A COMPARISON OF CACHE REPLACEMENT ALGORITHMS FOR VIDEO SERVICES

Areej M. Osman and Niemah I. Osman

College of Computer Science and Information Technology,
Sudan University of Science and Technology, Sudan

ABSTRACT

The increasing demand for video services has made video caching a necessity to decrease download times and reduce Internet traffic. In addition, it is very important to store the right content at the right time in caches to make effective use of caching. An informative decision has to be made as to which videos are to be evicted from the cache in case of cache saturation. Therefore, the best cache replacement algorithm is the algorithm which dynamically selects a suitable subset of videos for caching, and maximizes the cache hit ratio by attempting to cache the videos which are most likely to be referenced in the future. In this paper we study the most popular cache replacement algorithms (OPT, CC, QC, LRU-2, LRU, LFU and FIFO) which are currently used in video caching. We use simulations to evaluate and compare these algorithms using video popularities that follow a Zipf distribution. We consider different cache sizes and video request rates. Our results show that the CC algorithm achieves the largest hit ratio and performs well even under small cache sizes. On the other hand, the FIFO algorithm has the smallest hit ratio among all algorithms.

KEYWORDS

Cache update, Hit Ratio, Video Popularity, Zipf Distribution

1. 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 web documents. Caching provides an efficient solution to this problem by bringing documents closer to clients [1].

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 differs 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.

DOI:10.5121/ijcsit.2018.10208

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 requested in the future.

The work in this paper simulates a video service that employs caching where the contents of caches are updated using a number of cache replacement algorithms [2]. By applying the popularity distribution of content, the popularity of each video is determined and popular ones are stored in 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.

The contribution of this work is summarized as follows:

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.

The remainder of this paper is organized as follows:

Section 2 is an overview of caching, cache architectures and video popularity distributions. Section 3 reviews popular cache replacement algorithms and their implementation and algorithm steps. Section 4 explains the evaluation method. Section 5 demonstrates by results the implementation of different replacement algorithms and compares cache hit ratios for different scenarios. Section 6 is the conclusions.

2. CACHING AND VIDEO POPULARITY DISTRIBUTION

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. 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.

2.1 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 [3].

2.2 CACHING ARCHITECTURES

There are different ways that caches can be connected to each other in order to collaborate and benefit when employing more than one cache. The most famous architectures are hierarchical caching and distributed caching [4].

2.2.1 HIERARCHICAL CACHING

Hierarchical caching arranges caches in a tree-like structure where similar caches are placed on the same network level, and then connected to another level of caches. 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.

Although this architecture has several advantages such as reducing communication path and bandwidth usage, is hard to implement, as caches are required to configure neighbour caches and cache misses which causes extra delays. In addition, caches in higher levels must be very efficient and powerful to perform efficiently. Moreover, this architecture requires implementing load balancing algorithms to avoid the congestion of client requests to prevent delay.

2.2.2 DISTRIBUTED CACHING

In distributed caching all the caches in the network communicate with each other and work to serve each other's clients. When a user makes a request, the object will be looked up in the local cache. If the object is not there, then other caches are contacted. The cache with the requested object serves the user, lifting the burden off the server. The server is only contacted if no cache holds the desired object.

An advantage of distributed caching is that data transmission is easy and efficient, since there is less congestion around caches. However in large distributed cache systems, when the transmitted data is not from the neighbour cache but from caches over long distances, connection time can be quite slow. Therefore sometimes it might be faster to connect to the server directly.

2.3 THE ZIPF DISTRIBUTION

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 [5]. 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:

$$P \sim \rho-\beta \quad (1)$$

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 conform 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.

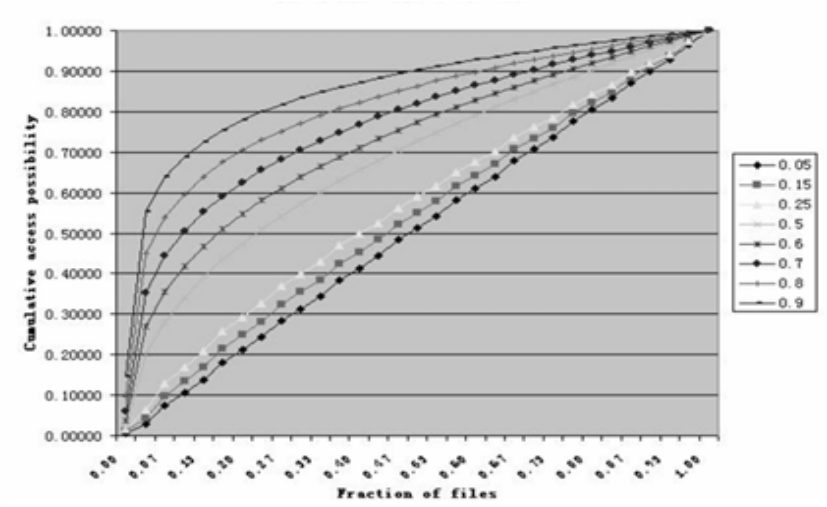

Figure 1: Zipf-like distribution [5]

Figure 1 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 approaches 1, popular objects receive greater fraction of requests [5].

3. CACHE REPLACEMENT ALGORITHMS

This section explains a number of cache replacement algorithms for different types of video services. It thoroughly explains the concept and steps of the algorithms that we evaluate in our work. Moreover, it also explains additional cache replacement algorithms that have been proposed in literature.

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 (OPT) and other two algorithm which are designed especially for videos (The Chunk-based Caching algorithm (CC) and Quality-based video Caching algorithm (QC). 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. Following is an overview of these algorithms.

3.1 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 steps of the FIFO replacement algorithm are:

```
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
```

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, we consider it in our work for comparison purposes.

3.2 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 change [7]. The LRU steps are:

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

```
        EndIf
Until end of all requests
Output the number of faults
```

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 [8].

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.

3.3 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 steps of the algorithm are:

```
Take inputs
Initialize cache and count array to -1
If (cache miss)
        Find the least frequently used video from the videos in the cache
        Replace video in cache by current video.
        Create array of counts and store it in 'count' array
EndIf
Increment counter
```

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 [6]. Also, new videos that just entered the cache are subject to being removed very soon because they start with a low counter, even though they might be used very frequently after that.

3.4 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 [9]. The steps of the Optimal algorithm are:

```
Take array n of videos
Initialize fault and cache array to -1
If (cache miss)
        If (cache is ful)
                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 into the cache
        EndIf
        Increase fault
Output number of faults
```

3.5 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 [6].

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. The pseudo-code of the LRU-K algorithm is:

```
LRU-K: on request for video p at time t
// scan cache queue to see if p is already in cache
q := the video at queue end
hit := false
While (q != null) do
        If (q equals p) then // hit
                hit := true
                 break
        EndIf
        q := next video before q
EndDo
If (hit) then
        // 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 video 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 video before q // vedio 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 video 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
```

3.5 THE CHUNK-BASED CACHING ALGORITHM (CC)

This algorithm is specifically for streaming video taking into account the dynamicity of the library [10]. The algorithm segments each video into chunks and assumes that 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.

The algorithm keeps 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.

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 NoT this chunk will have, knowing which videos are currently watched by the users and assuming that no user aborts watching a video .

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 .

The CC algorithm steps are as follows:

```
Repeat
    Input video
    If (current video is in cache)
        Update SK, NoH
    Else
    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 ++
    EndIf
Until end of all requests
Output Page_Fault
```

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 our simulation.

3.6 QUALITY-BASED VIDEO CACHING ALGORITHM

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.

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 [11].

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 [11]. where horizontal replacement is used to remove upper layers and vertical replacement is used to remove lower layers.

3.7 TREND-AWARE VIDEO CACHING THROUGH ONLINE LEARNING

A study proposed in [12] was accomplished to optimize the cache performance according to the trends of video content. The algorithm (Trend-Caching) forecasts the trend of video content and makes cache replacement decisions based on forecasted trend. The method explicitly learns the popularity trend of video content which is learned in an online fashion, and uses it to determine which video it should store in the cache and which it should evict .

The study showed that when compared to the optimal strategy, the performance loss of Trend-Caching is sub-linear in the number of processed requests, and that guarantees a fast speed of learning as well as the optimal cache efficiency in the long-term. The study also provided an extension to the original proposed algorithm, Collaborative Trend-Caching, that can exploit the trend similarity among multiple locations.

3.8 MIXED LFU AND FIFO FOR VIDEO POPULARITY DYNAMICS

The authors of [13] studied video popularity as a dynamic system and showed that the video popularity can change over time according to these facts (how quickly is the passing of glory days of a video's popularity, how low is the probability of replaying a video by the same user and the continuous arrival of new videos). These facts can also affect the optimal video caching strategy.

As the dynamics of video popularity depend on the age of the video, following a pattern for each type of video, the relative popularity of different videos at a given time is complex, and is governed by many factors. The authors proposed a mixed strategy (of LFU and FIFO) that can handle different kinds of videos .

The study results showed that the caching performance achieved by the mixed strategy is very close to the performance achieved by the offline strategy (offline strategy assuming tomorrow's video popularity is known in advance to be used as a performance benchmark).

3.9 VIDEO CACHING ALGORITHMS IN CONTENT DELIVERY NETWORKS

A Content Delivery Network (CDN) delivers a significant fraction of the entire Internet traffic, estimated at 71% of the entire Internet traffic by 2021 [14]. Effective caching at the edge is vital for the feasibility of these CDNs, which can otherwise incur significant monetary costs and resource overload in the Internet.

The study in [15] analyzed the requirement with the non-standard solutions and develop multiple algorithms for caching in these CDNs (LRU-based baseline solution to address the requirements, an intelligent ingress-efficient algorithm, an offline cache aware of future requests (greedy) to estimate the maximum caching efficiency that can be expected from any online algorithm, and an optimal offline cache for limited scales). The study used anonymized actual data from a large-scale, global CDN to evaluate the algorithms and draw conclusions on their suitability for different settings.

3.10 MIDDLEMAN: A VIDEO CACHING PROXY SERVER

The work is [16] described MiddleMan, which is a collection of cooperating proxy servers connected by a local area network (LAN). MiddleMan is a proxy research that concentrates on video only. It reduces the start-up delays and the possibility of adverse Internet conditions disrupting video playback. It can also reduce the server load by intercepting a large fraction of server accesses.

The work implemented an LRUk cache replacement algorithm, which resulted in the highest hit rates. It also implemented the HistLRUpick, a variation of the LRUk algorithm, which achieved good hit rates as well as effective load balancing. The results also revealed that a relatively small global cache size which is about 13.7% of total video sizes, resulted in very high byte hit rates. The study showed that a larger number of proxies were preferable to a smaller number of proxies for the same global cache size.

MiddleMan reduces load in servers by intercepting a large number of server accesses. Hence, the major effect of MiddleMan is to greatly increase the effective bandwidth of the entire video delivery system, allowing more clients to be serviced at any given time.

3.11 Time-Based Green Cache Replacements for Future HDTV

The cache update algorithm proposed in [17] measure the popularity of TV programmes and makes use of the TV guide to predict the times of day that TV viewers watch popular programmes during the 24 hour period. Based on this, the work proposes updating cache contents several times a day with popular programmes that will be requested at each time of day. The work investigates the power consumption of the IPTV core network with respect to the number of times cache contents are updated. It also considers Standard Definition (SDTV) as well as High Definition TV (HDTV).

The results showed that time-based content replacements increase cache hit ratios as the number of daily cache updates increases. The proposed strategy also reduces the network power consumption by up to 86% compared to no caching. Although the improvement in cache hit ratios is the same for SDTV and HDTV, more power savings are achieved for HDTV, as larger sizes of video files consume more power to be downloaded.

4. Simulation of Cache Replacement Algorithms

The simulation in our work evaluates and compares replacement algorithms. Following is an explanation of the evaluation model, input data and simulation flowchart.

4.1 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 2.

Figure 2: Simple view of the model

4.2 Input Data

The input data is a series of requests that are generated randomly using the Zipf distribution. We apply Equation (2) defined in [4][6] to find the popularity P_i of the i^{th} video, where the harmonic value (skewness) of the Zipf distribution α =0.75.

$$P_N(i) = \frac{\Omega}{i^{\propto}} \qquad (2)$$

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

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

4.3 REPLACEMENT ALGORITHM FLOWCHART

Figure 3 shows the simplified flowchart 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 replaced 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. In addition, 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.

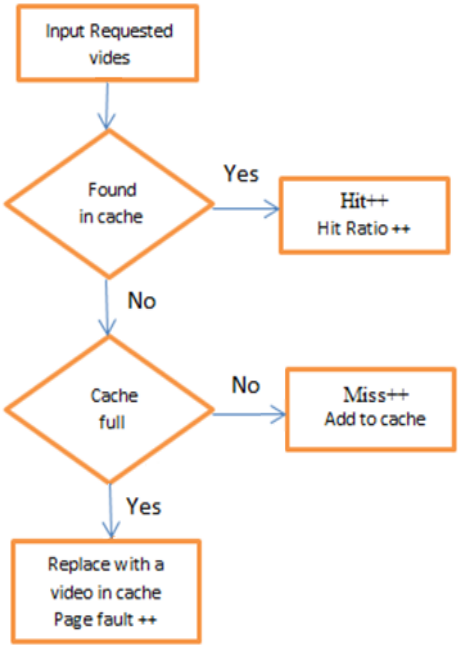

Figure 3: Flowchart for replacement algorithms

5. IMPLEMENTATION AND EVALUATION OF CACHE REPLACEMENT ALGORITHMS

A separate simulation code for each of the seven replacement algorithms is implemented and run. 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. The output is the hit ratio for each algorithm under different scenarios. For the LRU-K algorithm, we select the value k=2.

5.1 THE HIT RATIO FOR 200 VIDEO REQUESTS WITH DIFFERENT CACHE SIZES

We compare the hit ratios attained by different cache replacement algorithms having 200 video requests under different cache sizes. The hit ratio values are shown in Table 1 and Figure 4 shows a comparison between the hit ratios. In general, as we increase the size of the cache, the hit ratio increases with the CC algorithm achieving the highest hit ratio with a cache size of 10. As the cache size reaches 50, all algorithms saturate at a hit ratio of over 0.7.
Observing Figure 4 and Table 1 we find 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.

Table 1: Values of hit ratio for 200 requests under different cache sizes

Algorithm	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

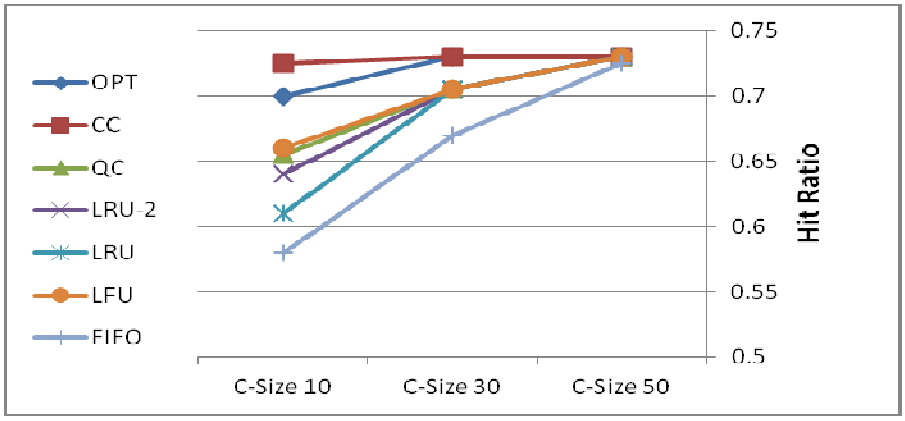

Figure 4: Comparison of algorithm hit ratios for 200 requests under different cache sizes

5.2 THE HIT RATIOS 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 2 and Figure 4 and Figure 5. 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 2: Hit ratio values for 5000 video request under different cache sizes

Algorithm	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

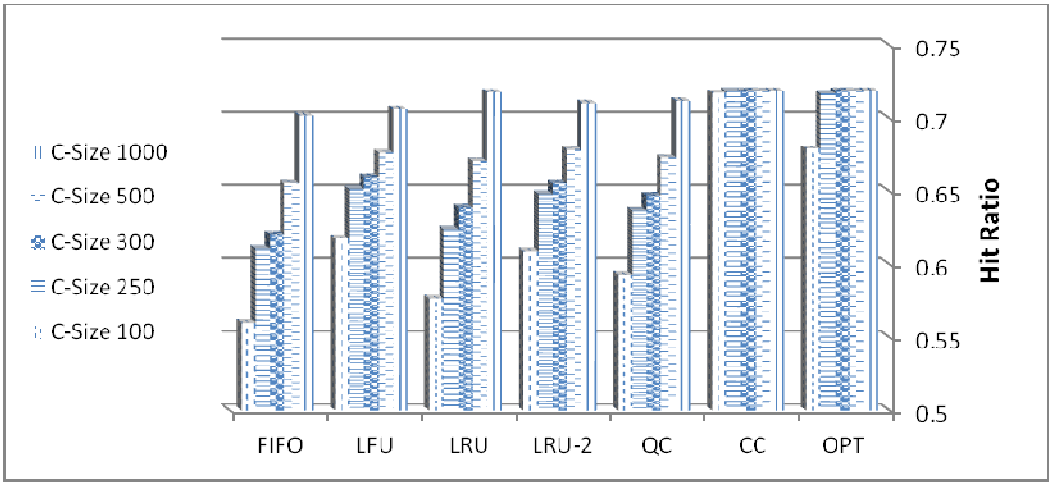

Figure 4: Hit ratios of algorithms having 5000 video requests under different cache sizes

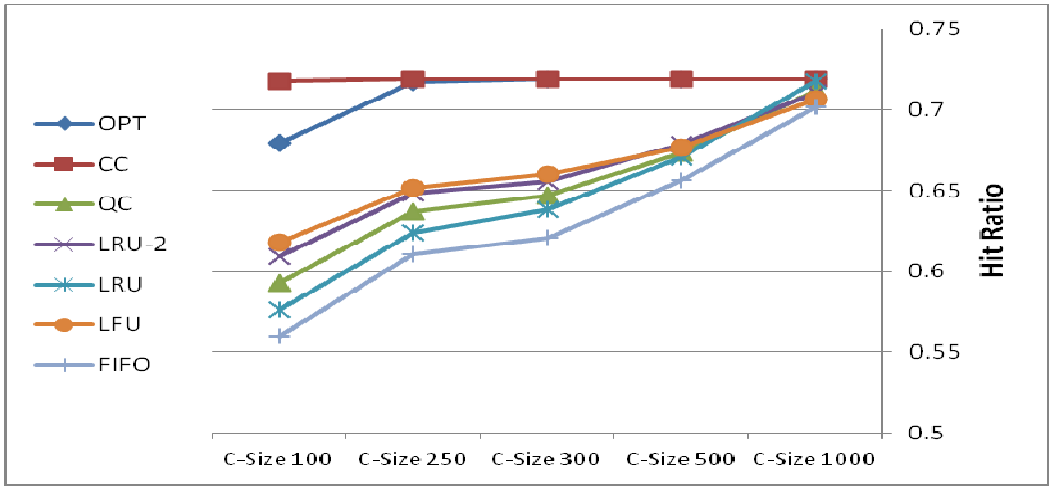

Figure 5: Comparison of hit ratios of algorithms having 5000 video requests under different cache sizes

5.3 THE HIT RATIO FOR DIFFERENT VIDEO REQUESTS

We evaluate cache replacement algorithms considering different combinations of number of video requests 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). The resulting hit ratios are shown in Table 3 and Figure 6. We found that the QC algorithm always has a lower hit ratio than the OPT algorithm and the CC algorithm.

Table 3: Values of hit ratios for different video requests and cache sizes

ALGORITHEM	50 requests/ cache 10	200 requests/ cache 30	2000 requests/ cache 150	5000 requests/ cache 300
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
LRU-2	0.6	0.705	0.658	0.6558
LRU	0.58	0.705	0.6485	0.6388
LFU	0.63	0.705	0.6565	0.66
FIFO	0.55	0.67	0.634	0.6206

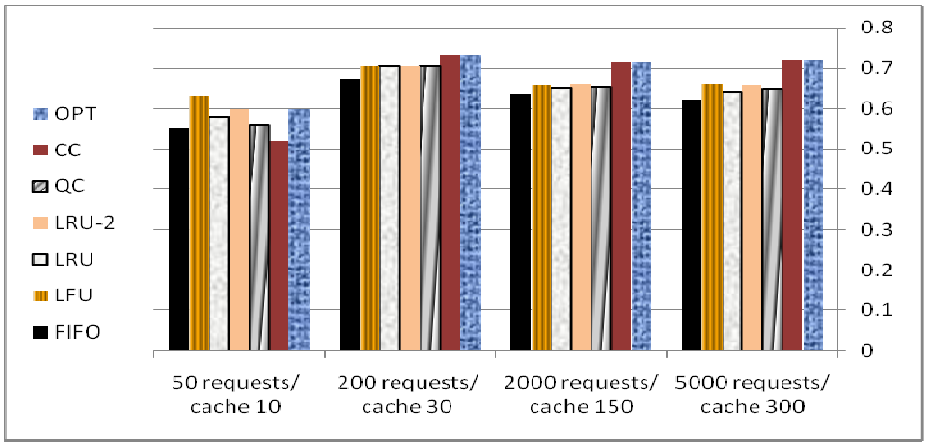

Figure 7: Hit ratios for different video requests and cache sizes

6. CONCLUSIONS

This paper has evaluated and compared a number of cache replacement algorithms in terms of resulting hit ratio under different number of video requests and cache sizes. Based on the experiment results for the seven considered cache replacement algorithms, we can rank these algorithms with respect to their achieved hit ratio. Rank one is for both the CC algorithm and OPT algorithm as their hit ratio values are the highest. 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. Rank four is for the algorithm with the smallest hit ratio, the FIFO algorithms. For all algorithms, increasing the cache size will increase the hit ratio, but increasing the cache size too much (more than a specific size) does not add any significant improvement in hit ratio.
For future work we suggest analyzing the history of requested data to find out the suitable cache replacement algorithm considering request history. Also evaluating replacement algorithms using variable video sizes produces a more complex and realistic model. Other video popularity

distributions such as the Pareto and Bimodal distributions may also be used in the evaluation to exemplify different video services.

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

[4] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching", *IEEE/ACM Transactions on Networking*, Vol. 9, no. 4, August 2001.

[5] Lei Shi, Zhimin Gu, Lin Wei, and Yun Shi," An Applicative Study of Zipf's Law on Web Cache," *International Journal of Information Technology*, Vol. 12, No.4, 2006.

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

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

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

[9] 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.

[10] 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.

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

[12] Suoheng Li, JieXu, Mihaela van der Schaar and Weiping Li ,"Trend-Aware Video Caching through Online Learning,” *IEEE Transactions on Multimedia*, vol. 18, pp. 2503–2516 , July 2016.

[13] Yipeng Zhou, Member, IEEE, Liang Chen, Chunfeng Yang, and Dah Ming Chiu, Fellow, IEEE "Video Popularity Dynamics and Its Implication for Replication,” *IEEE Transactions on Multimedia*, vol. 17, No.8, pp. 2503–2516, August 2015.

[14] Cisco Visual Networking Index: Forecast and Methodology, 2016–2021 [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, [Accessed: 30/4/2018].

[15] Kianoosh Mokhtarian, Hans-Arno Jacobsen, "Caching in Video CDNs: Building Strong Lines of Defense," *Proceedings of the Ninth European Conference on Computer Systems*, April, 2014.

[16] Soam Acharya, and Brian Smith," MiddleMan: A Video Caching Proxy Server”, Proceedings of NOSSDAV, 2000.

[17] Niemah I. Osman, Taisir El-Gorashi, Louise Krug, and Jaafar M. H. Elmirghani, "Energy-Efficient Future High-Definition TV," *Journal of Lightwave Technol*ogy, vol. 32, pp. 2364-2381, 2014.

AUTHORS

Areej Mohamed Osman received the B.Sc. degree (Honours) in Computer Science in 2013 and the M.Sc. degree in Computer Science in 2016 from Sudan University of Science and Technology, Khartoum, Sudan. She worked as a Teaching Assistant in Sudan University (2013-2015). Her current research interests include caching in IPTV services and Video-on-Demand.

Niemah Izzeldin Osman received the B.Sc. degree (first class honours) in Computer Science from Sudan University of Science and Technology, Khartoum, Sudan, in 2002 and the M.Sc. degree (with distinction) in Mobile Computing from the University of Bradford, U.K., in 2006 and the Ph.D. degree in Communication Networks from the University of Leeds, U.K in 2015. She is currently an Assistant Professor at the Department of Computer Systems and Networks, Sudan University of Science and Technology, Sudan. Her current research interests include performance evaluation of 4G LTE networks, Internet of Things and QoE of video services.