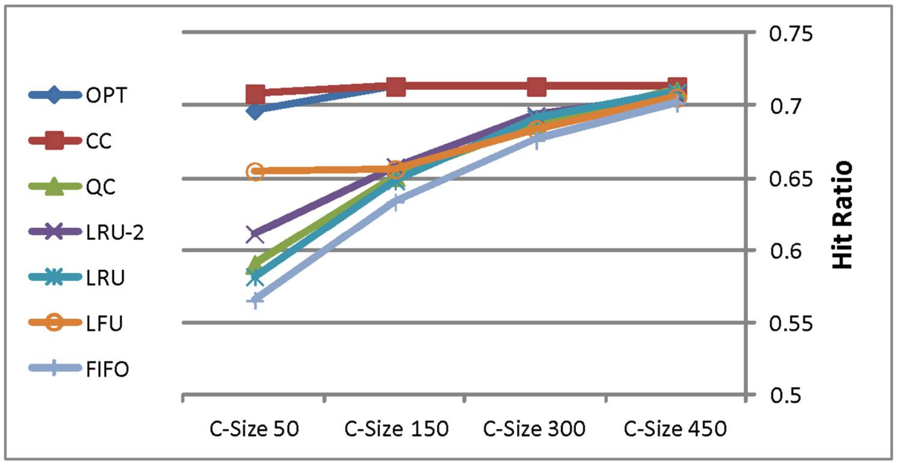

Figure 4-6: Hit ratio of different replacement algorithms for 2000 request

4.1.3 The hit ratio for 5000 video requests with different cache sizes

Here we consider 5000 video requests to evaluate the replacement algorithms with different cache sizes (100, 250, 300, 500 and 1000). The results are displayed in Table 4.6 and figure 4-7 and 4-8. We notice here that the QC algorithm has a lower hit ratio than the (LFU, LRU-2 and LRU) algorithms under small cache sizes, and when we increases the cache sizes the QC algorithm has a hit ratio greater than the LRU-2 algorithm and the LFU algorithm.

Table 4-6: Hit ratio for 5000 video request

ALGORITHEM	C-Size 100	C-Size 250	C-Size 300	C-Size 500	C-Size 1000
OPT	0.6792	0.717	0.7188	0.7188	0.7188
CC	0.7176	0.7188	0.7188	0.7188	0.7188
QC	0.5932	0.637	0.6472	0.6733	0.712
LRU-2	0.6092	0.6482	0.6558	0.6786	0.7106
LRU	0.5764	0.624	0.6388	0.671	0.718
LFU	0.6178	0.6514	0.66	0.6766	0.7064
FIFO	0.5602	0.6108	0.6206	0.656	0.7018

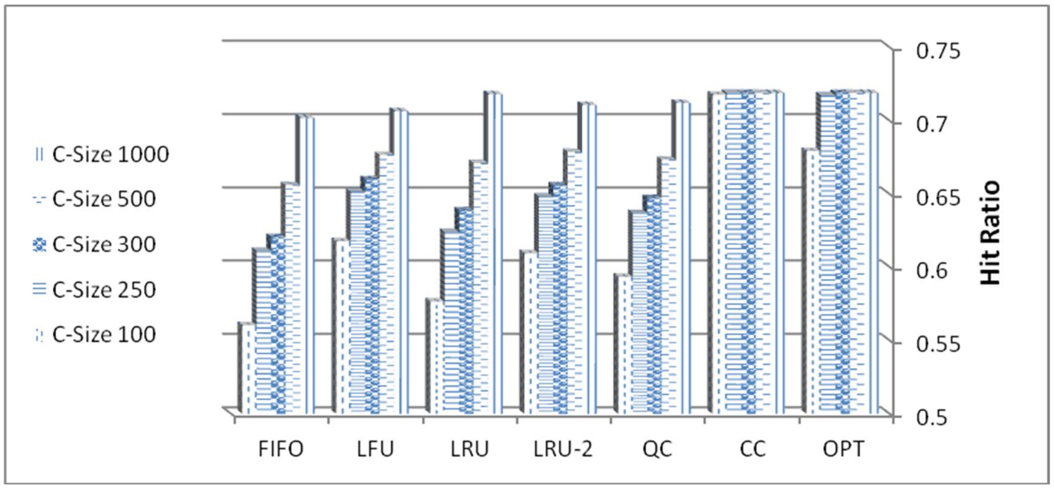

Figure 4-7: Hit ratio for 5000 video request

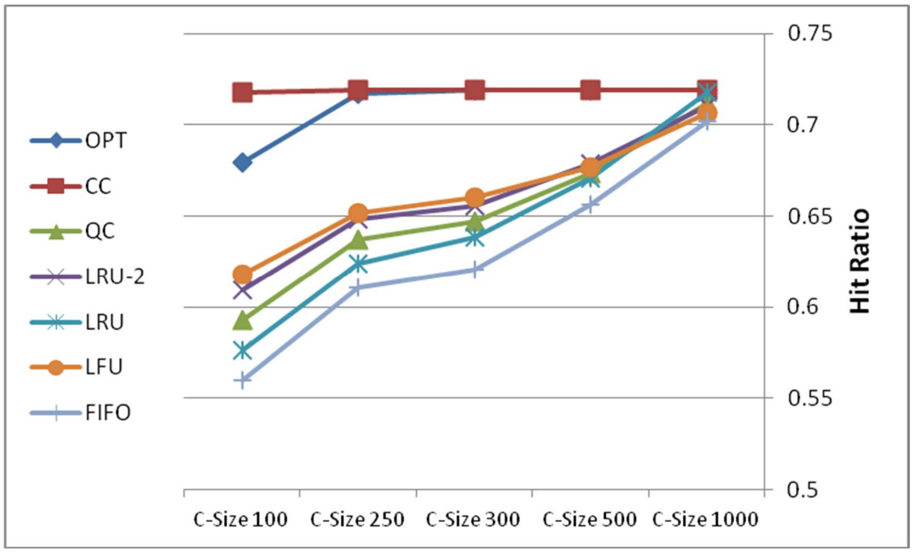

Figure 4-8: Hit ratio for 5000 request

4.1.4 The hit ratio for different video requests

In Table 4-7 we consider different combinations of number of videos and cache sizes (50 video requests with Cache Size=10, 200 video requests with Cache Size=30, 2000 video requests with Cache Size=150 and 5000 video requests with Cache Size=300). These inputs are applied to the OPT, CC and QC algorithms. We found that the QC algorithm always has a lower hit ratio than the OPT algorithm and the CC algorithm as presented in figure 4-9 below.

Table 4-7: Hit ratio for different video request on (OPT, CC and QC)

ALGORITHEM	50 Request	200 Request	2000 Request	5000 Request
OPT	0.6	0.73	0.7135	0.7188
CC	0.52	0.73	0.7135	0.7188
QC	0.56	0.705	0.651	0.6472

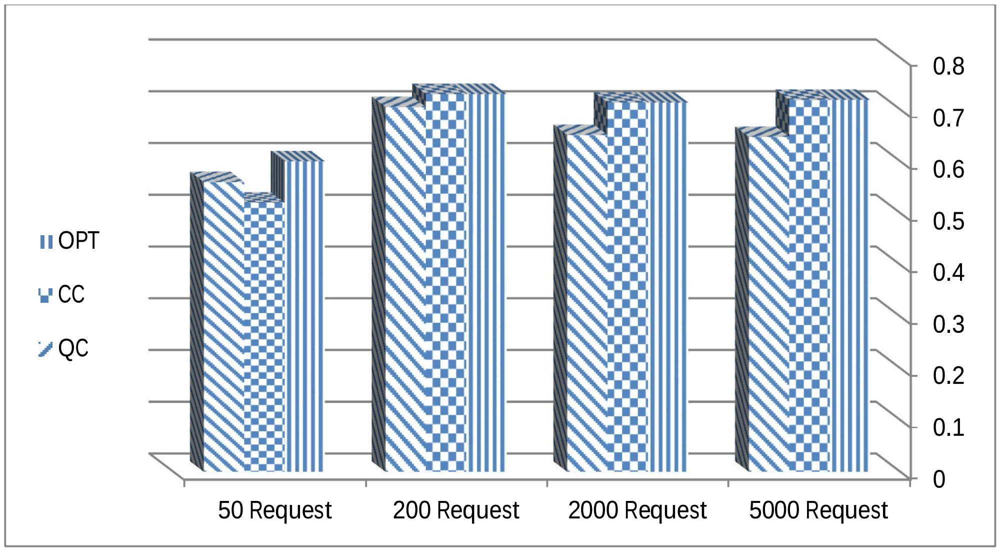

Figure 4-9: Hit ratio for different video request on (OPT, CC and QC)

In table 4-8 we consider different combinations of number of videos and cache sizes (50 video requests with Cache Size=10, 200 video requests with Cache Size=30, 2000 video requests with Cache Size=150 and 5000 video requests with Cache Size=300). These inputs are applied to the OPT, CQ, LRU-2 and LRU algorithms. We found that the OPT algorithm has the highest hit ratio under any request-cache size combination, as presented in figure 4-10 below.

Table 4-8: Hit ratio for different video requests using (OPT, QC, LRU-2 and LRU)

ALGORITHEM	50 Request	200 Request	2000 Request	5000 Request
OPT	0.6	0.73	0.7135	0.7188
QC	0.56	0.705	0.651	0.6472
LRU-2	0.6	0.705	0.658	0.6558
LRU	0.58	0.705	0.6485	0.6388

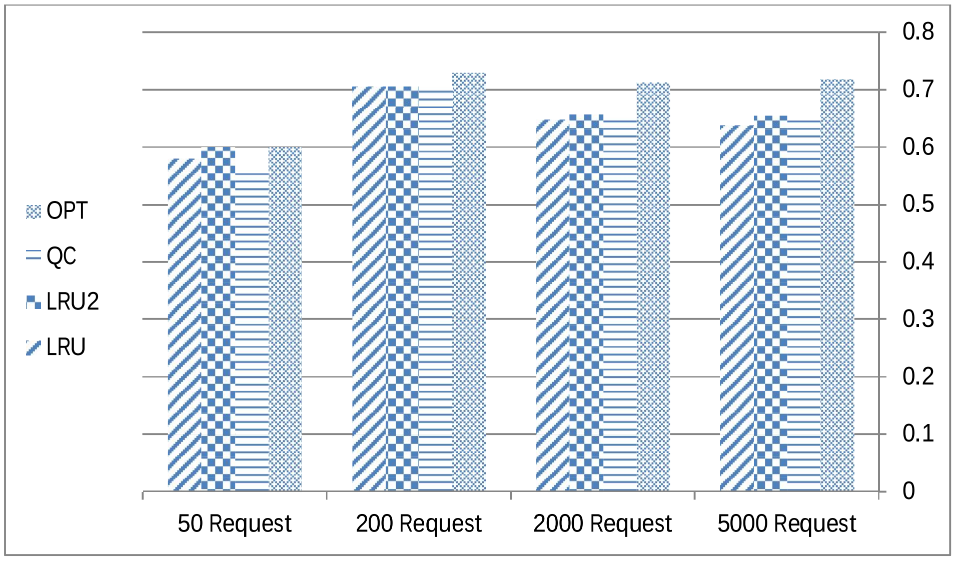

Figure 4-10: Hit ratio for different video requests using (OPT, QC, LRU-2 and LRU)

Below in figure 4-11 and table 4-9 we plot the hit ratio for different numbers of video requests (we have 200 request with cache size =30, 2000 request with cache size =150 and 5000 request with cache size=300) for all the evaluated algorithms.

Table 4-9: Hit ratio for different video request using all algorithms

ALGORITHEM	200 Video request	2000 Video request	5000 Video request
OPT	0.73	0.7135	0.7188
CC	0.73	0.7135	0.7188
QC	0.705	0.651	0.6472
LRU-2	0.705	0.658	0.6558
LRU	0.705	0.6485	0.6388
LFU	0.705	0.6565	0.66
FIFO	0.67	0.634	0.6206

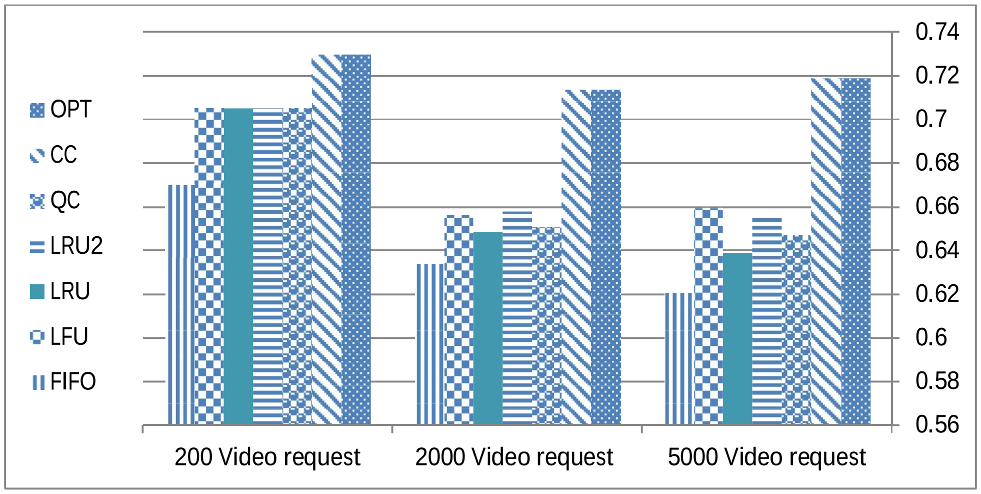

Figure 4-11: Hit ratio for different video request using all algorithms

Chapter Five

Conclusions and Future Directions

The limited storage capacity of caching is a concerning problem because of the large size of videos. Therefore the best replacement algorithm needs to be used to make accurate decisions for evicting a video currently in the cache to make room for a new video. In other terms it is important to employ the replacement algorithm with the right replacement policy that works best with the video nature. Thus we choose seven of the most popular cache replacement algorithms to find out the replacement algorithm that works best in video caching. This chapter summarizes the work that has been accomplished and presented in this thesis as well as suggesting possible directions for future research in the area.

5.1 Conclusions

The evaluation of the cache replacement algorithm can obtain different results depending on the nature of the input data (the number of requested videos and request frequency). In this work data (requested videos) are generated using the Zipf distribution. Based on the experiment results on the seven cache replacement algorithms, we come to the following conclusions:

- The CC algorithm always has the highest hit ratio, especially when applied on a small cache size; therefore the CC algorithm outperforms the other algorithms.
- The FIFO algorithm has the lowest hit ratio among all compared algorithms, under different cache sizes and different number of video requests.
- The QC algorithm with a small cache size has a lower hit ratio compared LFU, LRU and LRU-2, but the hit ratio becomes higher than these algorithms when the cache size increases.
- When ranking the algorithms based on their hit ratio achieved results, the ranking will be as follows: rank one is for both the CC algorithm and OPT algorithm because their hit ratio values are approximately the same, and the CC algorithm gives a better result only under small cache sizes. Rank two is the QC algorithm and rank three is for each of (LFU, LRU and LRU-2) algorithms because the difference between their hit ratios is marginal. The slight difference in cache hit ratio depends on the input video requests and how frequently they are requested. Rank four is for the algorithm with the smallest hit ratio, the FIFO algorithms. So the

order of the algorithms with respect to hit ratios ([CC, OPT], [QC, LFU, LRU-2, LRU], [FIFO]).

- For all algorithms increasing the cache size will increase the hit ratio, but increasing the cache size too much (more than a specific level) may not add any improvement.

5. 2 Future Directions

Our work in this thesis suggests several potential topics for further study:

- Analyses of the history of requested data to find out the suitable cache replacement algorithm

  A program that works as an analyzer can be installed in each personal computer (PC) or in a network computer. This program analyzes the history of requested data and decides the most suitable cache replacement algorithm that will work the best for this nature of requested data. The requested data analysis can be the frequency or the recency of requested date, or if the requested data is a part of a series and requests are made one chunk after the other. For each outcome there is a suitable cache replacement algorithm that achieves the largest hit ration.

- Evaluating replacement algorithms using variable video sizes

  This work is concerned with the case when requested videos are of the same size. A more complex and realistic model may consider the size of the requested videos to be variable such as in the Greedy Dual (GD) algorithm.

- Considering different popularity distributions

Other video popularity distributions can be used such as the Pareto and Bimodal distributions. These distributions are used to exemplify different video services including IPTV and Video-on-Demand.

References

[1] Kapil Arora and Dhawaleswar Rao, "Web Cache Page Replacement by Using LRU and. LFU Algorithms with Hit Ratio: A Case Unification," International Journal of Computer Science & Information Technologies, Vol. 5 (3), 2014, pp.3232 – 3235

[2] Abdullah Balamash and Marwan Krunz, "An Overview of Web Caching Replacement Algorithms," IEEE Communications Surveys and Tutorials, vol. 6, no. 2, 2004.

[3] Dong Zheng," Differentiated Web Caching – A Differentiated Memory Allocation Model on Proxies," PhD Thesis, Queen's University, (2004).

[4] "Java programming language" [Online]. Available: https://en.wikipedia.org/wiki/Java_%28programming_language 29 [Accessed: 07-10-2016].

[5] "About Microsoft Excel" [Online]. Available: [Accessed: 01-15-2016]. Philip Koopman, "Cache Organization", September 2.1998 [Online]. Available: https://www.ece.cmu.edu/~ece548/handouts/04cachor.pdf

[6] Philip Koopman, "Cache Organization", September 2.1998 [Online].Available: https://www.ece.cmu.edu/~ece548/handouts/04cachor.pdf

[7] Sarmed AL-Najim, "Web Caching: Architectures, Models and Importance to the Internet," 2002.

[8] Lei Shi1, 2, Zhimin Gu1, Lin Wei2, and Yun Shi3," An Applicative Study of Zipf's Law on Web Cache," International Journal of Information Technology, Vol. 12, No.4, 2006

[9] "Least Recently Used Caching Algorithms definition" [Online]. Available: https://en.wikipedia.org/wiki/Cache_algorithms#LRU.

[10] "The Least Recently Used (LRU) Page Replacement Algorithm". [Online]. Available: http://www.informit.com/articles/article.aspx?p=25260&seqNum=7 [Accessed: 07-10-2016].

[11] "LRU Algorithm Strategy" [Online]. Available: http://www.ustudy.in/node/7356 [Accessed: 15-09-2016].

[12] S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy and G Amjad Khan,"A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management," International Journal of Engineering Research and Applications (IJERA) Vol. 2, Issue 2, Mar-Apr 2012, pp.126-130

[13] Dohy Hong, Danny De Vleeschauwer and Fran¸cois Baccelli "A chunk-based caching algorithm for streaming video", NET-COOP 2010 - 4th Workshop on Network Control and Optimization, Nov

2010.

[14] Stefan Podlipnig and Uszlo' Boszonnbnyi, “Replacement strategies for quality based video caching", Multimedia and Expo, IEEE International Conference Vol. 2, 2002.

[15] Gille Damien, "Study of Different Cache Line Replacement Algorithms in Embedded Systems" MSc Thesis, ARM France SAS Les Cardoulines B2 - Route des Dolines Sophia Antipolis - 06560 Valbonne France, March 2007.

[16] Dhawaleswar Rao, "Study of the Web Caching Algorithms improvement of the Response Speed", Indian Journal of Computer Science and Engineering (IJCSE), Lovely Professional University, India, Vol. 3 No. 2, Apr-May 2012.

[17] Anvita Saxena, " A Study of Page Replacement Algorithms", Mewar University, Rajasthan, International Journal of Engineering Research and General Science Volume 2, Issue 4, June-July, 2014