Movie Retrieval System based on video ranking

Yueh Wu, Ting-Yu Lin, Peng-Jhih Lin

Abstract

In this project, our goal is to design a movie retrieval system[1]. We aim to simulate movie retrieval system from scratch without using any load balancing module. In our design, there would be a master and several servers. Servers are used to deposit movies. The master would Keep all the servers' condition and choose the most appropriate server to download the requested video. Besides this efficient distribution mechanism, we have also ranked the movies based on their accessed times. Top ranked movies would be replicated from one server to other servers so as to handle high volume of accesses. Our proposed system could handle the movie requests more efficiently and less time consuming. We present systematic analysis and comparison of our method on the average access time of the movies.

I. Introduction

To design a system for movie retrieval, it is very important to have a scalable system that could distribute the request to different machines and process the data. Take Netflix for example, it is impossible for this website to merely use one machine to deal with thousands of requests and process the data at the same time[2]. This is the reason why we need load balancer. It could be used to improve the distribution of workload in order to maximize throughput, minimize the latency and avoid overload on a single machine.

In order to design an ideal load balancer, we first think of a traditional proxy-based load balancing[3] solution. First of all, we assume that we use two instances as server and its respective database, one of them is a master node and the other is a slave node, whose database is synchronized with the master. These two instances have the same code base and same database. The only difference is that slave database is used only by Read operation while master database is used by both Read and Write operation. Second, we would have a load balancer on top of it. It would distribute workload randomly to different instances. Then, we also apply sticky session to route the request to the same instance for a particular session if the session is created before.

This load balancing method seems to work for movie retrieval system. However, it is not scalable at all and couldn't distribute and handle the requests efficiently. Therefore, we surveyed other papers and implemented a new architecture. We implement a master to keep the resource usage of all the servers and appropriately distribute the requested movies to each server. Video ranking algorithm is implemented in each server to rank the movies based on the access frequency and the bandwidth. Based on the video ranking, the highly ranked movies would be replicated when the loading of a server is higher than a threshold. This could help to save the time to access the popular movies.

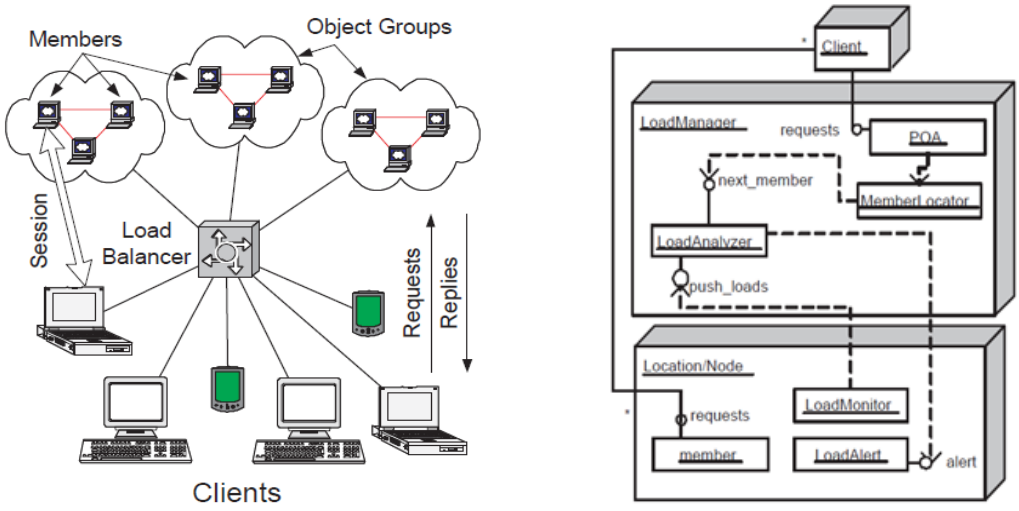
The structure of this report is organized as follows. Section 2 discusses the related work about movie retrieval system. Section 3 illustrates the design of our movie retrieval system. Video ranking and video replication mechanisms are illustrated. Section 4 is the implementation of our system. Section 5 is the experimental setup of our system. It explains the way we analyze our system. Section 6 is the result of our analysis. Section 7 is our final conclusion.

II. Related Work

Many systems rely on a powerful load balancer to gain resilience against delays and achieve high availability of the system. In fact, systems we daily use implement some specific load balancing strategies[4]. In this section, we explore different kinds of the load balancing algorithm in real world services.

A. Cygnus Load Balancing

Fig. 3 depicts the architecture of the Cygnus Load Balancing Framework[5][6]. The load manager acts as a mediator that helps clients and servers to make load balancing decisions. The member locator is to connect a client to a server. The load analyzer is to decide which server will receive the next request. The load monitor will make the reports of the load for the load manager that helps it to make load balancing decisions. The load alert is a mediator between the load balancer and the server. It will prevent the server from load shedding instructions. The load balancing methods used in Cygnus framework are *RoundRobin*, *Random*, *LeastLoaded*, *LoadMinimum*.



(a) (b)

Fig. 1. Cygnus Framework. (a) Overall architecture. (b) Load balancer structure

B. Ribbon

Ribbon [7] is part of Netflix Open Source Software family and provides software load balancers to communicate with cluster of servers. The main features that Ribbon offers are as follows:

- Supply the public DNS name or IP of individual servers to communicate client
- Rotate among a list of servers according to certain logic
- Establish affinity between clients and servers by dividing them into zones and favor servers in the same zone to reduce latency
- Keep statistics of servers and avoid servers with high latency or frequent failures
- Keep statistics of zones and avoid zones that might be in outage

Ribbon could be departed as three components [8]: Rule, Ping and ServerList. Rule component would make a determination to which server to return from a list based on algorithms; Ping component would check whether a server is active at that time or not; ServerList Component is a state of load balancer, which is either static or dynamic. In this survey, we would focus on the Rule component.

After viewing several kinds of load balancing methods, we decided to construct a new load balancing algorithm based on our goal, which is to design it for a Movie Retrieval App like Netflix, video trailers possess largest portion of data. The reasons why we decided to design a new algorithm are as follows. First, we tried to implement those two algorithms to our movie retrieval system but it is actually not scalable. Since we only have five servers in our latest system's version, these two algorithms, however works efficiently only with much more servers. Second, the integration of these two algorithms and our systems are not match. In Cygnus, the methods provided are not ideal with our movie retrieval system because we prefer our load balancer be like a master not only a mediator. In Ribbon, the load balancing algorithm is definitely built for a movie retrieval system but again our servers are not clusters scale. It will be more difficult to modify entire code than create a new algorithm. Third, we are actually unfamiliar to analyze entire system's performance with managing communication between load balancer, servers, and clients by RESTful API. These are the reasons why we decided to construct a new load

balancing algorithm but not modifying an existing one.

III. Design

In this section, the structure of the movie retrieval system is introduced, including the architecture of master and servers, movie ranking algorithm, and movie replication mechanism.

A. Architecture

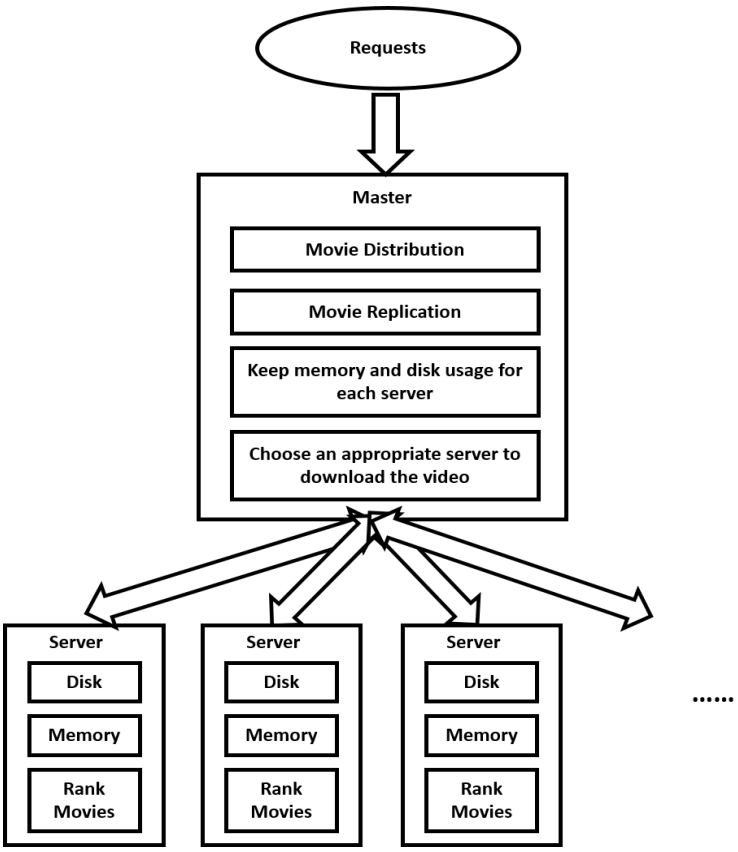


Fig. 2. Movie Retrieval Architecture.

The master would gather all the movie requests. It would initially distribute movies requested to each server. Each server consists of disk and memory. The access time for memory is a lot faster than the disk. It would also rank movies based on the number of accesses and bandwidth periodically. The master would keep memory and disk usage of each server. It would have a memory table to record the servers that own a specific movie in memory. It would choose an appropriate server based on memory table and the loading of the server to download the video. The master would also replicate frequently accessed movie from a server to other servers to make it more accessible.

B. Video Ranking Algorithm

In this part, we introduce the concept of video ranking algorithm[9] which aims to determine the ranking of movies in each server. It allows us to arrange placement of movies whether in disk or memory in order to minimize the latency of accessing request. Basically, the main point of this algorithm is to sort number of requests of each movie descendingly in a server. If the number of requests of two movies are identical, it would recalculate average bandwidth and size of movies to retrieve updated bandwidth, and then apply the respective bandwidth to sort the ranking. The symbols of parameter are illustrated in Table 1, while the detail of the ranking algorithm is illustrated in Fig. 3.

A_j^i	Access Request of j^{th} video on i^{th} proxy server
R_j^i	Rank of j^{th} Video on i^{th} proxy server
S_j^i	Size of j^{th} Video on i^{th} Proxy server
N_j^i	No. of video on j^{th} disk of i^{th} proxy server
B_j^i	Bandwidth of j^{th} Video on i^{th} Proxy server
S_{avg}^i	Average size of all video on i^{th} proxy server
B_{avg}^i	Average Bandwidth of all video on i^{th} proxy server

Tab. 1. Parameter and definition of algorithm

B. Algorithms to Determining Rank of Videos:

1. Algorithm Video Ranking ( )
2. Input: $A_j^i,\ S_j^i,\ N_j^i,\ B_j^i$
3. Output: Rank of video in ascending order R_j^i
4. Start
5. For each video,
   - 5.1. Calculate
     $$\frac{A_j^i * 100}{\sum_{j=1} A_j^i} \quad A_j \text{ is Access request of video } j$$
   - 5.2. Arrange the above values in descending order and assign R_j^i as index of each video starting from 1.
   - 5.3. In case of same index check the following condition
     - 5.3.1. $S_{avg}^i = \sum_{j=1}^{N_j^i} S_j^i \Big/ N_j^i$
     - 5.3.2. $B_{avg}^i = \sum_{j=1}^{N_j^i} B_j^i \Big/ N_j^i$
     - 5.3.3. $B_j^i = S_j^i * 100 \Big/ (S_{avg}^i * B_{avg}^i)$
   - 5.4. Arrange the index in ascending order to remove conflict of same rank.
6. End.

Fig. 3. Detail of video ranking algorithm. It takes access request, size of movie, number of movies, bandwidth as input in order to retrieve ranking of movie in each server.

C. Video Replication Mechanism

To cater the high demand of some popular movies, we implement replication strategy[10]so as to make the movies more accessible. Periodically, the master would check the loading of each server. Once the loading of a server is higher than a threshold, the movie that is mostly accessed in the server would be downloaded by the master. The master would choose appropriate servers to put the movies replicated. Replication is a time-consuming process. Therefore, we only do this process when a server's loading is very high so as not to influence the performance of servers. By implementing this strategy, it could not only make use of the storage of each serve but also enhance the efficiency of processing popular movie requests.

IV. Implementation

This section is the detailed implementation of the proposed structure. The following illustrates the structure of our code, initialization and some basic unit settings.

A. Structure of Code

To prove our hypothesis that movie ranking algorithm would work on load balancing. We simulate movie placement and load balancing on distributed servers on Python. Four classes: *Main*, *Master*, *Server*, *Utils*, were created to fulfill the simulation. The functionalities of each class are illustrated below:

1. Server

It is used to implement movies insertion and movies access. It is also responsible for manipulating movie ranking algorithm. All the data, such as load usage, movie list and ranking are stored dynamically in this class. Last but not least, we also synchronize time with other servers, and use priority queue to release loading of request and de-replicate least frequent movie based on global time.

2. Master

The main purpose of Master class is to allocate requests to server which not only has the movie but has

higher ranking. If more than two servers satisfy this condition, it would choose the one with the smallest load. Nevertheless, what Master do is more than load balancer. It is also in charge of updating global time, and sending load update, replicate/de-replicate requests to servers.

3. Main

Main class is like a wrapper to parse the arguments from command, and imitate clients by sending movie request to Master.

4. Utils

All the functionalities that are unrelated to Main, Master or Server. Such as random sampling method of movie, load, and size.

B. Initialization

Before starting the experiment, we would first initialize Server object and allocate movies. We assume that the movies are uniformly distributed in all Servers and the size of movies is randomly chosen in a specific range. All the initialization steps are done in Master.

C. Basic unit

To simplify the calculation and avoid unit conversion, we define the basic unit of data is 1 Megabyte and the basic unit of time is 1 Second.

V. Experiment Setup

In order to evaluate the performance of our load balancing algorithm, we design and conduct a series of experiments. In this section, we will describe how we setup the experiment and explain the reason. While we change the param to conduct the experiment, the other params are fixed. Here is the fixed param we used: number of movies = 1000, number of requests = 1000, memory disk ratio = 100, request distribution = exponential distribution. The followings are the params we changed:

A. Number of movies

Since the number of movies is one of the critical factors that will affect our movie retrieval system, we will set this variable from 1 to 2000 to test our system's accessing time. By doing this we could analyze our system's capacity.

B. Number of requests

Normally, a better load balancing algorithm means that it could allow more clients and more requests during the same period. Therefore, we will set the number of requests from 1 to 5000 and test our system's performance. By doing this we could analyze our system's efficiency and reliability.

C. Memory speed and disk speed

Other than number of the movies and the requests, server itself might be the reason that change the performance of our system. We will set the memory/disk speed ratio to see that if the server actually affects our load balancing's efficiency.

D. The distribution of requests

We will set the requests with uniform distribution at the beginning. However, the requests in real world might not be uniform distributed so we would also set the requests with exponential distribution and see the difference.

E. Rank algorithm

Rank algorithm is the main idea we would like to apply in our load balancing algorithm so it is definite that we should test its performance. In each params that we decided to change below, we will test them both with rank algorithm and without rank algorithm.

VI. Result

In Fig. 4-6, we show our result by comparing average access time with/ without movie ranking algorithm. It is obvious that regardless of changing number of requests, movie, or memory/ disk speed ratio, access time with our proposed method is lower than those without algorithm. These figures prove that applying this ranking algorithm did optimize movie servers by allocating high ranking movies on memory and replicate high ranking movies in prior to lower ones.

In addition to comparing access time with/ without algorithm, there are some noteworthy observations. First of all, we found that the number of requests is in monotone ascending order at Fig. 5. However, the number of movie and memory/disk speed ratio don't follow this order while we increase their quantity. We hypothesis that this is due to time unit(1 second) we apply and high statistically variance. For instance, if it took 25 seconds to transmit a movie on disk, it would take 1 second to transmit on memory for both ratio=30, and memory/disk speed ratio=100 since the basic time unit is 1 second.

Secondly, if we assume a few popular movies were downloaded intensively by applying exponential distribution on movie request, reduction of average access time is larger than those applying uniform distribution. Such a condition is illustrated in Fig. 7(a)-(b). We can observe that the gap of function chart is larger in Fig. 7(a), while the gap of function chart is smaller or even overlapping in Fig. 7(b). We note that intensive request would cause starvation on those popular movies and applying ranking algorithm would mitigate this effect.

Last but not least, we find that the average time are huge when we apply large value on parameters. It is due to the assumption of taking 1 basic time unit, which is 1 second, to deal with 1 request. This cardinality is too large to simulate real world specification. If we applying smaller cardinality, optimization by movie ranking algorithm would be more significant.

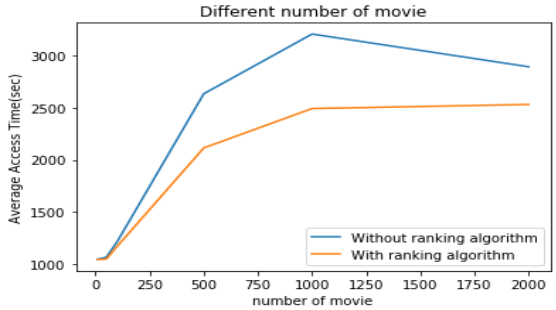


Fig. 4. Different number of movie

Fig. 5. Different number of request

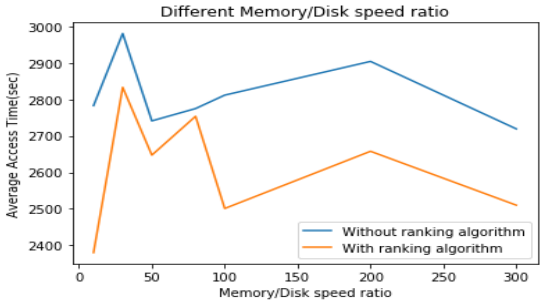


Fig. 6. Different Memory/Disk speed ratio

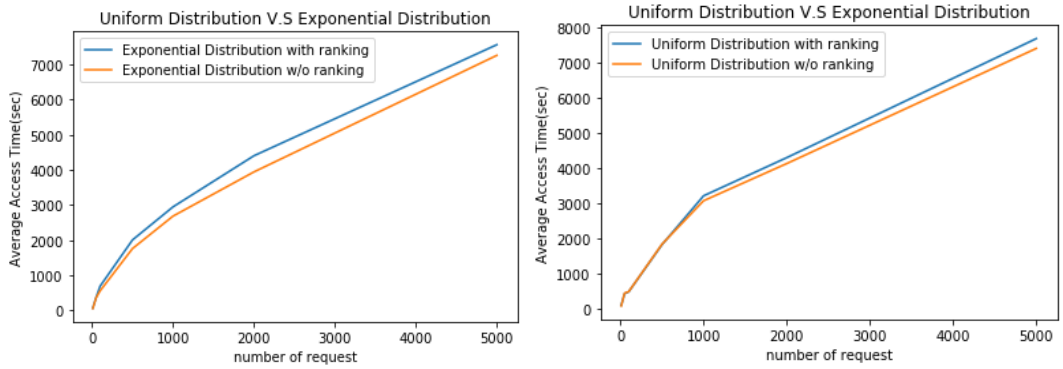


Fig 7. Comparison of Distribution (a) Exponential Distribution, (b) Uniform Distribution

VII. Conclusion

In this project, we propose to apply movie ranking algorithm on servers for movie retrieval system. The simulating code is built from scratch to evaluate how this algorithm affect accessing time. According to the result, integrating this algorithm with memory caching and replication not only reduce the average access time in different extent but also achieve scalability. As for the possible extension, we wish to put this ranking algorithm into practice by integrating it into real servers. Although developing a simulation code is the first step to prove it will work, it still makes some assumptions either simplifying mechanisms or ignoring failures.

Reference

[1] T. Kunieda and Y. Wakita, "Package-Segment Model for movie retrieval system and adaptable applications," *Proceedings IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, 1999, pp. 944-948 vol.2*.

[2] J. Summers, T. Brecht, D. Eager and A. Gutarin, "Characterizing the workload of a netflix streaming video server," *2016 IEEE International Symposium on Workload Characterization (IISWC), Providence, RI, 2016, pp. 1-12*.

[3] Fox A., Goldberg I., Gribble S.D., Lee D.C., Polito A., Brewer E.A. (1998) Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot. *In: Davies N., Jochen S., Raymond K. (eds) Middleware'98. Springer, London*

[4] Kaur, Shubhinder & Kaur, Gurpreet. (2015). A Review of Load Balancing Strategies for Distributed Systems. *International Journal of Computer Applications. 121. 45-47. 10.5120/21644-4985*.

[5] Jaiganesh Balasubramanian, D. C. Schmidt, L. Dowdy and O. Othman, "Evaluating the performance of middleware load balancing strategies," *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004., Monterey, CA, USA, 2004, pp. 135-146*.

[6] Othman, Ossama, Jaiganesh Balasubramanian and Douglas C. Schmidt. "The Design of an Adaptive Middleware Load Balancing and Monitoring Service." (*2003*).

[7] Netflix/ribbon [Online]. *Available at: https://github.com/Netflix/ribbon (Accessed: 12 June 2019)*.

[8] Netflix/ribbon/Working with load balancers [Online]. *Available at: https://github.com/Netflix/ribbon/wiki/Working-with-load-balancers (Accessed: 12 June 2019)*.

[9] S. Dhage and B. B. Meshram, "Disk load balancing and video ranking algorithm for efficient access in video server," *2012 International Conference on Communication, Information & Computing Technology (ICCICT), Mumbai, 2012, pp. 1-6*.

[10] Database Replication Technical White Paper [Online]. *Available at: https://websmp106.sapag.de/~sapidp/011000358700001121852012E/assets/Syb_ReplicationTechnical_WP.pdf (Accessed: 12 June 2019)*.