**Topic 31: Distributed Systems (CDN + Storage) Design**

**Team Members**

Yani Jin ([yanijin@usc.edu](mailto:yanijin@usc.edu))

Xinyi Cai ([caixinyi@usc.edu](mailto:caixinyi@usc.edu))

Ryan Afranji ([afranji@usc.edu](mailto:afranji@usc.edu))

**Problem Description**

*Challenges especially in terms of software and algorithms development, motivation, any novelty (in terms of algorithms, math, etc.) compared to any prior work (3 points)*

Challenge 1: Figuring out what system to use. We spent the first several weeks spending all of our time exploring different systems we could use to build up our project. Unfortunately, we only could find platforms that cost money. After meeting with Akshata, we had to decide to not use up any more of our time searching for a free solution and to instead build our own system on our personal computers.

Challenge 2: Since we needed to build the system on our own personal computers, we were challenged with figuring out not only how to use new technologies, but also discovering which new technologies we should/can use to implement our project. So it has taken a lot of time learning the technologies such as node.js, html, ngrok, and others.

**Project Timeline**

*What you have done so far:*

1. Came up with a design plan for integrating CDN and Cassandra to provide higher quality service.
2. Converted a local PC into a web server which also connects to Cassandra.
3. Designed the workflow and Cassandra logical data model for our web application.

*What you expect to do by Phase II:*

1. By that time, our web application could provide a set of functionalities as we expect.
2. Build up a caching CDN which also handles load balancing work.

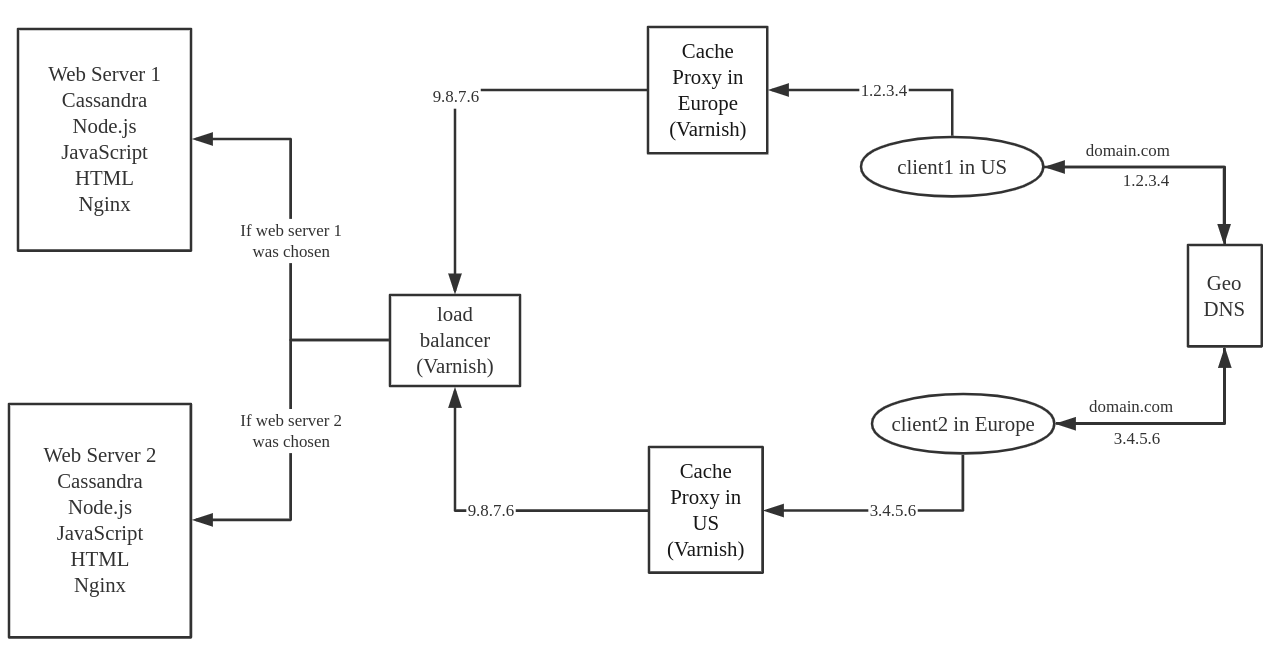
*What you expect to do by Phase III:*

1. Evaluate the performance of our system, and compare it to commercial CDNs.

**Progress****report** *for phase I, adding any information the course staff and supervisors should be aware of (any change of plan since the proposal and the reason (10 points):*

We came up with the project architecture, including Geo-DNS, cache proxies, and load balancer. We also built up a web server which could handle simple Cassandra queries for now. Meanwhile, we referred to an application called KillrVideo which is about video streaming, and started the data modeling for our own application. Based on our project architecture, we also dove into algorithms in each component, which will be listed in the next section.

**Analysis** *of your solution, framework, algorithm set, mathematical models. Analysis of time and memory complexity. For the power try your best to model the power consumption. Writing the corresponding pseudo-codes, flow charts, etc. (20 points) \*\**



The above figure shows how our system works. Clients access our web application with the domain name of our application, say “domain.com”. WIth the help of GeoDNS, requests will go to the closest cache proxy sitting around their clients. Then the cache proxy will fetch the data from our original web server if needed. To increase the efficiency, there will be a load balancer sitting between cache proxies and original servers.

Inside cache proxy, cache replacement algorithm and cache consistency maintenance algorithm will be implemented. Although we haven't decided which algorithms we are going to use, we’ve researched potential solutions. For the replacement part, we could choose from a wide range of algorithms, such as Least Recently Used (LRU), Least Frequently Used (LFU), Pitkow/Recker. And for the consistency part, we are going to select an algorithm from following families: Server-based Two-Phase Locking (S2PL), Optimistic Two-Phase Locking (O2PL), and Callback Locking (CB).

As for the load balancer sitting between web servers and cache proxies, there are many options as well. Here is the list of known algorithms: Round Robin, Least Connections, Least Time, Hash, IP Hash, and Random with Two Choices. We will figure out the algorithm matches our project most in Phase II.

Current Software:

The following will describe the initial web server design. Currently the web server is running on a local computer. The server is Javascript-based and ran locally using node.js. There is also an instance of Cassandra running on the local computer that has database the web server accesses.  
  
Node.js runs the server locally and through the use of ngrok, we can make this server available to the world via a public address.  
  
The web server uses a Cassandra driver which allows it to set up a communication connection between itself and the Cassandra database. When someone accesses the server using the public accessible address, a node.js web server makes a query to the Cassandra database. The result of this query is then passed to through to the web page of the client that accessed the address.

Future Database Design:

The following are queries of our application:

Q1: Find user by email address

Q2: Find user basic info by user id

Q3: Find user subscriptions by user id

Q4: Find uploaded videos by user id

Q5: Find comments by video

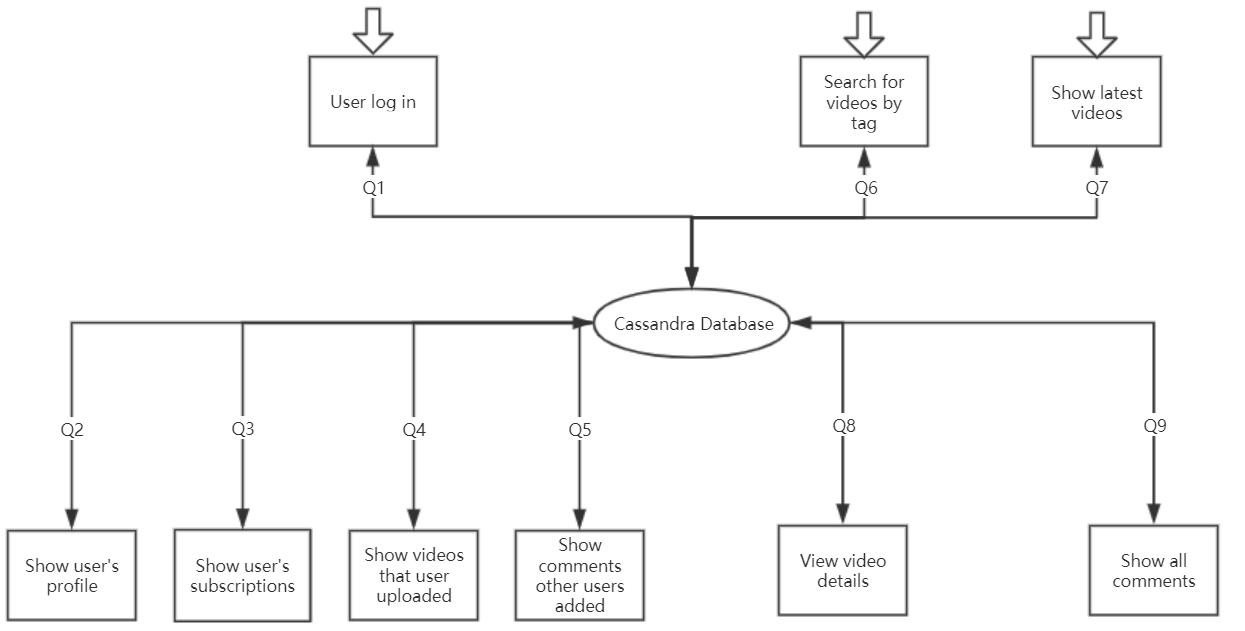
Q6: Find videos by tag

Q7: Find videos by date

Q8: Find video by video id

Q9: Find comments by video id

Based on our queries, we drew the workflow of our application as follow:



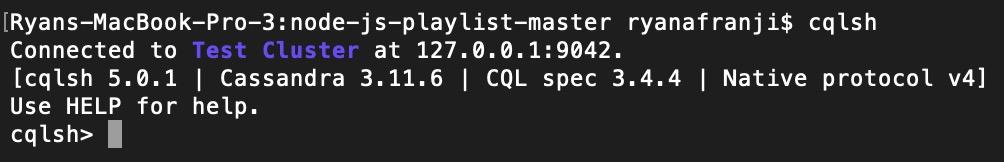
Cassandra Physical Data Modeling:

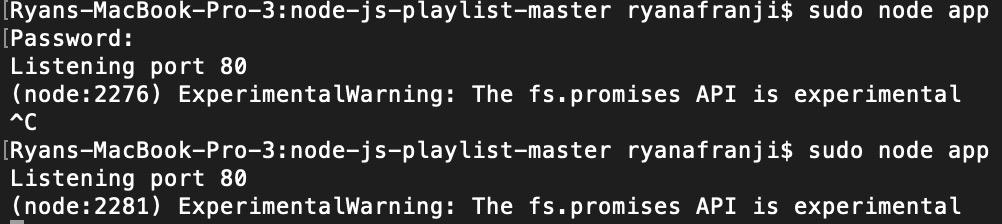
|  |  |  |
| --- | --- | --- |
| user\_credentials | users | videos |
| email | text | K  password | text  user\_id | uuid | user\_id | uuid | K  email | text  first\_name | text  last\_name | text  user\_subscription | text | video\_id | uuid | K  video\_name | text  user\_id | uuid  date | date  video description | text |

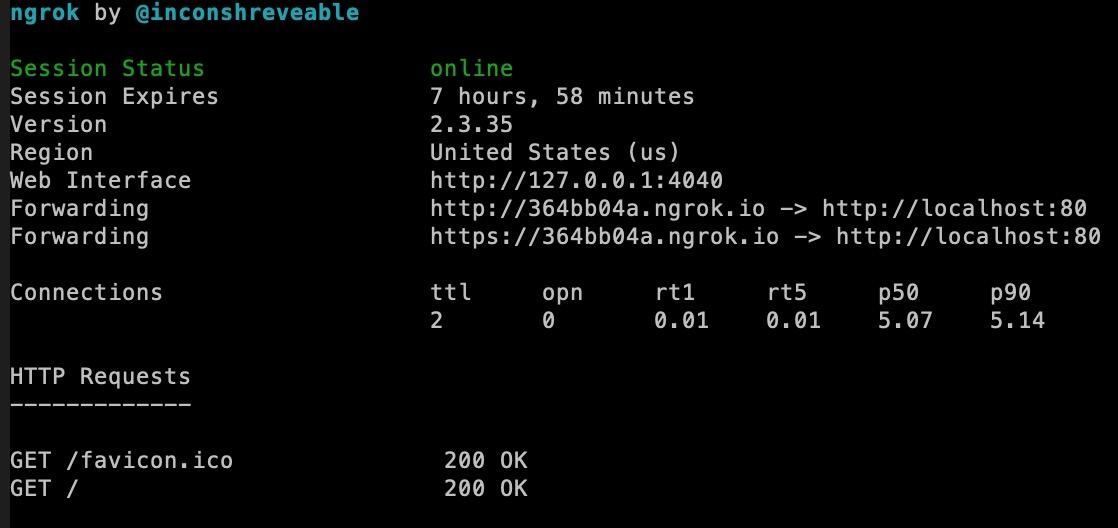
|  |  |  |
| --- | --- | --- |
| video\_by\_tag | comment | ... |
| tag | text | K  video\_views | int | C↓  video\_id | uuid | C↑  video\_name | text  user\_id | uuid  date | date | comment\_id | uuid | K  comment\_date | date  user\_id | uuid  comment description | ... |

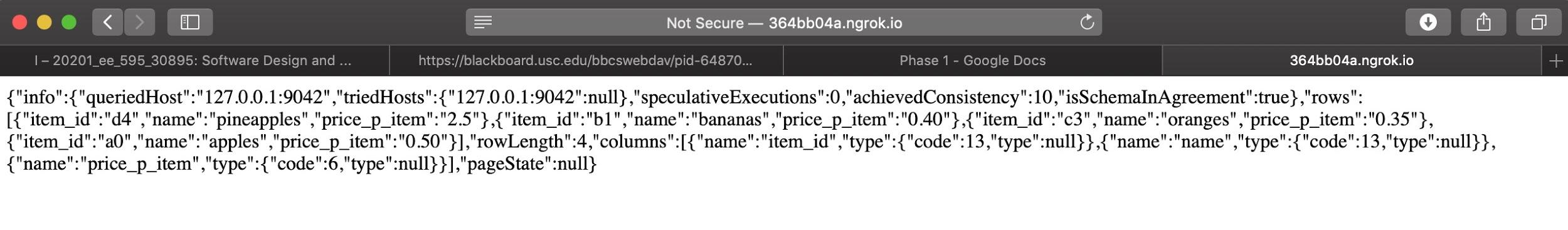
**Implementation** *(submission requirements: code, short summary of any software packages you have used, and any other information related to the implementation work) (35 points). \*\**

Screenshots from implementation of existing system:

*(Started up cqlsh to verify Cassandra is up and running)*

*(Running node.js server for our app.js file)*

*(ngrok, which was started up with ‘ngrok http 80’, is up and running giving us a publicly accessible address)*

*(Cassandra query result that was executed and displayed when accessing the node.js server via the ngrok public accessible address given by ngrok)*   
  
Software Packages Used:

* cqlsh:
  + This is a shell to interact with cassandra. I used this shell to enter a new keyspace (database into Cassandra) to test out the system.
* ngrok:
  + This package makes a web server running on a localhost publicly accessible and provides a publicly accessible address.
* cassandra-driver:
  + This package allows node.js to communicate with a cassandra database. So we use this package to make queries from our node.js web server.
* node.js:
  + Since cassandra is running locally, we also need to run our javascript code locally and not on a requested web server in order to access it.

**Writing the abstract** *(section 1) of the paper/report. Writing most of Section 2 (introduction, i.e., prior work summary, the key novelties of your work) (30 points)*

*Abstract*

End users of the Internet always want to access Internet contents faster and more stable. With the explosive growth of the variety of contents on the Internet as well as user's requirements for content delivery, internet service providers choose to use CDN to offer a better Internet service. Most websites and Internet contents that people interact with are spreading over multiple physical locations. The physical distance between the origin server and the end user will cause large transmission delay. In assistance of CDN, user's experience in terms of speed would be highly increased. Apache Cassandra database could be the right choice of database in terms of scalability and high availability without compromising performance. It supports replicating across multiple datacenters and providing lower latency at the same time. There are already many successful examples in industry using Cassandra effectively. In this paper we will discuss how to build our CDN with Cassandra database and provide content delivery service with high quality.

*Introduction*  
The internet has become one of the most important factors of human life. Every day, almost every human being has a large interaction with the Internet. One trend that continues year after year is in the increase of data consumption. A large cause of this is attributed to streaming services. There is a massive migration of people ditching their cable/satellite television providers to get their media through internet based streaming service. Another trend is that camera quality is increasing so the size of files to be distributed by these streaming services are also getting larger.

To handle significant network traffic on streaming services sites, they need to handle requests to their distributed systems in an effective way. This is done through the use of Content Delivery Networks (CDNs) and databases. An efficient tool to establish a CDN is Varnish, which is utilized in our system. When developers are making decisions on which database to use in the case of distributed systems (CDN could be regarded as a distributed system), they refer to CAP theorem and think about desired features of their system first. For an application providing world wide streaming service, it must make sure users from all over the world could access it. Also, even numerous videos getting uploaded to it, it could handle the increase of video source easily with low cost. In a word, high availability and partition-tolerance are essential to streaming service. This is the reason why we choose Cassandra as our database.

**List of Papers**

1. *Dynamic load balancing for distributed memory multiprocessors*

This paper is concerned with dynamic load balancing. This is different from static load balancing which is the user setting up how the system to divide up “tasks” it needs to among the available computing resources. Dynamic load balancing is needed the amount of tasks to be computed are not known until runtime; therefore, a technique is needed to most effectively distribute “tasks” in a way that achieves the best overall performance. The paper shows two mathematical algorithms to distribute the work: the first distributes work from one processor to another once the first processor is running at capacity and the second distributes work evenly. The paper later goes on with extensive mathematical proofs of why its load balancing algorithm is ideal. To be honest the math was quite confusing and I could not fully grasp it. However this paper serves our project well because since we are building a CDN, we may potentially use their algorithm to distribute work in our system.

1. *Dynamic load balancing on Web-server systems*

This paper discusses different approaches of how to make web sites served by a distributed system. Most importantly, for the user, it must appear to them that they are accessing a single website. The paper discusses four different load balancing approaches: client-based, DNS-based, dispatcher-based, and server-based. The paper goes into depth of how the DNS works and different load balancing techniques it uses. Hidden-weight which is the expected load a client’s request to a server is something we should look at in our project to help determine load balancing. It is suggested that we look at both the client’s current state and the server’s current state when making load balancing decisions. This project used adaptive TTLs in hope of boosting the performance of DNS. The paper later goes into delete of different communication methods which are not going to be looked at because it is not within the scope of our project. The experimental results are not really useful for comparison against our experiment but we can extract a good concept out of it. They used simulations to simulate web traffic to the systems so we should do something similar when we test.

1. *Cassandra: a decentralized structured storage system*

This paper just summarizes the system of Cassandra so there is not too much to discuss. Cassandra is not the same as a traditional database and lacks some traditional database functionality. It discusses the importance and need for databases and why it needs to be fast and fault tolerant. It presents us with some of its functionality and how it replicates and partitions data. The only thing really worthwhile making note of from this paper is that there are results from some of its queries in its experiment. This is important because this is the results from the baseline Cassandra. So we compare our changes to this.

1. *A Big Data Modeling Methodology for Apache Cassandra*

This paper introduces a query-driven big data methodology, i.e., given a set of application functionalities, think of all needed queries first, then come out with logical and physical data model and design an efficient schema on this basis. It’s worth mentioning that it introduces several desirable data modeling principles, mapping rules and mapping patterns which help transfer conceptual data models to logical data models. In addition to teaching us how to manually design a Cassandra database, this paper also recommends us to use a web based data modeling tool called KDM which automates their proposed methodology. Based on this paper and instructions on Cassandra official website, we designed our own model described in the Analysis section above.

1. *Book: Web Caching and its Applications, Chapter 7, Cache Replacement Algorithms.*

In this chapter, it not only divides popular cache replacement algorithms into three families (Traditional algorithms, key based algorithms and cost based algorithms), but also explains and evaluates them one by one. To understand a new technique, it’s safe to start with traditional solutions. Here I only summarize those traditional algorithms: 1) LRU, which disgards the least recently used items first, 2) LFU, which counts how often an item is used and discards the least frequently used items first, 3) Pitkow/Recker adds a new rule based on LRU, i.e., treats those content accessed on the same day as they have the same recency.

1. *Book: Client Data Caching, Chapter 5, Performance of Cache Consistency Algorithms.*

To maintain the consistency among cache servers, not only are cache replacement algorithms important, but also are consistency algorithms needed. This chapter summarizes and evaluates three families of those algorithms which I have mentioned in the Analysis section. Among them, it was said that the Callback Locking (CB) algorithm performs better, so that we are considering using CB as the cache consistency maintenance algorithm for our cache proxies. Furthermore, it discusses some tradeoffs we might meet when we are considering maintaining consistency.

1. *A Tale of Three CDNs: An Active Measurement Study of Hulu and Its CDNs*

This paper basically shows how Hulu selects CDNs among multiple choices and how each CDN allocates resources response to the client's request. Hulu utilizes multiple CDNs to serve users’ video requests. Although Hulu frequently changes preferred CDNs for users, once a CDN is selected, Hulu would keep that CDN and delivering the target content during the entire length of the videos even when the performance of that CDN might degrade sometimes.The preferred CDN is decided on instantaneous available bandwidth observed at the client side. So in our project, we will also stick to one CDN in the process of delivering one specific content to the end user. And we are gonna measure the performance of delivering content in this CDN.

1. *Determining Network Delay and CDN Deployment*

This paper includes the way to determine network delay. The key point of this process involves identifying a set of DNS servers. These servers could be regarded as vantage points to test delay performance to a CDN content server. To identify a content server that serves a vantage point DNS server, we need to retrieve an IP address using DNS query to the DNS server corresponding to the vantage point. The delay performance between the DNS server and the CDN content server could be estimated. The physical location of the data center could be selected based on the delay performance ranking. The methods taken in this paper to measure the performance could be used in our project as well when we need to analyze our CDN performance.

1. *Analyzing the Performance of an Anycast CDN*

In this paper, the authors introduced their ways to measure and analyze the performance of an anycast CDN. Anycast is simple to operate, scalable. They are also naturally resilient to DDoS attacks. Anycast CDN usually performs better than unicast devices despite the lack of precise control. A CDN can direct the user’s requirements to a front-end in multiple ways. The route which CDN chooses could change the performance of delivering contents. When anycast method is taken, the same IP address came up from multiple locations over the world. BGP routes clients to one front-end location based on BGP’s notion of the best path. The CDN size and geo-distribution of CDN will also impact the result of analysis. In our CDN performance analysis, we need to choose a reasonable algorithm to figure out the best route for CDN. When measuring the performance, we also need to come up with factors that might impact the results.

1. *Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery*

This paper introduces a technique that collects the name and ip of addresses during DNS resolution and does the server’s owners. We potentially might need to use this process in our project. The paper details that Netflix uses the 3 CDNs of Akamai, LimeLight, and Level-3. An experiment concluded that the only time the CDN currently being used would switch to another is when bitrate fell to 0. They also observed that once an account is created, they are always associated with the first CDN assigned to them even when they move around. All 3 CDNs seem to perform relatively similarly. They propose an increase to performance by instead of always using the same CDN, choosing the CDN that has the highest available bandwidth.

# *11. World-wide-web server with delayed resource-binding for resource-based load balancing on a distributed resource multi-node network:* <https://patentimages.storage.googleapis.com/b7/5d/e9/81d14d51008caa/US5774660.pdf>

*12. A method to increase Services Availability on Web-based Inter-enterprise service delivery* platforms:

<https://ieeexplore.ieee.org/document/5662794>