Project Proposal

Data Storage / Retrieval with Access Control, Security and Pre-Fetching

Presented By:
Shashank Newadkar
Aditya Dev
Sarvesh Sharma

Advisor: Prof. Ming-Hwa Wang
COEN 241 - Cloud Computing

Table of Contents

Table of Figures

1.Introduction

1.1. Objective

A cloud computing system is a set of huge networks and computing nodes, where enormous amount of data flow takes place. An efficient performance is needed along with security to access and store data.

In order to meet the availability and confidentiality properties with performance, is the backbone of any computing system and is the basic expectation of a vendor from its Providers in Cloud Computing environment.

The main purpose of this manual is build up a system based upon procedures and algorithms to provide a full-proof system to vendors. The focus in this paper would be upon:

1. Security
2. Accessibility
3. Availability and Confidentiality Properties
4. Performance

1.2. Problem

The whole world is tied up with World Wide Web. The internet is an enormous collection of computers spanning the globe that when linked together. The amount of information accessed today is immense. Every user expects all the knowledge to be at hand with a single key-mouse click. This enforces lots of responsibility upon the Provider to meet the needs of its users and vendors.

The cloud is being considered as an efficient and economical solution for storing huge organizational data, accessible to public. The pay-as-you-use billing model has attracted many organizations, not wanting to invest into the data centre infrastructure and the maintenance cost that is associated with managing these expensive centres, towards cloud.

Though the cloud offers scalable and flexible services, the data stored on the cloud service providers' end is susceptible to attacks from unauthorized entities as well as from the service provider itself. Data owners need to be aware of these security issues and are expected to take preventive measures such as encrypting data before moving it to the cloud. The problem today is there are

not many easy to use and lightweight encryption mechanisms that could encourage data owners to go with this option.
The provider needs to keep the information confidential and secured. It also has to be kept accessible around the globe at any time of day, 365 days a year. Along with these factors; Performance of the system is also vital. It completely depends upon Data Congestion and Network Delay.

Transfer time of data intensive applications accounts for a larger proportion of the overall running time. System response less than 2 seconds is considered to be efficient.

In order to meet the expectations of World Wide Web users, a whole integrated technology needs to be developed and implemented in the Cloud System.

1.3. Why this project?

The whole learning needs to be based upon the Cloud technology and its Implementation. This project will give an insight upon minute details of the information stored and how different access properties play a role in mechanism of security and access constraints. Also, keeping in consideration the performance factors like network speed, cache data characteristics and other aspects which play a role in speed of data flow; this project gives a complete end-to-end picture of a Cloud System where a user reads data from and writes data to.
The project touches upon the essential security challenges in the cloud as well as the performance improvement scope in data retrieval speed by making use of proactive caching mechanism. Data is being stored on cloud presuming that service providers are trustworthy and will never try to access confidential data. The project attempts to provide a lightweight and flexible encryption module that will help data owners protecting their data from cloud providers.

1.4. Area or Scope of Investigation:

We would build up a system considering the important factors that play an important role in making a cloud system successful. The scope would be:

1. Data Security in Cloud.
2. Mechanism to store data in cloud
3. Mechanism to fetch data from cloud
4. Access Control Lists with respect to roles on Data

5. Performance improvement with Pre-Fetching and Caching Mechanism.

2.Theoretical bases and Literature Review

2.1. Definition of the Problem

Data security from cloud service providers is a concern today and there is no easy to use solution available today that will encourage data owners to take preventive steps such as encryption before moving data to cloud.

Also, data congestion and network delay are the important factors that affect performance of cloud computing systems. Network communication delay becomes the bottleneck of computing performance. The traditional concept is to use hot data as cache. However, it is difficult to meet the demand of data access in the cloud computing system.

2.2. Theoretical background of Problem

The solutions available for securing data from cloud providers involve encryption mechanisms that require intense communication between parties involved in encryption and decryption of the data. The key management part is handled by Public Key Infrastructure that takes care of generation and distribution of keys. But PKIs are burdensome to manage and are complex to use.

From caching point of view, the solution provided by caching the data with high hit ratio is applicable for small scale systems with low average seek time. For an implementation at the Cloud System level, cloud environment can land up into following issues:

1. Computing node uses memory as a cache; the capacity is several GB, but the current cloud computing system has a large data set, of which the volume is usually hundreds of TB even petabytes; also the active data set is greater than the cache capacity.
2. A lot of access to data on a regular basis. Take the system or service start up as an example, the computing node requires access to the storage server to get the start-up data, and these data usually will not be used again in a long time. So it is difficult to improve the performance of data access by using the cache.

2.3. Related Research to solve the Problem

The researches have come up with a lightweight and flexible encryption mechanism to secure data from cloud providers. A framework is proposed based on commutative symmetric encryption to protect the data from third parties.
In order to solve the performance problem with caching factor the researchers developed a storage cache placement algorithm - C-Aware, which traces history access information of cache and data source, adaptively decides whether to cache data according to cache media characteristic and current access environment, and achieves good performance under different workload on storage server.

2.4. Advantages of related research

Our research helped us uncover some of the persistent problems in the field as detailed under:

2.4.1. Problems:

Cloud Computing is a very nascent technology and is greatly susceptible to issues related to data security, availability, performance etc. which are in turn dependent on other factors like cloud service providers and their policies, available disk access speed, network congestions etc. and greatly vary in different scenarios.
Some of the issues like data security, performance and availability are of great concern to cloud users and companies throughout the world. Security is among the major concerns as the information stored on the cloud is often of a sensitive nature pertaining to companies and their employees, customers and transactions. Performance and availability issues lie more on the business end of matters as a more reliable and fast cloud service provider can obviously attract more customers to use their service. However, with the enormous amount of data that gets passed between servers and between servers and clients, it becomes difficult to maintain the QoS in terms of speed in file access and battling network congestion.
This often results in a frustrating experience while using cloud services and people may either move to a different service or be dissatisfied altogether.

2.5. Disadvantages of related research

C-aware caching definitely shows a very promising future but still has some way to go before it can be considered as a commercially viable option. Even

with less expensive hardware, cache components are costlier than traditional disks. Also, to experience C aware works best when the network congestion is higher and there multiple accesses to the same file from different users at the same time. Here again, there are issues pertaining to updating the metadata with file hits on file replicas in multiple servers.

2.6. Your Solution to solve this Problem

We propose a solution to provide role based access over the stored data and to handle the encryption key used to encrypt the data by a commutative encryption mechanism.

By providing role based access to users, we not only make it easier for the owner of a file to share their files among other roles but also to easily view the files shared with them by others. The usage of a commutative encryption mechanism-enabled key manager to handle keys ensures that the key control rests with the user and not the cloud service provider by allowing for a double encryption for any data that passes to the cloud.

In addition to this, the introduction and improvisation of C-Aware algorithm for caching frequently accessed files as opposed to the traditional LRU mechanism will help improve the performance even better as it takes into account various environment specific aspects like network speed, response time, media speed, and cache levels, as we consider the frequency the data is accessed and also cache the related data for the data that is in cache. This improves caching and increases accessibility time.

We propose a solution to provide role based access over the stored data and to handle the encryption key used to encrypt the data by a commutative encryption mechanism.

2.7. Where your solution differs from others

The current limited approaches to secure data from cloud providers today are either computationally heavy or difficult to handle by the average cloud user. Our comprehensive solution provides a wieldy encryption mechanism to secure users' data on cloud by enforcing role based access control to further reduce the threat surface.

Original Solution considers below factors for Performance:

1. Speed of the cache media
2. Historical information of I/O request
3. Predicts storage server performance

4. Decision making whether to store data in cache
5. Response Time to serve the request

Thinking in line of performance, we have improvised upon the current system by considering below factors too:

1. Frequency the data is accessed.
2. Maintain a 2-level cache.
3. Also Cache the related data for the cached data.

2.8. Why your solution is better

Our solution considers the communication overhead of exchanging keys among parties involved in data exchange on cloud. We make use of the framework that uses commutative encryption to protect the key from cloud providers and we build the role based access control over that framework.

As per the thumb rule followed for this paper, the data to be cached should not be based upon the number of times it has been accessed but rather should be according to current access system. Our solution for performance, considers all the properties mentioned in C-Aware algorithm and also considers a few more factors that help improvise any computing system, hence improving the performance of a cloud system.

3. Hypothesis

3.1. Positive Hypothesis

We will provide the role based access over the stored data and will handle the cipher key used to encrypt the data by commutative encryption mechanism. We will store this encrypted key on third party provider so that cloud provider don't get access to it.

C-Aware algorithm and our latest additions to it will help improve the performance even better by taking into account environment specific aspects like network speed, response time, media speed, and cache levels.

3.2. Anticipated Result

We hope to show an enhanced performance resulting from our C-Aware caching module, especially when network loads are higher, as opposed to LRU

which tends to work better when the network load is lower. The C-Aware core implements the generic logic and selective mechanisms for cache data, and relies on the history information collected by the C-Aware tracer to make decision. Based on the history information, C-Aware core heuristically anticipates which selection is better for system performance: caching data or directly sending request to source. If the anticipation heuristic is to cache current request, C-Aware will utilize a traditional algorithm to continue handle the request. Otherwise, C-Aware will directly send current request to the data source bypass system cache. In short, C-Aware considers caching data according to the combination of cache media speed characteristics and server's workload, and try to eliminate the negative effect of caching data on low-speed cache media when there is only light load on servers.

The role based access system assigns files to various roles as per the requirements of the owner. This should allow users to access the appropriate files after authentication as per their level of authority. To do this, we list out only those files that are allowed access by the user after login and restrict access to files that are not on the list.

Finally, the key manager module ensures security of encryption keys by commutative encryption of data as well as the keys. Each time an access is made to the cloud to fetch or write data, the module functions to provide an extra layer of security to maintain obscurity to cloud service provider and remain transparent to the end user who does not see anything different on their end in terms of performance or availability.

4. Methodology

4.1 Input Data

Input data is provided in XML format and then parsed and populated into database. Songs data with selected attributes is used for demonstration purpose.
Attributes: title, lyrics, duration, artist, copyright, and genre.

4.2 Problem Solution

4.2.1 Algorithm Design

We have implemented two algorithms: one for data security and other for cache performance enhancement.

Data security: Following steps describe the algorithm design:

1) When user is created a random public key is generated and stored against user's record in database. (128-bit AES)
2) At the time of data uploading, the data is encrypted with user's public key first and then stored in database.
The public key is then commutatively encrypted with random key (SHA1) generated at user's end. Commutative encryption involves XORing the two keys. This encrypted key is then forwarded to cloud provider.
3) Cloud provider generates a random key (SHA1) using a seed value and commutatively encrypts the key forwarded by user. This double encrypted key is sent back to user.
4) User removes his layer of encryption by XORing the key forwarded by cloud with the random key generated in step 3. The resultant key is then stored in separate database.
This way the original cipher key used to encrypt data is further encrypted with cloud provider's key and stored on third party storage.
5) When the user logs in to access data, his credentials are validated against the roles configured in the system, after successful validation data accessible to the designated roles are displayed. For decryption, the key stored on third party provider is forwarded to the consumer. Consumer then commutatively encrypts this key and forwards to cloud provider.
6) Cloud provider commutatively encrypts the key and removes its layer of encryption added earlier in step 3 and sends back this key to the consumer.
7) Consumer then removes his layer of encryption and gets access to the original cipher key and uses this to decrypt the data.

Caching performance: In order to design the solution, we work upon 3 major parameters for every I/O request:
Sample, presents the total requests of each type;
Total-time, records the total handle time until the last request is finished by C-Aware;
Meantime, presents average response time for different types of request, the basis for C-Aware to judge the current load.
The values are calculated as follows:

1. **totaltime(n+1)** = 7 x totaltime(n) + 256 x last_request_handle_time/ 8
2. **samples(n+1)** = 7 x samples(n) + 256 / 8
3. **meantime(n+1)** = totaltime(n+1) + 128 / samples(n+1)

From 1 it's clear, that the handle time for the most current request has the biggest weight.

Now, C-Aware needs to decide whether to cache eevry incoming read-write I/O request or access from storage disk. Below are few rules that help us decide that:

Rule 1: If the current request is read and the cache is full, Meantime (Cache Read Request) < Meantime(Source Read Request), C-Aware decides to cache data and selects a cache block to replace according to predefined replacement algorithm.

Rule 2: For write request, if cache is full, Meantime (Cache Write Request) < Meantime (Source Write Request), C-Aware will make a decision to cache current request and then make a cache block replacement.

Rule 3: If the current request is read, there are still free cache blocks, and Meantime (Direct Cache Read Request) > Meantime (Source Read Request), C-Aware would not cache the data request.

Rule 4: For write request, there are still free cache blocks, and Meantime (Direct Cache Write Request) > Meantime (Source Write Request); C-Aware would not cache the data request.

Rule 5: If the cache is full and it does not meet Rules 1 and 2, C-Aware would not cache the data request.

Rule 6: If there are still free cache blocks and it does not meet Rules 3 and 4, then C-Aware allocates a cache block to cache the data request.

Once the data is cached, the related data is searched on basis of pattern matching and other properties in terms of forming relation between the data. This related data is stored in 2nd Cache level.

Pseudo Code for Caching:

```
Cache Handle (Request){
        Get cache block from data cache according to current request
        if (cache block ==NULL){
                if (Cache is full){
                        if (Rule 1 or Rule 2 is satisfied)
                                Cache Miss (Request);
                        else
                                Access the storage network directly;
                        } else if (Rule 3 or Rule 4 is satisfied){
                                Access the storage network directly;
                        } else {
                                Cache Miss (Request);
                        }
                } else
        Cache Hit (Request);
```

}

4.2.2 Language Used

1. **Java Swing / AWT:**

   Java Swing / AWT Kit will be used for providing a platform independent Desktop based application. It will help build a User Interface using multiple GUI components.

2. **J2EE:**

   Java EE 7 SDK i.e. Java Platform Enterprise Edition is a free source development platform to build in the implementation for this project.

4.2.3 Tools Used

1. **NetBeans:**

   NetBeans version 8.1 is an open source IDE for java development which will be used along with Swing Kit.

4.3 How to Generate Output

Steps to generate output:

1. The file will be uploaded and deleted from the Storage Disk file system via User Interface
2. The performance check i.e. output will be visible in graph format to the user
3. The encrypted data stored on cloud should be displayed in original format whenever requested

4.4 How to Test against Hypothesis

We will test data security by checking system safety against unauthorized access. We will compare our results with implementation for caching with few other algorithms and strategies that involve 1 Level cache and implementation of algorithms for D-Cache and Dm-Cache.

4.5 How to Proof Correctness

By measuring the performance while uploading, deleting files and searching information from the system and comparing the results with

traditionally used implementations like D-Cache and Dm-Cache, single level cache.

5. Implementation

5.1 Database Schema

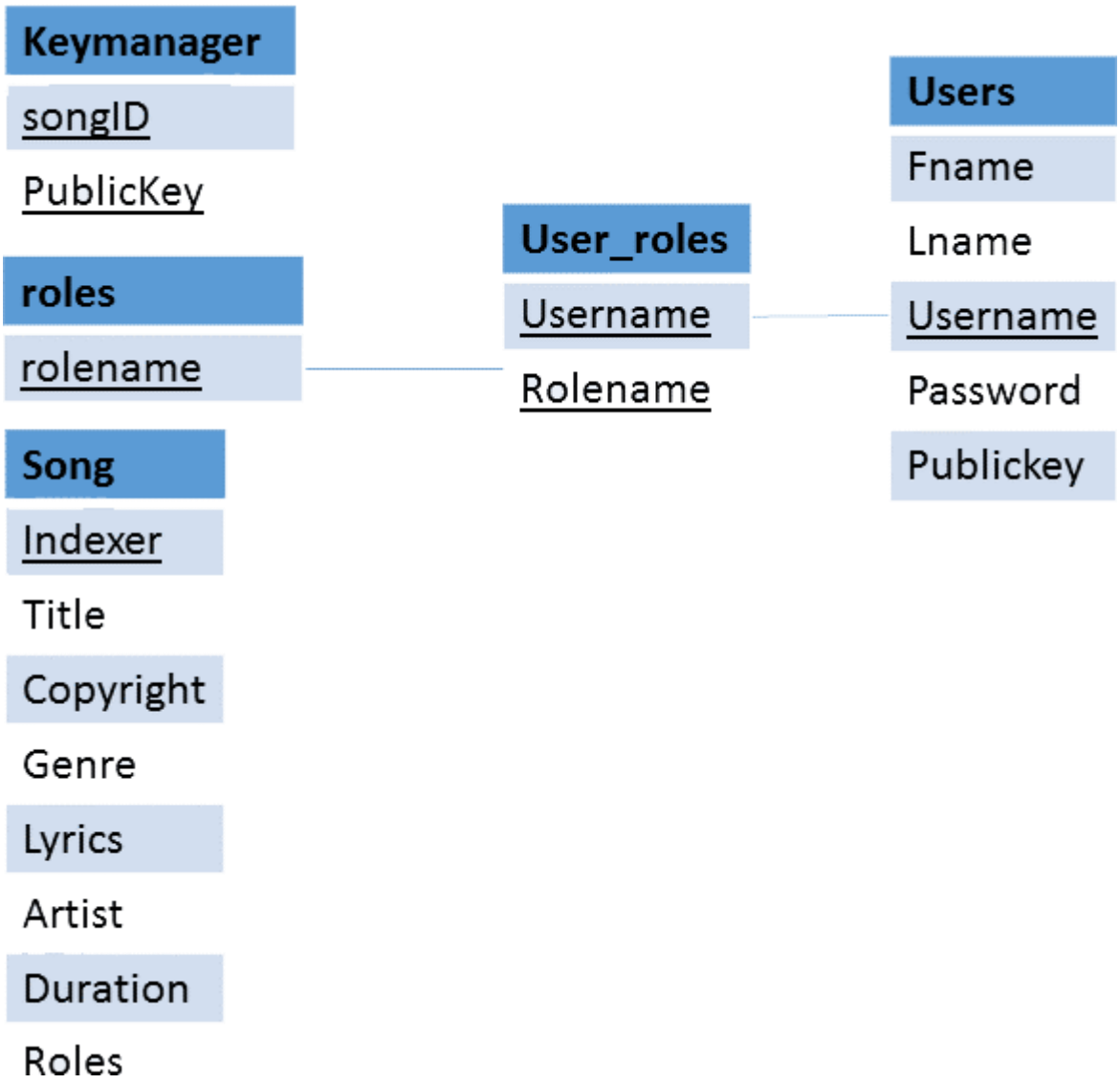

Figure 1: Database schema for this implementation

6. Data Analysis and Discussion

6.1 Output Generation

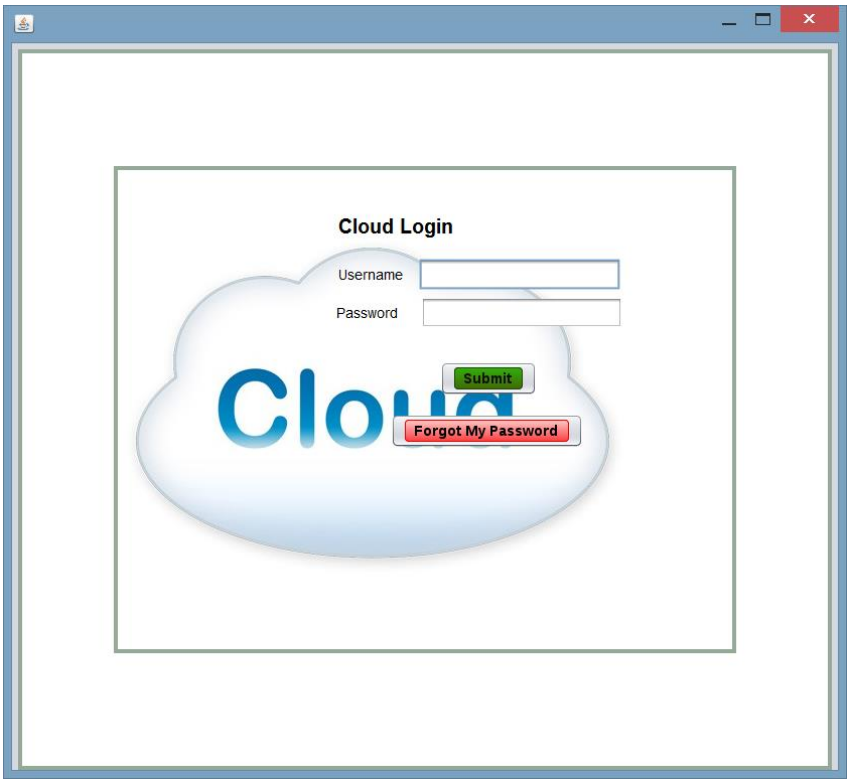

Figure 2: Sign in page – Authentication

Figure 3: Song List

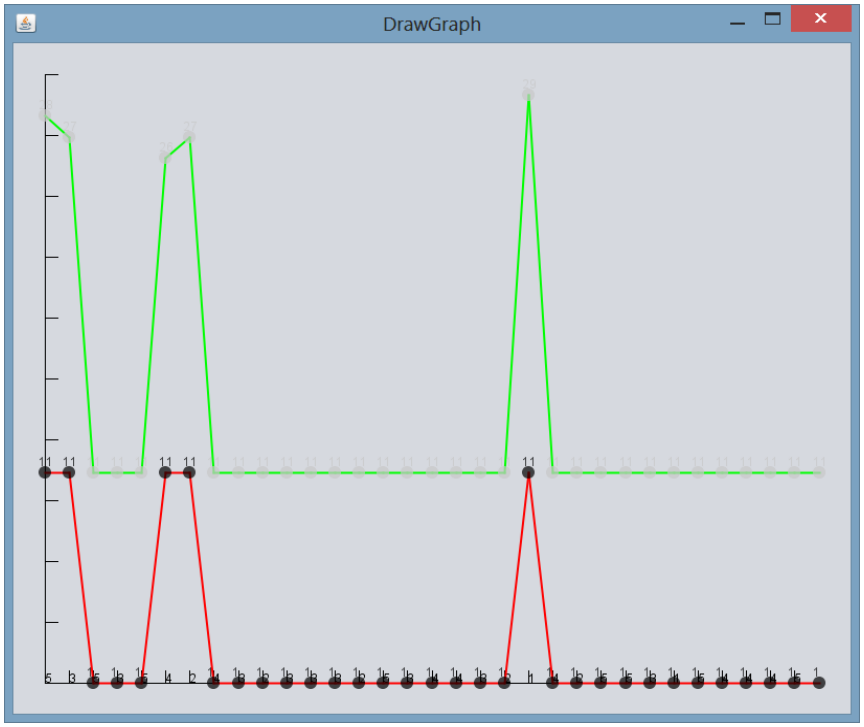

Figure 4: File access time using LRU

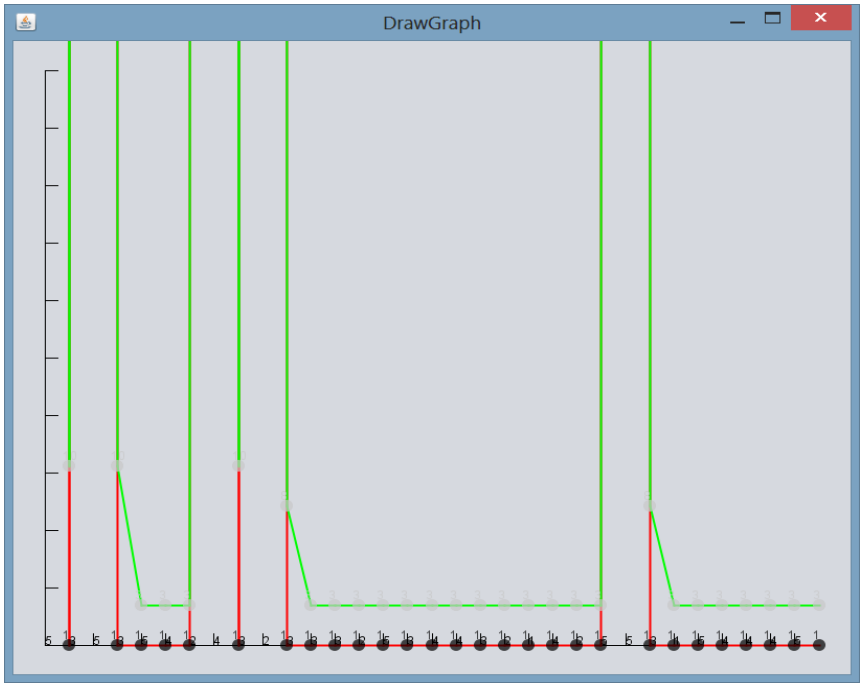

Figure 5: File access time using C-aware

7. Conclusion and Recommendations

7.1 Summary and Conclusions

In our project, we have tried to demonstrate how the usage of C-Aware can be beneficial as the amount of data being stored in the cloud servers and its set of associated issues increases exponentially each day. By measuring the time taken to retrieve the files after the user logs in and begins a search, we plotted out two graphs that take into account factors such as network load and disk speed for C-aware and LRU. The graphs clearly show the greatly reduced access time and a more optimum performance by C-aware, especially as the number of accesses to each file and the network load increases.

7.2 Recommendations for future studies

Since our solution proposes a key manager module to interface between the user and cloud during I/O operations, the introduction of C aware as opposed to traditional LRU caching can help reduce the slightly increased time required to encrypt and decrypt the files but in order that it make an overall impact, C aware needs to consider many more factors such as a tentative cost comparison between storing the data in cache and accessing it each time from disk, change in disk access speeds over time etc. which may call for a much more complex module. Nevertheless, implementation of C aware in predictable networks spanning a sufficiently large area is quite a start to begin with.

8. Bibliography

1. *Research Article* - C-Aware: A Cache Management Algorithm Considering Cache Media Access Characteristic in Cloud Computing
2. Privacy aware access control for data sharing in cloud computing environments