University of Plymouth

PEARL **https://pearl.plymouth.ac.uk**

Faculty of Science and Engineering School of Engineering, Computing and Mathematics

2019-02-19

Cache Performance Optimization of QoC Framework

Laghari, AA

http://hdl.handle.net/10026.1/13269

10.4108/eai.13-7-2018.156594
EAI Endorsed Transactions on Scalable Information Systems
European Alliance for Innovation (EAI)

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

EAI Endorsed Transactions
on _____ Research Article EAI.EU

Cache Performance Optimization of QoC Framework

Asif Ali Laghari[1], Hui He[1,*], Rashid Ali Laghari[2], Asiya Khan[3], Rahul Yadav[1]

[1]School of Computer Science & Technology, Harbin Institute of Technology, Harbin, China
[2]School of Mechatronics, Harbin Institute of Technology, Harbin, China
[3]School of Engineering, University of Plymouth, Plymouth PL4 8AA, U.K.

Abstract

The main aim of this paper is based on the cache performance test of the QoC: quality of experience framework for cloud computing on the server. QoC framework is based on the server-side design and implementation of the use of hierarchical architecture. Reverse proxy technology is used to build a server cluster, which is composed of front-end access layer to achieve the server for load balancing, improve the performance of the system and the use of built-in distributed cache server. The cluster consists of the cache acceleration layer, which reduces the load of the backend database. The second database server cluster, which is constructed by the database master and slave synchronization technology, forms the data storage layer, which realizes the database read and writes separation and data redundancy. The server-side hierarchical architecture improves the performance and stability of the entire system, and has a high degree of scalability, laying a solid foundation for future expansion of system business logic and increases user volume. This paper presents new cache replacement algorithm for inconsistent video file size and then analyses the specific needs for the multi-terminal type of QoC framework, and gives the client and server-side outline design; it describes the implementation details of the client and the server-side and finally the whole system of detailed functional and performance testing.

Keywords: Load balancing, Cache management, QoE, QoC, Video platform, Cache replacement algorithms.

Received on DD MM YYYY, accepted on DD MM YYYY, published on DD MM YYYY

Copyright © YYYY Author *et al.*, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi: 10.4108/_____
corresponding author email: hehui@hit.edu.cn

1. Introduction

Hypertext transfer protocol is used for communication of server-side and client applications JSON-type data used for encapsulation of data communication [1, 2]. Server-side side interface is designed for the convenience of the administrator to perform the operation of management and the client side site made for user operations. Through the entire server, the architecture can be dividing into front-end operation server and database server. The main work and operation of front end of the server is work for the client and the information exchange when the client sends an information request. The front-end operation of the client requests information contained in the client to resolve to understand the command to get the corresponding limits of the command and send a command with limits to the back-end database server for the query. When the database server returns the processing result, the front-end server encapsulates the returned data again into a JSON string and returns the message to the client via the HTTP protocol [3]. On this basis, a content server is build, the user uploaded by the client all the video content for storage and processing.

The proposed QoC framework does the function of monitoring the internal cloud environment, the client device and middle network environment from cloud to end user's device [4]. The quality of experience/service (QoE/S) data submitted by end users and objective QoE/QoS data collected by the system will be analysed for service delivery according to SLA. The proposed QoC framework distinguishes the negative and positive QoE by comparison of current service delivery parameters. The QoC framework will upgrade policy for the time being if the user does not get QoS according to SLA and extend package limitation for users to complete current task. The proposed Quality of Experience framework for cloud computing (QoC) designs and implements a multi-type terminal for the video service platform. The client can run on multiple types of smart

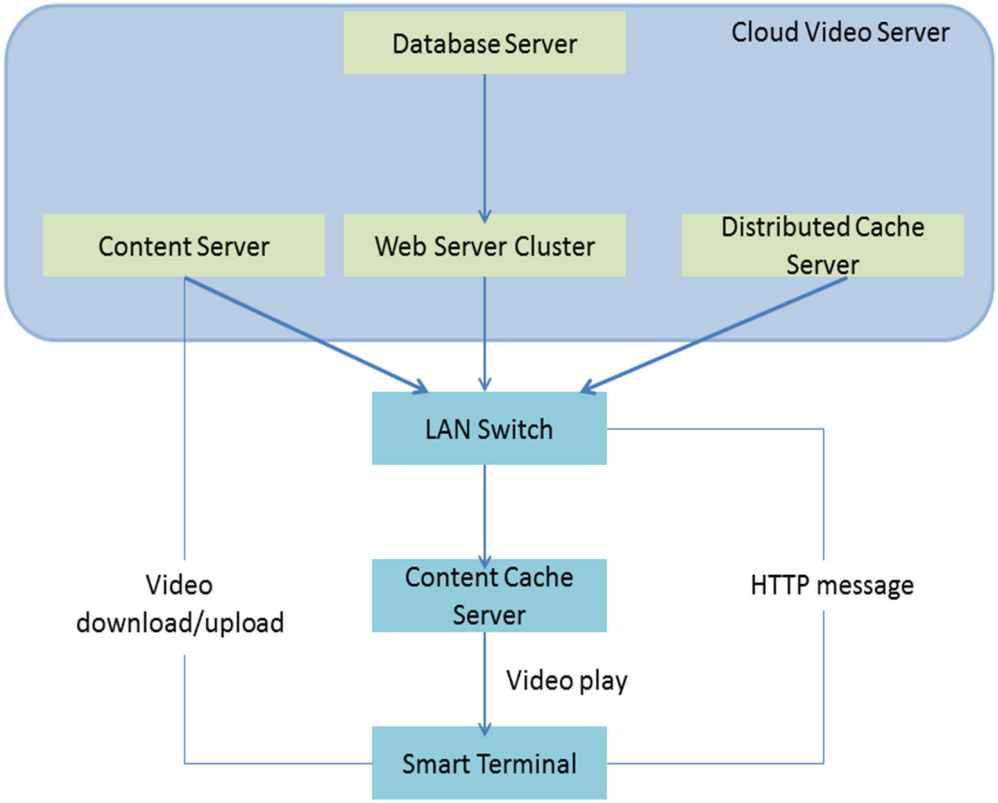

Figure 1 Simple architecture for multi-terminal type video service platform

terminals, such as the Android app and PC platform to offer user login and register, the personal information management, recommended the video, video search, view and play video, video reviews, video uploading and downloading The customer's end set up and other eight functional modules, allowing users to anytime, anywhere through a variety of intelligent mobile terminal to use the video service platform services. Simple architecture for multi-terminal type QoC video service platform is given in Figure 1.

The motivation of this work is to address the major issue of QoC framework in terms of cache management on the server-side to handle more requests and provides fast response to users on their requests. Therefore, there is a need to analyze the performance of QoC video framework on a server in single and cluster mode. Beijing Sixiang Time Technology Co. Ltd provides a technological solution and provides free servers for a test of websites and performance analysis so we use their server to test our QoC video framework [34]. In this paper, we test the performance of the QoC framework on the server and gradually improve the system's ability to resist the limitations and solve the system defects to meet the real needs of the video service platform after optimization. Finally, give the server the most reasonable and optimized basic design.

The rest of this paper is organized as follows: Section 2 describes related studies, section 3 provides details of design of server and cloud video server and section 4 is based on the front-end access layer implementation. Sections 5 and 6 presents the implementation of cache and data storage respectively. Section 7 describes realization of business logic and section 8 is based on the system performance testing. Finally in section 9 we conclude our work.

2. Related Work

QoE is used to collect reviews about product or services [36, 37]. QoE based video cache management scheme for cellular communication has been proposed by Wang et al. [9]. In the proposed scheme three parameters under consideration were a base station, client and RAN cache server for provision of quality of service of the video under limited cache capacity and statistics of video popularities. During the experiments, the author developed relationship and recording between the QoE value and three parameters such as the response from the cache server, request rate from client and bandwidth air interface. The first step of experiment cache assigned to different video clips according to their reputations and second step based on the

optimization of cache space allocation for different video clips based on the QoE value relationship, bandwidth and request rate based on the different models. The results show that video cache management scheme provides better QoE performance under constraints of total cache capacity, specific distributions of the request rate and the bandwidth. Hierarchical video cache scheme for the wireless cloud was proposed by Ahlehagh and Dey [10]. The purpose of a hierarchical video cache scheme is to fetch videos from CDNs, which decrease the backhaul traffic, increase the network capability to provision more concurrent video request with better video QoE. The hierarchical model of cache management increases the network capability by allowing many cell sites to share the hierarchical levels of the cache without increasing the total cache size, thus improving the cache hit ratio. The hierarchical cache scheme provides better mobility when the user moves from one cell to another cell with a video session; it is likely that video downloaded is already in the cache with the radio access network (RAN) or CDN associated with the new cell. The results show that using hierarchical video cache scheme hit ratio improved by 24% and network capability up to 45% compared to caching only in the RAN.
In [11], Hoang et al. proposed a mechanism named as Movie Atom Caching (MAC) which increases the user experience by reusing previously downloaded metadata atoms and cache metadata of mp4 movies at video players before users play the video. The user behaviour model avoids long startup delay and increases the QoE of users who leaving or abandons their video streaming before the video start to play. The results showed that MAC improves startup delay significantly in MP4 file streaming applications and have reasonable startup delay with modern fragmented streaming schemes with very little difficulty and overhead.

3. Cloud Video Server

The server side consists of three roles: the server, the database server and the content server.

3.1 Web server

Web server is mainly responsible for receiving and sending HTTP messages [5]. After receiving the HTTP request sent by the client, the server will analyse the JSON data, get the result from the query database, encapsulate the result as JSON data, and return it to the client as the response of the HTTP message. In the special request, such as upload video, download video and play video, interaction with the content server is required.

3.2 Database server

The database server is responsible for storing the entire system in addition to all the information outside the video file [6]. After the Web server receives the HTTP request from the client, it will return the result to the client by querying the database.

3.3 Content server

Content server stores all uploaded files of users after transcoding, multiple copies of video may exist with the different resolution of the transcoding. When users play video, through HTTP progressive download way and direct access to the content server specifies the resolution of the video file.

The basic structure of the content server is the number of users, small flow, low degree of concurrent and low load applications can basically meet the needs of users. But when the number of users increases on the system then traffic is also increasing, the system stability requirements increase, and there will be the following defects [7].
(1) Server and content server stand-alone hot: all requests from the client are handled by the server, all upload, and download from the client and play video requests are handled by a content server. When the number of user requests increases, the traffic continues to increase, it will reach a single server of the calculation and storage resources limit, a single server response speed and concurrency will be greatly reduced.
(2) A single point of dependence on the three types of servers: Web server, database server and content server, once a server rock situation, the entire system services will be affected and cannot work properly.
(3) System security is poor: the front of the Web server and content server, if attacked, will make a lot of consumption of computing resources that impose an impact on the entire system services [8].
(4) The lack of cache server: front-end server to receive each message, basically need to interact with the database, including add or delete data, which data query operation, high repeatability, many queries may obtain the same data, which will produce extra expenses. On the other hand, when the number of users increases, the user will upload and play the request each time, will visit the same content server, and will inevitably cause the user to block, so the need for the content server also increases the corresponding content cache server.

3.4. Hierarchical Design of the server

The hierarchical design of the server, making the system architecture is very clear and easy to carry out independent design and development work at all levels, but also reduces the maintenance and upgrading of the pressure [12]. At the same time, in the hierarchical design of the server-side scalability is also greatly enhanced [13].
The server architecture is divided into three layers from top to bottom:

3.4.1 Front-end access layer

A Nginx reverse proxy server and many Apache Tomcat server components [14]. Nginx server used for traffic distribution and load balancing; Apache Tomcat server used for business logic to give the operating environment.

3.4.2 Cache layer

Cache layer composed of two parts, one composed of multiple Memcached distributed cache server composed of cache acceleration layer and back-end database query structure cache which reduces the load on the database. The other part is the multi-content cache, the server composed of the contents of the cache layer and the content server in the video file cache, the content server load to more than one server to meet the purpose of load balancing [15, 16].

3.4.3 Data storage layer

Mainly composed of two parts master and slave MySQL database server cluster and file server. The master-slave data cluster uses the master-slave replication mechanism of the database so that read and write operations to the database can do separately, reducing the load on each database server in the entire database server cluster and improving the performance of the entire database server cluster. The File server is mainly used to store all users after uploading the video file after transcoding them.

4. Front-end access layer implementation

4.1 Reverse proxy load balancing

Load balancing refers to the load (work tasks) to balance, spread to multiple operating units to do, to work together to complete the task Load balancing has two meanings: First, many of concurrent access or data traffic sharing to multiple nodes on the device separately to reduce the user to wait for the response time; second, a single heavy load operation to share multiple nodes on the device to do parallel processing, each node device processing is complete. The results will summarize, returned to the user, the system processing capacity has been greatly improved. There are many ways to solve server-side load balancing, where the reverse proxy is one of the most important ways. The reverse proxy refers to the proxy server to accept external network connection request, and then send the request to the internal network of a server, and the results of the server after processing to return to the external request to connect the user, then the entire server the cluster represented as a server for external users.

The system uses Nginx server to achieve reverse proxy. Nginx is the same as the engine X, a high-performance HTTP and the reverse proxy server developed by Russian programmers for the Rambler search engine. At present, China's Nginx server users use Sina, Netease, Tencent and other large network sites. Its features are less memory consumption, concurrency, support rewrite rules, built-in health check function, and high stability. Many operating systems are supported, including FreeBSD, Linux, Solaris, MacOS X, and compiled versions support a series of operating systems. The QoC framework uses the Linux operating system.

4.2 Nginx installation

Nginx is an installation configuration file and is very simple, but also supports Perl syntax. Firstly the Nginx installation package downloaded, the current version is 1.0.2 version, and the installation of the source code has ngix-1.0.2.tar.gz. By default, Nginx will install in / usr / local / ngix, by setting the compiler option, the installation directory can modify. The installation process is as follows:

```
#tar zxvf nginx-1.0.2.tar.gz
#cd nginx-1.0.2
#. / configure -prefix = / home / nginx -user = asif
#make
#sudo make install
```

Nginx's installation is over

5.3.3 Nginx configuration

Nginx configuration files are mainly composed of events module, HTTP module, and server module configuration [17, 18]. The configuration of the events module configures Nginx's working mode and the maximum number of connections allowed. The following modes work: select (standard mode), poll (standard mode), kqueue (efficient mode), epoll (efficient mode), / dev / poll (efficient mode). In this system, select the epoll mode of operation. As shown below:

```
events {
use epoll;
worker_connections 50000;
}
```

The configuration of the HTTP module mainly refers to the configuration of Nginx as a server, including upload file size restrictions, gzip compression, server name hash table size, default file type and so on as shown below.

```
upstream backend {
server 192.168.1.100: 8000 weight = 1 max_fails = 3 fail_timeout = 30s;
server 192.168.1.100: 8000 weight = 1 max_fails = 3 fail_timeout = 30s;
server 192.168.1.100: 8000 weight = 1 max_fails = 3 fail_timeout = 30s;
server 192.168.1.100: 8000 weight = 1 max_fails = 3 fail_timeout = 30s;
}
```

Through the upstream field of the reverse proxy server is responsible for the set [19], backend Nginx is responsible for the server cluster, the cluster has four Apache Tomcat server, IP was 192.168.1.91 ~ 192.168.1.94, work at 8000 ports. The weight parameter indicates the weight assigned by the server, where all Apache Tomcat servers have the same weight, and each machine can access with the same probability. The parameters max_fails and fail_timeout-cooperate to control how Nginx determines that a server in

the upstream is invalid. When the fail_timeout time, a server connection failed max_fails times, the server will be considered a failure. At the same time will no longer be distributed to the failure of the server, to ensure the reliability of the entire server cluster.
http module configuration refers to the configuration of the Nginx virtual host, the corresponding configuration is as follows:

```
server {
listen to 50000;
server_name localhost;
root / home / nginxroot;
location / vsReq {
proxy_pass http: // backend;
proxy_set_header Host $ host;
proxy_set_header X-Forwarder-For $ remote_addr;
}
Error_page 500 502 503 504 / 50x.html
Location = /50x.html {
Root html
}
}
```

The above configuration indicates that the listening port of the reverse proxy server is 50000 and the server named as local host and will send to the backend server cluster for processing.

5. Implementation of the cache layer

5.1 Cache Acceleration Layer

5.1.1Memcached Introduction

Memcached is a high-performance distributed memory object cache server for dynamic web applications to reduce the load on the database [20]. It reduces the number of reading databases by caching data and objects in memory, providing dynamic, database-driven websites. Memcached based on a hashmap that stores key-value pairs. All operations, including insert, update, and delete, are done via the key. Its characteristics are as follows:
(1) The agreement is simple. Memcached server-side and client-side communication uses text-based protocols instead of complex XML formats. You can also access Memcached via Telnet to insert, update, and delete data.
(2) Based on the libevent event handling. libevent is an event triggered by the network library, that can support select, epoll, kqueue and other system call management time mechanism for Windows, Linux, BSD and other platforms.
(3) Built-in memory storage. In order to improve the speed of access, Memcached will cache objects stored in memory. The advantage is that access is very fast and the disadvantage is to restart the software or system and the cache data will be lost. When the memory space allocated by Memcached is exhausted, the program removes the most frequently used cache objects according to the LRU (Least Recently Used) algorithm [21].
(4) Memcached server-side is independent of other Memcached and will not communicate with each other to share the cache information. Therefore, the distribution of the entire cache cluster depends on the implementation of the client.
Since Memcached data is based on a hashmap that stores key-value pairs, it has a very strong scalability [22]. If the cache capacity of the distributed cache server cluster of the current system is exhausted or too complicated, it is very convenient to extend the cache capacity of the entire distributed cache server cluster by adding an appropriate number of Memcached servers.

5.1.2 Memcached distributed algorithm

Memcached server does not communicate with each other, the server cannot share the cache information, so the entire cache cluster distributed algorithm depends on the realization of the Memcached client [23, 24].
The following describes the principle of its distributed algorithm; the implementation of each client is basically the same. Assuming that there are three Memcached servers, node1 ~ node3, the application needs to save the data named "Tokyo", "China", "Canada", "American". The first application adds "Tokyo" to "Tokyo" to the client, and the algorithm implemented by the client will select the server that saved the data. After selecting the server, the server is ordered to save "Tokyo" and its value. Similarly, "China", "Canada", "American" is the first choice of the corresponding server, and then save. Get the data, the first need to get the key "Tokyo" passed to the client, the client according to the same algorithm to calculate the server to save the data. And then send a command to the server to obtain the appropriate data. As long as the algorithm used to save and get the same, select the server is also consistent. Unless the data is deleted for some other reason, the corresponding saved data can get.
The Memcached client has many ways to do this, there are two commonly used distributed algorithms [25, 26]:
(1) Algorithm based on remainder
According to the remainder method of dispersion is very simple, refers to the number of servers according to the remainder of the dispersion. Use the hash function (such as CRC) to find the key value of the hash, and then divided by the number of servers, according to the remainder to select the target server.
The remainder of the calculation method is simple, the data has also been very good dispersion, but also has its shortcomings. That is when the server cluster in the server because of the failure to remove or adds a new server, the cost of cache reorganization is very large. Because the addition or removal of the server, the number of servers from N to N-1 or N + 1, the hash value in the spare time, the change will be very large, so you cannot get the same server with the save, will cause a lot of cache loss.
(2) Consistent Hashing algorithm
The Consistent Hashing algorithm finds the hash value of the Memcached server information (such as IP: port) and configured on a ring of 0 ~ 232. And then use the same way to find the need to store the data hash value, but also

mapped to a location on the ring from this position to start the clockwise search, the data saved to find the first server. If more than 232, no server has been found, the data will be saved on the first server.

After adding a server, according to the remainder of the algorithm because of preservation of the key server will all change and affect the cache hit rate. However, in the Consistent Hashing algorithm, only the corresponding key values on the first server in the counter clockwise direction of the server's location is affected. The Consistent Hashing algorithm can greatly limit redistribution of key values in the case of a change in the number of servers, which has little effect on the hit rate of the cache [27, 28].

5.1.3 Memcached deployment

Four servers used to build Memcached server cluster, IP address was 192.168.1.110 ~ 192.168.1.113. The following describes the installation process.

First install the dependent libevent library, the process is as follows:

```
#tar vxf libevent-2.0.21-stable.tar.gz
#cd libevent-2.0.21
#. / configure -prefix = / usr / local / libevent
#ake && make install
```

Then install Memcached, the process is as follows:

```
#tar vxf memcached-1.4.10.tar.gz
#cd memcached-1.4.10
#. / configure -prefix = / home / memcached -with-libevent = / usr / local / libevent
#make $$ make install
```

Finally start Memcached server, allocate 2GB of memory, listening port for 12000, the maximum number of concurrent connections to 256.

```
#. / memcached -d -m 2048 -p 12000 -c 256
```

5.2 Caching Substitution Algorithm for Content Caching Layer

The content cache layer refers to the contents of the cache video file server [29]. Its architecture is similar to the content distribution network and each content cache server to the different video users to the server, the content cache server content may also repeat. When the user sends a video playback request, the server will return a video playback HTTP address pointing to a content cache server, the client to the server to initiate a play connection [30]. If the file does not exist in this cache server, the file copied from the content server. When the cache server storage space is full then there is a need for cache replacement algorithm to select the existing cache file to delete.

Common cache replacement algorithms based on a premise that the size of each file is consistent [31, 35]. In the system each cache file, that is the size of the video file is inconsistent. Common cache replacement algorithms may not well adapted to the current system. In order to solve the problem of cache algorithm failure caused by file size, this system has developed a set of file-size cache replacement algorithm which adapts to the system environment. The following describes the specific implementation of this algorithm.

SLRU cache is divided into two segments, a probationary segment and a protected segment. Lines in each segment are ordered from the most to the least recently accessed [32]. Data from misses add to the cache at the most recently accessed end of the probationary segment, to sort the information of all the video files on the cache server in the order of the most recent access time [33], and then the file size to select the files that need to replace. To achieve this, all caches are divided into two areas. S-LRU algorithm 1 is given below.

Algorithm 1. Cache Page Replacement

```
Input: video file, size of file,
Output: Cache Page Replacement for new video
    Initialized request
    q = vide file
    s = size of file
    If ( q in cache)
      {
        Goto : LRU stack
      }
    Else
    if (s > remaining storage space)
      {
        Download video from content server to
    Cache server
      }
     Else
      {
        Get beg s from LRU stack
        Delete q
      }
    LRU Stack
    End
```

When the storage space is not enough time, the algorithm first selects the sorted stack in the old file size of the largest video file to remove. When the first file removed, the remaining storage space is still not enough to accommodate the new video file, continue to follow the above way to select the file removed until the remaining space is enough to accommodate the new video file. When the storage capacity of the cache server is different from the total content of the content server, the selection of the parameters is different.

6. Implementation of Data Storage Layer

The system uses the database master-slave mode to achieve the server-side metadata storage. In the master and slave database server cluster, there are three MySQL database servers, one for the Master database (192.168.1.120) and the other two as the Slave database (192.168.1.121 and 192.168.1.122). Writing to the database (insert, delete, and update) occurs only on the master database, and the read

operation (query) occurs only on the slave database. Read and write operations are from the smart terminal client request.

Mysql's replication mechanism can synchronize data on the master database and the data on the slave database. The specific implementation is when the master database is on the write operation, the write operation will be recorded in the binary log file, through the synchronization thread on the server to synchronize the binary log to the slave server, modify the data on the slave server to achieve data consistency between master and slave databases.

Compared with the stand-alone mode, the master-slave mode of the database has the characteristics of load balancing, reading and writing analysis and high data security, and is suitable for the environment of the system.

7. The Realization of Business Logic

The server receives the HTTP message sent by the system client, Restful Web Services determines the type of the request according to the URL address, and calls the corresponding module to deal with, including the string parsing, database query, JSON string encapsulation and other operations to complete The entire business logic, the package will be a good string as the corresponding BODY, back to the client. The following describes the module processing flow.

The client sends the HTTP request to the client, and the client divided into eight modules: registration module, user information management module, video upload and download module, video information module, video comment module, recommended video module, video search module, set the module. The eight modules in addition to set the module will interact with the server side. Most of the modules of the process are the client to the server to start data requests, the server from the database query to get the right data and the data packaged in a format, through the response to the client. The client performs further processing and presentation based on the data obtained. The following two sections (sections 8 & 9) describe the detailed flow description of the registered registration module in which the interaction process is more complex.

The registration login module divided into three parts: send verification code part, registration part and login part. The process of sending the verification code is given in algorithm 2.

Algorithm 2. Server Verification Code

Input: Login ID,
Output: Verification Code

```
1   Get facility from JSON
2   For(  facility  in database)
3   {
4     If ( Facility = Register )
5       {
6         Send Identification code
7          Status =0;
8       }
9     Else
10        Status = 1;
11  }
12  Set Status in JSON
13  Return  to client
14  End
```

8. System Performance Testing

8.1 Front End Access Layer Performance Test

This section uses Apache's own performance testing tool AB to perform performance testing on the front-end access layer. The test is divided into two parts, a single server for stand-alone mode test and load balancing under the cluster mode test. By simulating large-scale requests, the average response time in standalone mode and cluster mode is obtained in the case of different concurrency numbers. The test results are shown in Figure 2, where the horizontal axis represents the number of concurrent and the vertical axis represents the average response time in milliseconds.

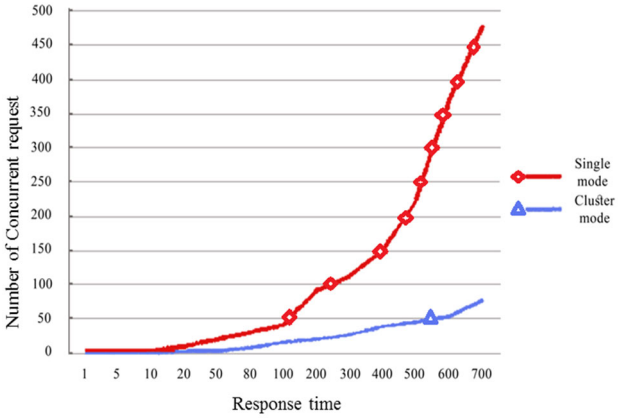

Figure 2. Performance test of front end access layer (x-axis = number concurrent and y axis = response time)

As can be seen from Figure 2, the average response time of stand-alone mode and cluster mode is not much different in the case of low concurrency. From the data analysis, the performance of stand-alone mode is slightly higher than cluster mode. In the cluster mode, the reverse proxy server is responsible for traffic distribution. With the increase in the number of concurrent, single-mode performance greatly reduced, but the performance of cluster mode can be maintained in a relatively stable state.

8.2 Cache layer performance test

This section uses libmemcached and the memaslap performance test tool provided by the source library to perform a performance test on the cache acceleration layer. The test object is Memcached distributed cache cluster, composed of four servers, working in 12000 ports, the memory space used by 2GB. The test is divided into multiple groups, each group of cache size is different, and the length of the key is fixed 16 bytes. For the set cache operation (set) and get the cache operation (get) were tested.

When the number of concurrent times is 100, the system throughput will have a different program of decline, encountered a bottleneck. Because the memory has reached the upper limit, and the server's network card traffic is close to the limit. Also from Figure 2, the server's cache data is smaller, the greater the throughput of the system. The cache acceleration layer, which consists of the server, exhibits good performance and is able to maintain good performance in high concurrent situations.

8.3 Data layer performance testing

The tests in this section are divided into two parts: testing a single server in stand-alone mode on a database server; testing the server cluster in cluster mode on the two database servers. Through the preparation of database testing procedures, simulation database read operation, the use of multi-threaded technology to simulate concurrent operations.

The results of the test are shown in Figure 3. The abscissa indicates the number of concurrent threads. The ordinate indicates the response time of the database read operation in Figure 3.

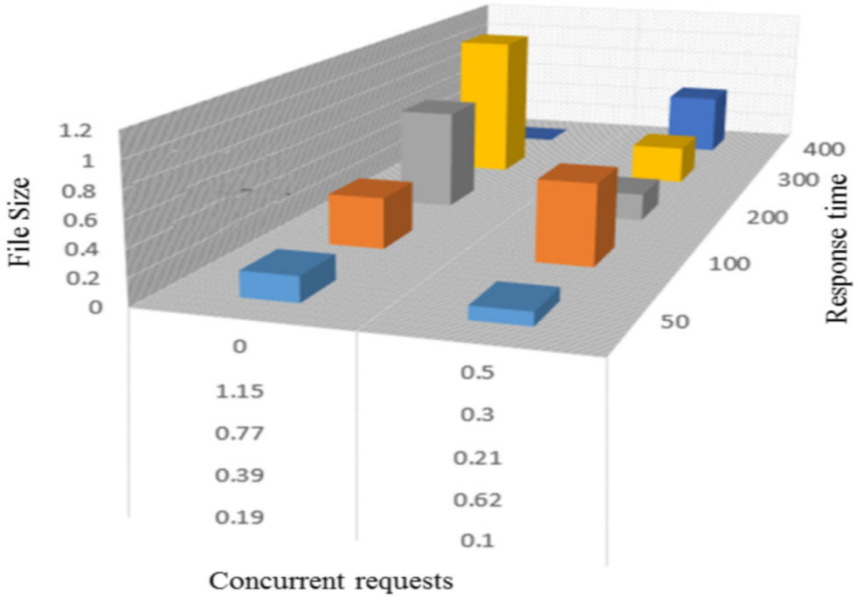

Figure 3. Data layer performance test results

As can be seen from Figure 3, in the stand-alone mode, with the increase in concurrency intensity, a single response time increases rapidly, and when the number of concurrent increases, the single response will reach the limit and cannot respond. In cluster mode, as the concurrency intensity increases, the response time increases slowly, but the performance is better than the stand-alone mode.

8.4 Cache replacement algorithm test.

For the test of the cache replacement algorithm, it divided into two parts. The first part is the comparison with other classical cache replacement algorithms. The second part is the performance change of the cache replacement algorithm under different parameter selection.

Use the program to simulate the test, assuming that the number of video files on the content server is 1 million, the average size of the video file is 100MB, and then the size of all the files on the content server is about 100TB. The access rules for video files conform to the normal distribution.

The first part of the test simulation algorithms is S-LRU, RAND, FBR, LRU, and LFU. Among them, the S-LRU algorithm set the parameter Fold = 60. FBR algorithm Fnew = 25, Fold = 60. In different cache size, the hit rate test, the test results are given in Figure.

As shown in Figure 4 the cache capacity is from the time, the hit rate than many other algorithms are much higher. In other words, when the cache capacity is not large than RAND is a better choice. When the cache capacity gradually increases, the hit rate is not the highest one algorithm, but compared with several other algorithms, the difference is not great. In the real cache system, the parameter Fold can be dynamically changed to achieve better performance by dynamically scaling the cache capacity and content capacity.

9. Conclusion

This paper presents the test of QoC video framework on the server system, through the text analysis and function screenshots in the form of a complete test of the project involved in the three modules of the specific functions. Through the performance test of the server side different cache algorithms were used, each layer of the hierarchical structure can run normally according to the design and can run normally in the case of high concurrent volume and the processing ability can meet the current business needs, and the ability to expand the business needs. After testing, the system can be in normal operation, the three modules can be a normal collaboration between the completions of the established demand targets of users. Through the test section, we can also more clearly understand the various functional modules of the project.

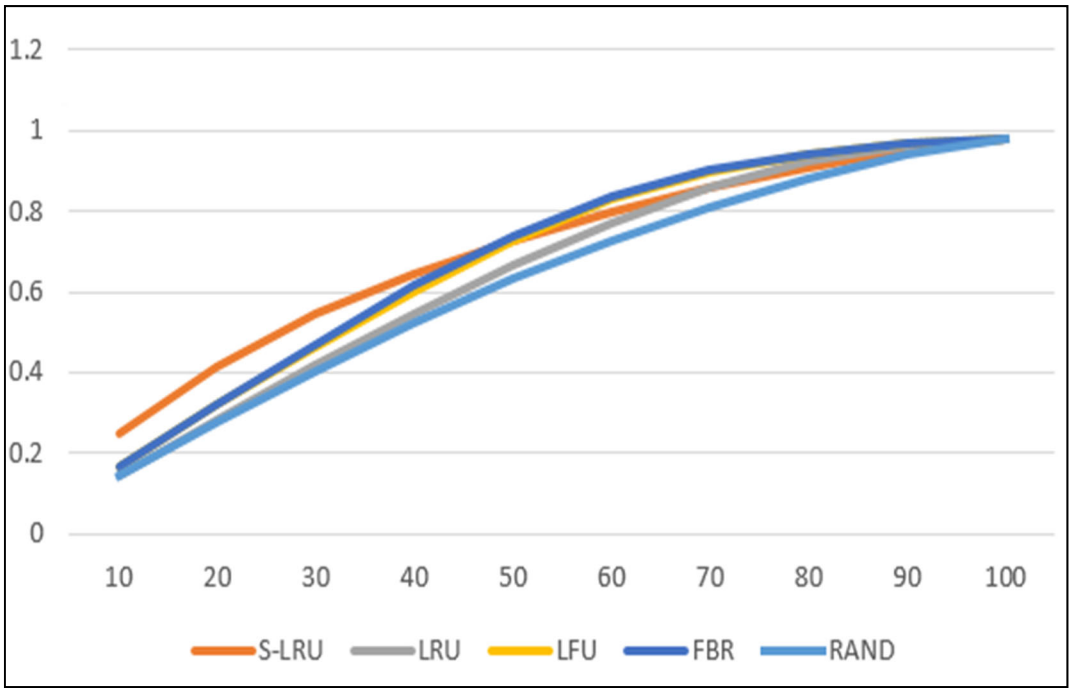

Figure 4. Replace the algorithm hit rate test chart x-axis is time and y-axis is file size

Declaration

Availability of data and material: No

Acknowledgments

Professor Hui He is corresponding author of this paper.

Funding

This work is supported by the National Key R&D Program of China under Grant no. 2017YFB0801801 and the National Natural Science Foundation of China (NSFC) under Grant no. 61472108 and Grant 61672186.

Author Contributions: First author has conducted the research and written the paper, the third author has set the template and formatted of paper. Rest of the authors reviewed the paper to set context and contributed as experts in the field.

Competing Interest: Declare conflicts of interest or state “The authors declare no conflict of interest."

References

[1] Mills, R. L., & Newman, D. M. (2013). *U.S. Patent No. 8,595,613*. Washington, DC: U.S. Patent and Trademark Office.

[2] Carlson, M., Martin C., Alex H., Scott H., Duncan J.-W., Anish K., Tobias K. (2012). "Cloud application management for platforms." *OASIS, http://cloudspecs. org/camp/CAMP-v1. 0. pdf, Tech. Rep.*

[3] Zhang, J., Liu, W., Zhao, W., Ma, X., Xu, H., Gong, X., ... & Yu, H. (2018). A Webpage Offloading Framework for Smart Devices. *Mobile Networks and Applications*, 1-14.

[4] Laghari, A. A., He, H., Khan, A., Kumar, N., & Kharel, R. (2018). Quality of experience framework for cloud computing (QoC). *IEEE Access*, *6*, 64876-64890.

[5] Wang, P., & Chen, X. (2017, November). Co_Hijacking Monitor: Collaborative Detecting and Locating Mechanism for HTTP Spectral Hijacking. In *Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence & Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2017 IEEE 15th Intl*(pp. 61-67). IEEE. *Congress (DASC/PiCom/DataCom/CyberSciTech), 2017 IEEE 15th Intl*, pp. 61-67. IEEE, 2017.

[6] Cantelon, M., Harter, M., Holowaychuk, T. J., & Rajlich, N. (2014). *Node. js in Action* (pp. 17-20). Greenwich: Manning.

[7] Wu Yuesheng , L. X. (2010). Eclipse 3.0 application development technology explains [M]. Tsinghua University Press, 128131.

[8] Ullman . (2009)Database system based tutorial. [M] YUE Li-hua translated. Machinery Industry Press, 251254.

[9] Wang, Y., Zhou, X., Sun, M., Zhang, L., & Wu, X. (2017). A new QoE-driven video cache management scheme with wireless cloud computing in cellular networks. *Mobile Networks and Applications*, *22*(1), 72-82.

[10] Ahlehagh, H., & Dey, S. (2012, June). Hierarchical video caching in wireless cloud: Approaches and algorithms. In *Communications (ICC), 2012 IEEE International Conference on*(pp. 7082-7087). IEEE.

[11] Hoang, X. T., & Nguyen, T. T. (2016). Reducing Startup Time in MP4 On-demand Video Streaming Services with Movie Atom Caching. *VNU Journal of Science: Computer Science and Communication Engineering*, *32*(1).

[12] Che, H., Tung, Y., & Wang, Z. (2002). Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, *20*(7), 1305-1314.

[13] Oliver R,.D, and Aubrey.J. U. (2003). Connecting with Java Web Services. InfoWorld., (l25): 48

[14] Bates, A., Hassan, W. U., Butler, K., Dobra, A., Reaves, B., Cable, P., ... & Schear, N. (2017, April). Transparent web service auditing via network provenance functions. In *Proceedings of the 26th International Conference on World Wide Web* (pp. 887-895). International World Wide Web Conferences Steering Committee.

[15] Xiang, L., Ng, D. W. K., Islam, T., Schober, R., Wong, V. W., & Wang, J. (2017). Cross-layer optimization of fast video delivery in cache-and buffer-enabled relaying networks. *IEEE Transactions on Vehicular Technology*, *66*(12), 11366-11382.

[16] Li, R., Zhang, J., & Shen, W. (2018). Replicas Strategy and Cache Optimization of Video Surveillance Systems Based on Cloud Storage. *Future Internet*, *10*(4), 34.

[17] Nedelcu, C. (2015). *Nginx HTTP Server*. Packet Publishing Ltd.

[18] Soni, R. (2016). Introduction to Nginx Web Server. In *Nginx* (pp. 1-15). Apress, Berkeley, CA.

[19] Guangji, B . (2007). Java programming tutorial examples. Beijing: Metallurgical Industry Press.

[20] Li H. (2003). Management information system development and application. Beijing: Electronic Industry Press.

[21] Mokhtarian, K., & Jacobsen, H. A. (2017). Flexible caching algorithms for video content distribution networks. *IEEE/ACM Transactions on Networking (TON)*, *25*(2), 1062-1075.

[22] Sa, S., X., and Shan W,. (2000) "Introduction to database system."

[23] Li, Z. (2010). Application of MVC pattern in data middleware. *Computer Engineering*, *36*(9), 70-72.

[24] Jose, J., Subramoni, H., Luo, M., Zhang, M., Huang, J., Wasi-ur-Rahman, M., Islam, N.S., Ouyang, X., Wang, H., Sur, S. and Panda, D.K., 2011, September. Memcached design on high performance rdma capable interconnects. In *2011 International Conference on Parallel Processing* (pp. 743-752). IEEE.

[25] McGhan, H., & O'Connor, M. (1998). Picojava: A direct execution engine for java bytecode. *Computer*, *31*(10), 22-30.

[26] Fitzpatrick, Brad. "Distributed caching with memcached." *Linux journal* 2004, no. 124 (2004): 5.

[27] Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). *UNIX Network Programming: The Sockets Networking API* (Vol. 1). Addison-Wesley Professional.

[28] Bremler-Barr, A., Hay, D., Moyal, I., & Schiff, L. (2017, June). Load balancing memcached traffic using software defined networking. In IFIP Networking Conference (IFIP Networking) and Workshops, 2017 (pp. 1-9). IEEE.

[29] Su, Z., Xu, Q., Hou, F., Yang, Q., & Qi, Q. (2017). Edge caching for layered video contents in mobile social networks. *IEEE Transactions on Multimedia*, *19*(10), 2210-2221.

[30] Song L., (2007). Thread pool based WEB server implementation and monitoring [MS Thesis].: Jilin University Library.

[31] Li P., Zhu Q., (2004) . Linux design and implementation of the support of resuming multi-threaded download tools. Computer Engineering and Applications 1: 121123

[32] Markatos, E. P. (2001). On caching search engine query results. *Computer Communications*, *24*(2), 137-143.

[33] Boating. (2002). HTTP and multi-threaded download (on) Programmer Technology (2): 9294

[34] https://baike.baidu.com/item/%E5%8C%97%E4%BA%AC%E6%80%9D%E4%BA%AB%E6%97%B6%E5%85%89%E7%A7%91%E6%8A%80%E6%9C%89%E9%99%90%E5%85%AC%E5%8F%B8/623449?fr=aladdin

[35] Benhamida, N., Bouallouche-Medjkoune, L., & Aïssani, D. (2018). Simulation evaluation of a relative frequency metric for web cache replacement policies. *Evolving Systems*, *9*(3), 245-254.

[36] Laghari, A. A., He, H., & Channa, M. I. (2018). Measuring effect of packet reordering on quality of experience (QoE) in video streaming. *3D Research*, *9*(3), 30.

[37] Laghari, A. A., He, H., Khan, A., & Karim, S. (2018). Impact of Video File Format on Quality of Experience (QoE) of Multimedia Content. *3D Research*, *9*(3), 39.