ENHANCEMENT OF YOUTUBE CDN WITH CACHING HIERARCHY AND NEW REPLACEMENT POLICY

Varangpa Suranuntakul, Chutimet Srinilta
King Mongkut’s Intitue of Technology Ladkrabang
Bangkok, Thailand
varangpa@gmail.com, chutimet@yahoo.com

ABSTRACT
YouTube has become a highly successful video sharing website and has had an enormous impact on the Internet traffic. Content Delivery Network (CDN) is one possible solution which can provide users on a global scale faster access while reducing traffic on the backbone. Its performance gain nevertheless depends in large part on efficient caching management.

The caching techniques using caching hierarchy in surrogate servers of YouTube CDN have been proposed in this paper. Moreover, a new replacement policy (equation) for use in the replacement system and our caching hierarchy are discussed. Through a detailed simulation environment, the proposed techniques can significantly improve caching performance relative to the other polices.

KEY WORDS
Caching System, CDN, Replacement Policy, YouTube

1. Introduction

YouTube is a popular short video sharing site that enables users to participate and contribute to the site. YouTube users can upload their clips, which are termed user-generated contents (UGC), and discuss the contents using interactive features. In addition, UGCs are accessible by anyone at any time.

Alexa [1] ranks YouTube the third most visited website on the Internet, behind Google and Facebook. According to data published by ComScore, YouTube is the dominant provider of online video in the United States, with a market share of around 43 percent and more than 14 billion videos viewed in May 2010 [2]. YouTube reported in January 2012 that the site was receiving more than three billion views per day [3].

The popularity of YouTube site has a significant impact on the Internet traffic, which results in scalability limitations for YouTube. Therefore, efficient content distribution and caching techniques are essential to YouTube’s network traffic management.

In this work the authors have proposed for YouTube CDNs the caching system techniques consisting of two major components: (1) Cache hierarchy schemas whereby total cache space is divided into hierarchy caches; and (2) new cache utility value (CUV) equation for replacement system and updating the locations of videos in the cache hierarchy.

There exist limited researches study related to this work and thereby give rise to this research.

Cha et al [4] have studied the file referencing behavior of user generated content. The researchers sampled Daumn UGC and YouTube repository using a web crawler to study file referencing patterns. In addition, the work presents the concept of caching schemas.

Pallis et al [5] propose logistic sigmoid function to classify the surrogate server cache into two parts: static cache and dynamic cache. The proposed approach, i.e. R-P, classifies the replicas with respect to their quality values. In particular, the quality value of each replica is expressed by the user interest or the lack of user’s interest in the underlying replica.

Abhari et al[6] introduce a workload characterization for generating user access session by analyzing statistical behavior of YouTube video attributes. This workload generator technique has been adapted for evaluation of our proposed research.

The remaining of the paper is organized as follows: Section II addresses the YouTube CDN architecture while caching system for CDN is discussed in Section III. Section IV presents the proposed caching techniques for YouTube CDNs. The experiments are detailed in Section V with the last section, i.e. section VI, concluding the paper.

2. YouTube CDN

CDN is a large system of distributed servers (surrogate server) deployed in multiple data centers on the Internet. The goal of a CDN is to serve content to end-users with high availability and performance.

CDN consists of a set of surrogate servers (in multiple locations worldwide), routers and network elements. Surrogate servers are the key elements that act as replcators since they store copies of original server contents. Once a client requests for content from an original server, the client’s request is directed to an appropriate CDN’s surrogate server.
2.1 YouTube CDN Architecture

Adhikari et al [7,8] use a reverse engineering-based methodology to uncover the design of the YouTube video delivery system. They have reported that YouTube’s delivery cloud consists of three components: the video-id space, hierarchical logical video server DNS namespaces, and physical video cache servers. YouTube uses a combination of static and dynamic load balancing approaches to distributing demands or requests to its physical resources.

Video-id space or video identifier is mapped to a unique hostname in hierarchical logical video server namespaces using static hash based mechanism. Then, YouTube uses semi-dynamic approach by mapping this hostname to IP address which represents a physical video based upon the user location and current demand. To further balance loads of different physical servers, YouTube cache uses HTTP redirection (dynamic load balancing) to direct user from busy servers to less busy video servers.

YouTube defines three multiple DNS namespaces, each representing a collection of logical video servers with certain roles for better availability and reliability: Primary (38 locations), Secondary (8 locations) and Tertiary namespaces (5 locations).

2.2 YouTube Video Delivery Mechanisms

![Figure 1. YouTube Video Delivery Mechanisms](image)

Following are the steps taken by YouTube to serve video contents to users worldwide:

1. A user accesses the YouTube website with a video id.
2. The YouTube web server will return a HTML page with embedded URL of logical video server of this video to the user.
3. When the user clicks the playback button, the web browser queries IP address of this logical video server from DNS server.
4. DNS server maps appropriate surrogate server IP address using locality-aware server selection.
5, 6. The user requests and downloads video contents from the surrogate server. Occasionally, the server cannot provide the contents; the client will thus be “redirected” to a different server using the dynamic load balancing approach.

3. Caching Technique in CDN

3.1 Proxy Servers and CDN

CDN and proxy servers have been proposed as different technologies; however, their common goal is to bring content closer to the users, thereby reducing the response time. Both technologies possess their respective advantages and disadvantages.

The focus of CDNs is to serve great quantities of requests and content volumes. The challenges however lie in the issues of replication and distribution cost. Since replica placements should be static for a certain time period, this leads to un-optimized storage capacity usage due to redundant, possibly outdated or unwanted, content contained in the surrogate servers.

On the other hand, proxy servers adapt content caching according to varying access patterns using cache replacement algorithm. Nonetheless, proxy servers do not scale well for serving large volumes of data or user populations.

The integration of the advantages of CDN and proxy can increase efficiency in information distribution to users over the Internet. Web caching is mainly implemented by proxy servers while content replication is the main practice on CDN.

3.2 Emerging Caching Techniques in CDN

In the content replication, surrogate servers keep replicas of objects on behalf of content provider. The replicated contents in CDN remain static for a definite amount of time. As a result, if user access patterns change, the replicas in the surrogate servers would not satisfy the user’s request. A solution to such an issue would be to integrate both caching and replication policies into the storage space of surrogate servers.

In the web caching, the cached objects are determined by cache replacement policies. The cache replacement policies decide which objects will evict from the cache so as to accommodate new objects.

As shown in Figure 2, the cache of surrogate server could be segmented into two partitions:

1. Static cache partition. The content of the static cache is identified with the application of a content replication algorithm.
2. Dynamic cache partition. Cache replacement policies are used in caching data in the dynamic cache. The dynamic cache is initially empty before being filled.
with content at run-time according to the selected cache replacement policy.

The challenges of integrating content replication algorithm (static) with web caching (dynamic cache) are in partitioning the surrogate server into sizes appropriate for caching and replication; and in assigning to an object a static or dynamic status. There exist an extensive body of research studies on the problems, examples of which are NP complete [6], Hybrid [6] SRC [7] and R-P [8].

Figure 2. Caching in CDN surrogate server

4. The Proposed Caching System

This work proposes the caching techniques for surrogate servers on YouTube CDN, which consists of two components:

(1) Caching Hierarchy: Total cache size is divided into three levels (i.e., Static, Hybrid and Dynamic) according to the cache value of video which is determined with the replacement policy or specific equation.

Videos with high value are cached in Static (high level cache), medium value videos in Hybrid, and low value videos in Dynamic. Only in Dynamic level does the replacement process take place while in the other two levels the objects remain static for a definite length of time.

This technique will enhance the efficiency of caching operation and avoid untimely video eviction.

Figure 3. The proposed caching hierarchy

(2) Equation for replacement policy and updating video location in cache hierarchy: In addition to caching hierarchy, this research presents an equation for calculation of video cache value for use in replacement process and updating video location.

The equation is the integration of frequency, time and video size factors. Further details of the equation are in section 4.3.

The proposed caching techniques consist of four sections as follows:

4.1 Workload Generator

Simulation data should be representative of real events and issues. In this research, we work with YouTube video data; therefore, the simulation data are (partial) real YouTube user requests. There exist research studies on streaming workload [9,11,12]; however, only [9] has focused on YouTube users’ requests.

Abhari [9] introduces the design of workload generation from YouTube video characteristics, which is divided into two main elements of server workload generator and client session generator. The server part generates a collection of files corresponding to the distributional model in order to create server database (i.e., videos’ contents and relationships), while the client session generator simulates a user’s accessing of video files.

In practice, requests to the server are of multiple requests from multiple clients simultaneously, rendering the data from the above client session generator inadequate and thereby requiring additional work. As such, new parallel requests are generated with the session client hash map which consists of keys (i.e., number of request sessions or number of users and values (i.e., collections of linked lists that represent sequences of user’s accessing video files); and the request queuing list, each node of which contains video id and requested time of this video. The steps of the new parallel requests are as follows:

(1) Assign current request* the first video of the first session (i.e., the first element of the first linked list in the session client hash map) with zero (0) timestamp.

(2) Update the timestamp (i.e., referencing time) with the time interval value from Poisson distribution.

(3) Check request time of videos in the request queuing list.

- If there exists a video with the requested time lower than timestamp, such a video will become current request*. Looping continues until none of the videos with requested time lower than timestamp remain.

- If no video with the requested time lower than timestamp exists, current request* is the first video of next session (i.e., first video of next linked list in the session client hash map).

(4) Repeat Step 2 until the session ends and request queuing list is empty.
Once current request* is obtained, follow the steps below:

1. Determine whether the current request should be viewed entirely or partially. If partially viewed, view time is sampled from range of [1, duration of video].
2. Next video of that session (i.e., next node in session client hash map) will be requested at the point of time that is calculated from the summation of view time in Step 1 with time interval value from Poisson distribution. Add this next video and requested time to request queuing list.

4.2 Caching Partition and Replacement System

This section describes division of cache space, data structures which represent each cache level, and replacement mechanism for handling user requests.

As previously mentioned, cache hierarchy is one possible method to enhance the efficiency of caching system. This work divides total cache size into three levels based on a specific equation or on the chosen replacement policy.

1. Dynamic Cache. It is the cache of lowest level and is the only cache that uses replacement process.
2. Hybrid Cache or Second Cache Level. This cache is between dynamic and static. All videos in this level come from videos in other levels, i.e. upgrading from dynamic or downgrading from static.
3. Static Cache or Highest Cache Level. The feature of this cache level is similar to that of hybrid cache level, but only static cache retains videos with highest probability of being requested in the future.

The two main issues on dividing cache are appropriate space size for each cache level and the cache update time, both of which are addressed in the section on experiment.

For simulation purpose, all cache levels are represented with three separate linked lists. The number of elements in each linked list depends on the cache size of each level. The replacement process takes place in dynamic cache, whereas the others (i.e., Hybrid and Static) remain static until the cache update time.

When a video is requested but not found in the three linked lists and if the remaining dynamic cache size is sufficient for this video, the video is added instantly at the rear of dynamic list. If not, the replacement policy or equation evicts lower cache value videos out of the dynamic list until there is sufficient space for the new video and then this new video is added at the rear of dynamic list.

In case that the requested video is found in the one of three linked lists, the video position remains unchanged, and the replacement mechanism calculates new parameters based on the replacement policy or equation.

Follow the aforementioned steps for each requested video until the end of time period or the cache update time.

4.3 Parameters and Cache Utility Value

This research presents the new equation for use in replacement mechanism and updating video location at cache update time period. Frequency, time, and file size factors are collectively taken into account to decide whether a video should be evicted or cached in cache.

The equation can be expressed by the following formulation:

\[
CUV = nref + 1 - \left( \frac{Trefk - Trefk-1}{Trefk - Tref0} \right) + \frac{1}{\text{Size (kB)}}
\]

(1)

Table 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUV</td>
<td>Cache utility value</td>
</tr>
<tr>
<td>nref</td>
<td>Number of times a video is requested as being stored in cache.</td>
</tr>
<tr>
<td>Tref0</td>
<td>The time that a video is cached.</td>
</tr>
<tr>
<td>Trefk</td>
<td>The current time that a video is requested.</td>
</tr>
<tr>
<td>Trefk-1</td>
<td>The previous time that a video is requested.</td>
</tr>
<tr>
<td>Size</td>
<td>Video file size (in kilobytes)</td>
</tr>
</tbody>
</table>

The proposed equation consists of three expressions that represent three different factors.

First, \(nref\) shows the video popularity as being stored in cache. This value will be reset when a video is evicted and increase when the video is requested.

Next is \(1 - \left( \frac{Trefk - Trefk-1}{Trefk - Tref0} \right)\) expression, which is concerned with time factor, where \(Trefk - Trefk - 1\) is the time interval of video’s current request and previous request and \(Trefk - Tref0\) is time period since video is stored in cache. As such, the result of a long cached video with low time difference of request is much greater than that of a recently cached video with high time difference of request.

In our simulation data, we assign the range of this expression as:

\[0.001 \leq 1 - \left( \frac{Trefk - Trefk-1}{Trefk - Tref0} \right) \leq 0.1\]

(2)

The \(\frac{1}{\text{Size (kB)}}\) expression is file size factor. It will return high value if video size is small and low value if video size is large. In the other words, this equation uses greedy algorithm in this last expression.

Similar to the previous expression, the range of expression outcome for this simulation is assigned:

\[0.00001 \leq \frac{1}{\text{Size (kB)}} \leq 0.001\]

(3)

Both ranges depend on simulation or evaluation data and thereby are used only in this research study.
This section describes changing of video location when reaching the cache update times, the mechanism of which is based on the selected replacement policy or equation. The cache update times in this research are of two types: static cache update time and hybrid cache update time.

As discussed earlier, static cache stores videos with highest possibility of request and all videos in this level upgrade from the others (i.e., hybrid and dynamic); thus, the time interval of static cache update time should be greater than that of hybrid cache update time. The appropriate time intervals are detailed in the Experiment section.

Before update of video location, three linked lists are sorted by either the replacement policy or equation. In this paper all lists are sorted by CUV values.

If it is the hybrid cache update time, video commutation occurs between hybrid and dynamic caches by moving as many higher cache value videos as possible (i.e., CUV values for this research) to hybrid cache. In case of static cache update time, switching occurs in all three levels by moving the highest cache value videos to static cache.

### Figure 4. Video Commutation

5. **Experiment**

The performance of the proposed caching techniques is evaluated by comparing the proposed techniques with the conventional replacement policy: LRU, LFU; and with the recently CDN caching research called R-P [5]

In order to assess our caching hierarchy and efficiency of the new equation, we simulate the caching systems both with and without caching hierarchy using LRU, LFU, R-P and the CUV equation. Updating video location in caching hierarchy employs the same method as the replacement policy. In addition, two total cache sizes of 10 GBytes and 50 GBytes are used to investigate the impact of cache size factor.

The important parameters to consider are Cache Size Ratio (Static-Hybrid-Dynamic), Hybrid Cache Update Time, and Static Cache Update Time. The proper parameter values and thereby the best experiment result can be achieved by adjusting the values of the parameters until the former are obtained. In this research, the Cache Size Ratio is 1:2:4, Hybrid Cache Update Time is 1 day, and Static Cache Update is 3 days.

#### Performance matrices:

1. **Hit ratio** is the fraction of cache hits to total number of requests. A high hit ratio indicates an effective cache replacement policy, increased user serviceability, and reduced average latency.

   \[
   \text{Hit ratio} = \frac{\text{Hit rate}}{\text{Total number of requests}} \times 100
   \]

2. **Byte hit ratio** is the fraction of the total number of bytes that were requested and existed in cache to the number of bytes that were requested. A high byte hit ratio improves the network performance.

   \[
   \text{Byte hit ratio} = \frac{\text{Bytes that found in cache}}{\text{Bytes that were requested}} \times 100
   \]

#### Parameters:

- Number of videos: 50,000 videos
- Number of requests per day: About 500,000 requests
- Experiment time: 30 days
- Cache size: 10 GBytes and 50 GBytes
- Cache size ratio (Static-Hybrid-Dynamic): 1-2-4
- Hybrid Cache Update Time: 1 Day
- Static Cache Update Time: 3 Days

#### Experiment Steps:

The experiment follows the following steps:

1. Create videos, video contents, and video requests for a period of 30 days using workload generator process (details in section 4.1).
2. Create data structures (three linked list) which represent each cache level in the cache hierarchy and allocate cache space for each cache level (see details in section 4.2).
3. Load all video requests of \(x^{th}\) day (i.e., Day\(x\)) and implement replacement mechanism for each request of that particular day (see replacement system details in section 4.2).
4. At end of requests of the day, sort all linked lists and update video location if the cache update time is reached (see description of updating video location in section 4.4).
5. Evaluate the caching performance of that particular day.
6. Tally up Day\(x\) and repeat Step 3 until Day\(x\) is 30 (day 30)

#### Results:

The results of Hit ratio for 10 GBytes and 50 GBytes cache sizes are respectively shown in Figures 5 and 7, while Byte hit ratios of 10 GBytes and 50 GBytes cache sizes are shown in Figures 6 and 8, respectively.

All the results can be summarized with the following expression:

\[
\text{CUV} > \text{LFU} > R - P > \text{LRU}
\]
The above expression manifests that the proposed policy is the best policy (equation) for replacement process and updating video location.

In LRU policy, caching hierarchy barely has an effect on Hit ratio and Byte hit ratio when updating video location. In addition, the replacement process indicates that this policy is the worst policy because it takes into account only the recent request time without tending to popularity of video or other factors.

Although using the complex equation that entails logistic sigmoid function and user interest, R-P policy considers merely time factor, thus leading to the experiment result similar to that of LRU policy.

The outcomes of LFU policy indicate positive effects of the frequency factor on the Hit ratio and Byte hit ratio, especially with the use of caching system with cache hierarchy. Even though in the initial stage results of caching system with cache hierarchy are worse than those without cache hierarchy, the results of the system with cache hierarchy gradually, after cache updating during the update time period, arise and surpass those without cache hierarchy.

The proposed CUV equation integrates three factors, i.e. time, size and frequency, each of which is assigned weight according to range limits. The following charts ascertain the capability of the proposed equation to improve Hit ratio and Byte hit ratio, particularly in the caching system with cache hierarchy. The results of initial stage are identical to those from the other policies. Nevertheless, after cache updating process Hit ratio and Byte hit ratio significantly increase and tend to surpass those of the caching system without cache hierarchy.

Taking into consideration the cache size factor, caching hierarchy conspicuously improves Hit ratio and Byte hit ratio in both cache sizes, i.e. 10 and 50 GBytes. As time passes, the difference between Hit ratios and Byte hit ratios of with and without caching hierarchy of low cache size tends to greater than that of high cache size.

6. Conclusion

This paper has proposed the method to improve the caching system of YouTube CDN caches. We have introduced the caching hierarchy and new equation, the latter of which integrates frequency, time and file size factors for replacement policy and updating video location process in caching hierarchy. The proposed system is evaluated by simulation and compared with the widely used replacement policies of LRU and LFU as well as R-P policy.

A comparison with the other policies in simulation process indicates that the proposed caching hierarchy and equation could improve caching system by increasing Hit ratio and Byte Hit ratio, both of which tend to rise with time.
References