MS Project Report

Document Placement Schemes in Cooperative Caching

By

Shashank Murali Menon

smm8507@rit.edu

Chair: Dr. Hans-Peter Bischof

Reader: Dr. Xumin Liu

Observer: Prof. Henry Etlinger

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences

Rochester Institute of Technology

Rochester, NY

Approved by:

Dr. Hans-Peter Bischof

Professor and Graduate Program Coordinator, Department of Computer Science

Committee Chair

Dr. Xumin Liu

Assistant Professor, Department of Computer Science

Committee Reader

Prof. Henry Etlinger

Professor and Undergraduate Program Coordinator, Department of Computer Science

Committee Observer

Table of Contents

Acknowledgement

I would like to thank my advisor, Dr. Hans-Peter Bischof for helping and guiding me at each and every step during the course of this Capstone Project. I am also extremely thankful to Professor Bischof for his advice and guidance during the Independent Study course which gave me the idea for the Project. Also, I would like to thank the committee Reader, Dr. Xumin Liu for reading my proposal and giving me feedback and suggestions on ways to improve my project. I would also like to thank Professor Henry Etlinger for being the Observer of the project.

I would also like to thank Professor Alan Kaminsky for his Parallel Java library. The simulation assignments I did in the Distributed Systems course helped me immensely in the implementation of this Project.

Abstract

The advent of P2P systems for file sharing, streaming and related activities have significantly increased the popularity of caching. Caching of files can lead to increased performance and efficiency since the number of requests from a client to the server is considerably reduced. Caching in a P2P system can either be Independent or Cooperative. Cooperative caching is a widely researched topic. As compared to Independent caching, Cooperative caching is more efficient. It yields better results with a reduced latency and lesser traffic in backbone links. This project focusses on cooperative caching in P2P systems.

Also, in Cooperative caching, document placement schemes would be the main concern of this project. An important question which needs to be answered here is whether a document should be moved or copied from one cache to another. There are a few document placement schemes proposed in the past such as the Ad hoc Scheme, Expiration age Scheme.

This project proposes a modified version of the Expiration age scheme. The modified expiration age scheme would have an additional frequency factor. In order to measure the success of the proposed modified approach, its performance i.e. mean response time (Average Latency) and document hit ratio has been compared against the performances of Ad hoc scheme and Expiration age scheme using a discrete event simulator by using Prof. Alan Kaminsky's Parallel Java Library.

Introduction

1.1 Why Caching

P2P popularity has increased over the years and is widely used for sharing of data, streaming files, transferring files etc. Most of the P2P systems that exist today are built over the Internet which connects the entire world [2]. There are two major concerns associated with such implementations of a P2P system, which are:

- Availability and transfer of data quickly to a user

- Reduce network backbone load thus reducing cost of Internet Service Providers (ISP)

How does this affect the ISP?

There are different ISP providers that offer internet service to users. In order to allow users to get access to data spanning across different parts of the globe, it is necessary for them to collaborate. An ISP would have to pay another ISP to use their resources for offering various multimedia services to the users. Since majority of the internet traffic constitutes of P2P traffic, it puts a heavy load on the backbone links and interferes with non P2P users over the internet [5].

A possible solution to alleviate the P2P traffic over network backbone and costs is to place caches in the network. So, how this would help is that if there is a P2P user requesting for some data, the peer would check for the object in its local cache first instead of going to the public links and utilizing the public bandwidth, and if it is found it will be returned to the user, otherwise it will be sent over the network to search for it in the other autonomous systems. In this way P2P traffic can be subdued and restricted from getting into the public networks. The way a cache is implemented will also impact the way it improves the efficiency of a P2P system.

1.2 Independent caching v/s Cooperative caching

In Independent Caching, a request generated from a peer is first checked for in its local cache. If the cache can serve the request it returns back the results. If the cache cannot serve the request, it is forwarded to a server which is a gateway to the internet. The server then serves the request.

However, the way Cooperative caching works is described as follows:

1. Whenever client makes a request to a peer for a data, the request is first checked in its local cache. Local cache is basically the cache with which the peer is associated with. A Cache can be the local cache of several peers or one peer.
2. If the requested data is found in the local cache, the data is returned back to the client who requested for it.
3. If the request is not found in its local cache, it is checked in other caches within the system.
4. If the requested data is found in other caches of the cache group, the requested data is returned back to the user.
5. If the requested data is not found in the caches of the entire cache group, then the request is forwarded to the server. Server is a term used here to represent the rest of the internet.
6. The data is fetched from the server and served to the client. Also, a copy of the requested data is stored in the local cache of the peer where the request originated.

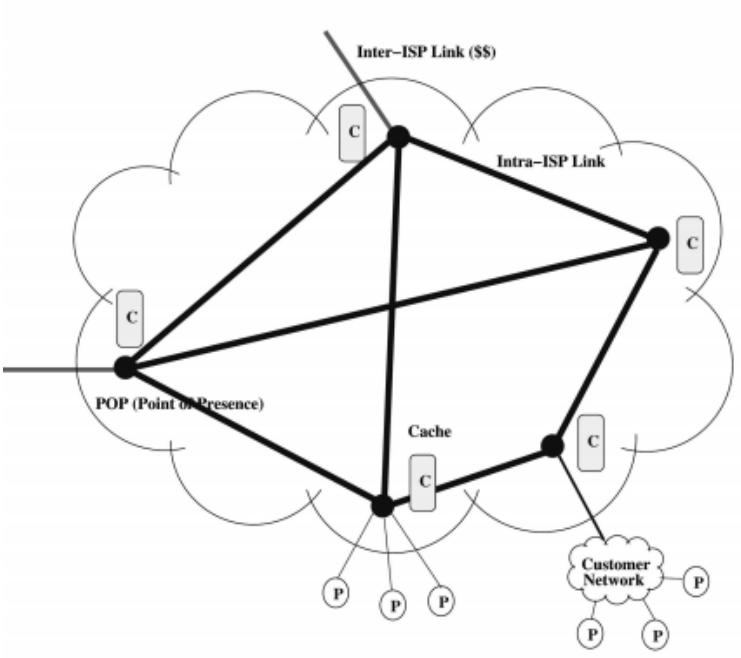

Figure 1 – Cooperation among caches within an AS

The model described above shows cooperation among caches within a large ISP. There can be multiple points of presences (POP) in the network. ISPs provide internet access to their

customers at POPs. The caches are deployed at Points of Presence to save P2P traffic from going on to the inter ISP links.

The intra ISP links between the caches are high bandwidth links whereas inter IS links also known as the backbone links are low bandwidth links. In case of independent caching, since the caches of an Autonomous system do not cooperate with each other, the intra ISP links are not present. The primary goal of Cooperative caching to reduce the amount of traffic sent on the backbone links.

There are many aspects or dimensions to be considered in Cooperative caching such as cache replication schemes, cache replacement policies, cache placement schemes etc. The primary focus of this project is on cache document placement schemes.

1.3 Benefits of Cooperative Caching

Cooperative caching is a vastly researched topic. Cooperative caching has many advantages over Independent caching.

- In Cooperative caching, since other caches in the system are checked or searched first before contacting the server, the chances of finding the requested data or file is higher than Independent caching.
- Thus, the number of cache hits observed in Cooperative caching is more than the number of cache hits observed in Independent caching.
- Also, the mean response time for requests in Cooperative caching is less than that of Independent caching. The reason is the same since the chances of finding the requested data is more in Cooperative caching. Thus, the requests are served quickly leading to reduced response time.
- In ISP's, installing caches which cooperate with each other can reduce the overall costs for the ISPs to a great extent. ISP's are connected to the internet using slow backbone links. Cooperative caching reduces the amount of traffic sent on the backbone links thus reducing the cost.
- Cooperative caching gives an illusion to the client of a much larger cache.

2. Problem Statement

In Cooperative caching, there is a high change that a requested data which is not found in the local cache is found in any other cache of the cache group. In such a situation, an important question which is whether the requested data is to be copied or not to the local cache needs to be answered. There could be two scenarios here. The first case is that the requested data is copied every time to the cache where the request originated. However, this will lead to uncontrolled replication of data items. The second case is that the requested data is never copied to the requesting cache. Then there is a possibility that the data will be evicted by cache replacement policies and the cache group would not contain that data item anymore. This problem is the document placement problem in cooperative caching and there are a few schemes which deal with this problem and have been evaluated in this project.

3. Literature Review

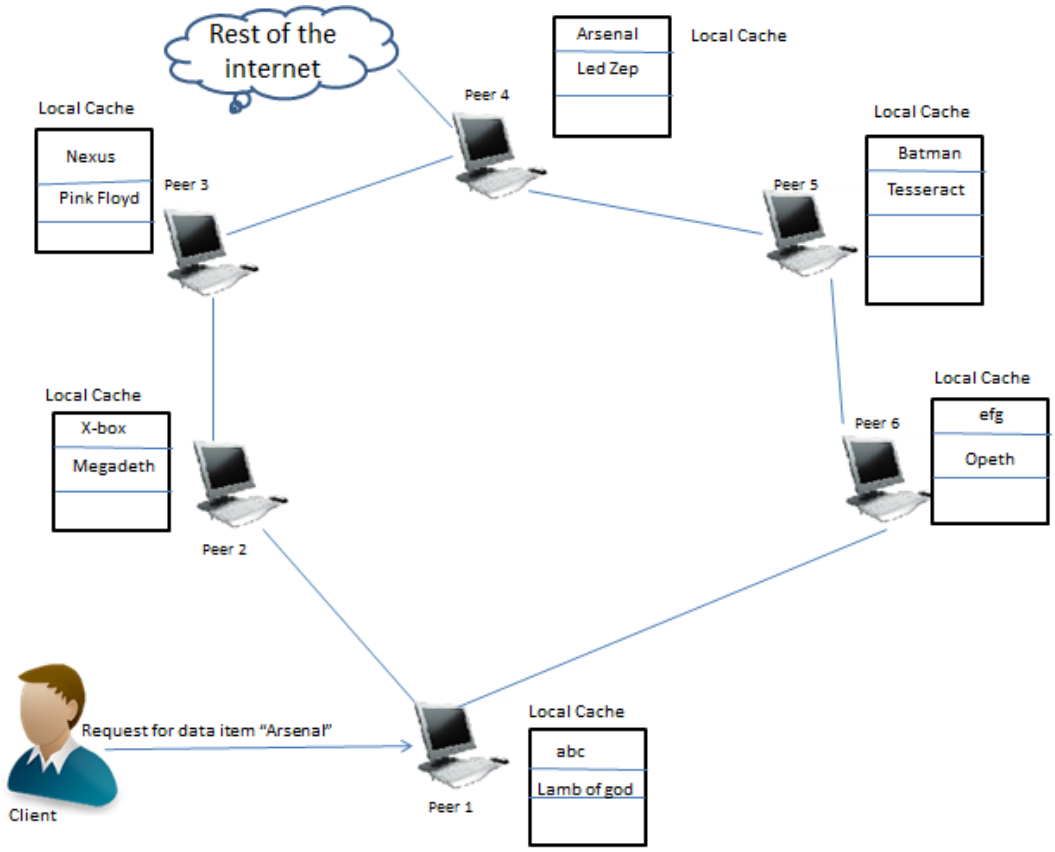

Figure 2 – Example of a Cache group consisting of six caches

The figure above depicts an example of a cache group. There are six peers in the cache group and each peer has its own local cache. In a Peer-to-Peer system, every peer can have its own local cache or few peers can be associated with a single cache. The cache group is connected to the rest of the internet or server through a low bandwidth link also called backbone link.

A client contacts Peer 1 and requests for the data item "Arsenal". Considering this scenario, we shall see how document placement schemes work.

3.1 Ad Hoc Scheme

In conventional caching techniques, when a data item is not found in the cache, the cache requests for the document from another cache belonging to the same cache group or from the external server (rest of the internet). The data item is fetched and stored at the cache which

requested for the data item. Thus, each cache is treated as an independent entity rather than a global entity when making document placement decision. This leads to replication of documents across the cache group. The worst case scenario would be when every cache has a copy of all the documents in the group. Such a scheme is known as Ad hoc Scheme and it is not very efficient in the sense that there is a lack of unique documents in the group due to uncontrolled replication. Given below is an example of how Ad hoc scheme works.

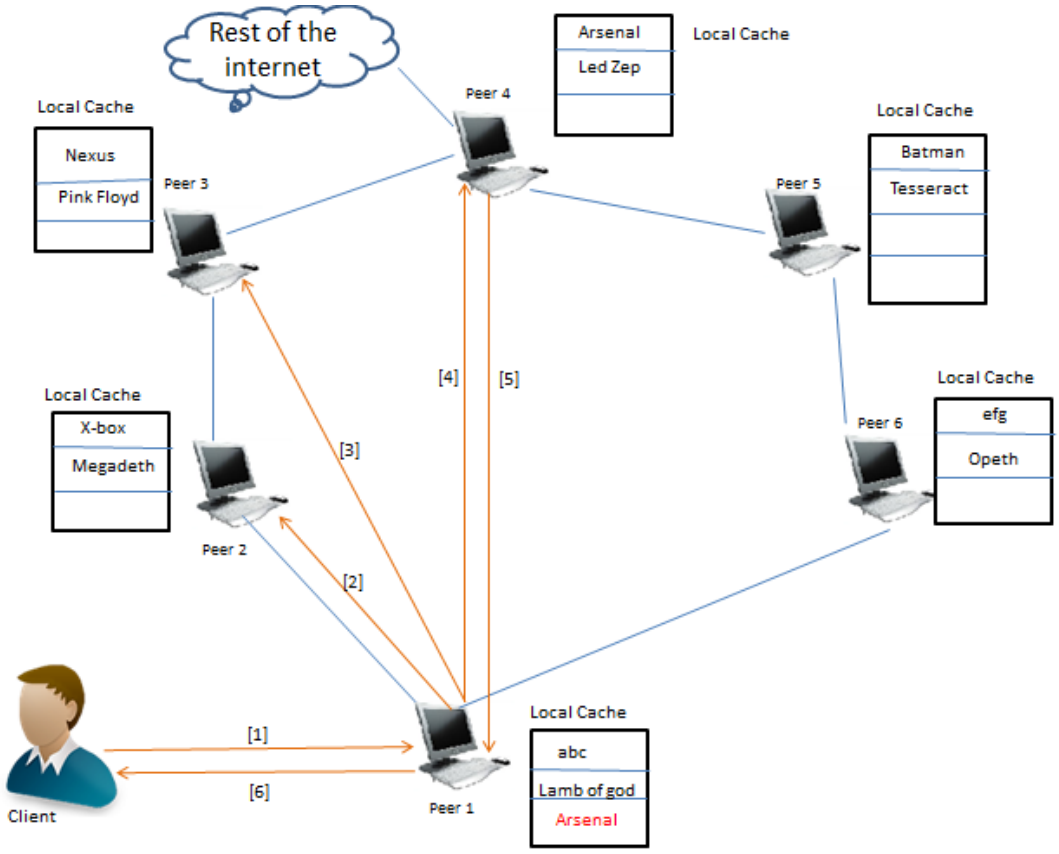

Figure 3 – Example of Ad hoc Scheme in a cache group

1. Client contacts Peer 1 and requests for the data item “Arsenal”. Peer 1 checks its own local cache for “Arsenal” data item. Initially peer 1’s local cache contains “abc” and “Lamb of god” data items as shown in figure 1. Thus, Peer 1 does not find the requested data item in its local cache.
2. Since, the caches of the cache group cooperate with each other, Peer 1 contacts peer 2 for the data item. Peer 2 also does not have the requested data item.
3. Next, Peer 1 contacts peer 3 for the data item. Peer 3 also does not have the data item.
4. Next, peer 1 contacts peer 4 for the data item. Peer 4 has the requested data item in its local cache.

5. Peer 4 replies back to peer 1 with the requested data item. Now, what happens in Ad hoc scheme is, every time the data item is found in some other cache, the cache copies the requested data item in its own cache. Thus, as seen in the figure above, peer 1 copy the data item “Arsenal” in its own local cache (shown in red color). Thus, we see that now Peer 1 and Peer 4 both have copies of “Arsenal”.
6. Peer 1 replies back to the client with the data item.

3.2 Expiration Age Scheme

The Expiration Age scheme leads to better hit rates and response times than the Ad hoc Scheme by limiting the no. of replicated documents in the group. It reduces unnecessary replicated documents but also ensures that a copy of that document is present at some cache in the group where it is certain to be stored for the longest time. This is basically done using a concept known as Expiration age.

The time for which a document stays in a cache is defined as the Lifetime of that document. The Average document lifetime of the cache is defined as the average of the lifetimes of the documents that were evicted from the cache in some interval of time. Thus, the Average document lifetime of a cache is inversely proportional to the disk space contention at the cache. Higher the disk space contentions at the cache, faster are the documents evicted from the cache.

For LRU replacement policy which has been used in this project, LRU Expiration age of a document is the difference between the time since it was last hit till the time it is removed from the cache. For this, the proxy caches have to maintain the time of the last hit for all the documents.

The algorithm for the EA scheme makes use of the expiration age of the cache. Each cache has to send its expiration age to other caches in the group in HTTP requests and response messages. For this, whenever a cache experiences a miss, it sends an ICP request message to other caches and includes its expiration age in the message. If any other cache in the group has the requested document, it replies back to the requester in an ICP response message and attaches its expiration age in the message. Now, the requestor compares its expiration age with the responder’s expiration age. If the requestor cache’s EA is greater than the responder cache’s EA, then it stores a copy of the document in its cache else it does not store. Similarly, at the responder’s side, if its EA is greater than the EA of the requestor cache, it promotes the requested document to the HEAD of the LRU list in other words, gives it a new lease on life.

The objective is a document is not cached at the requester's cache if the EA is less than that of the responder's cache because the document is more likely to stay longer at the responder's cache. Even if a copy of the document is made at the requester's cache, it is most likely that the copy is going to be removed before the copy of the document at the responder's cache.

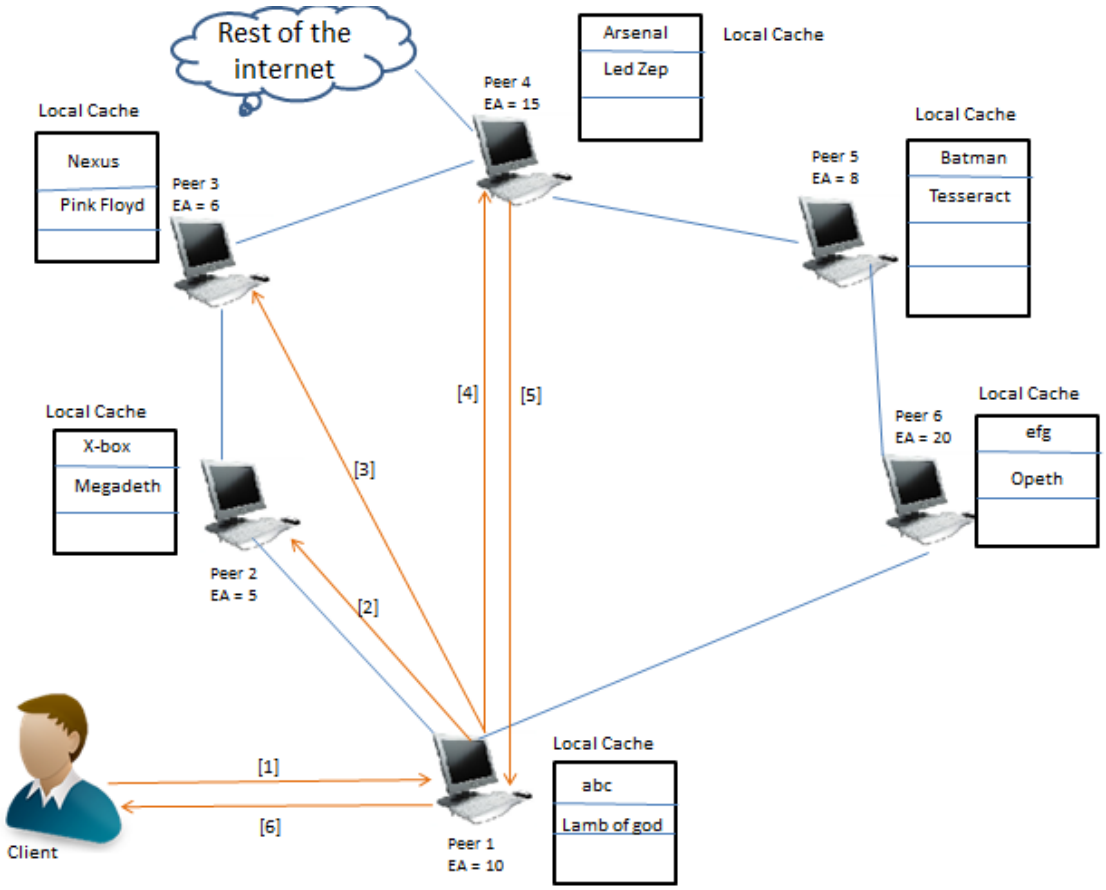

Figure 4 – Example of Expiration Age Scheme in a cache group

Consider the scenario as shown above

1. Client contacts Peer 1 and requests for the data item "Arsenal". Peer 1 checks its own local cache for "Arsenal" data item. Initially peer 1's local cache contains "abc" and "Lamb of god" data items as shown in figure 1. Thus, Peer 1 does not find the requested data item in its local cache.
2. Since, the caches of the cache group cooperate with each other, Peer 1 contacts peer 2 for the data item. Peer 2 also does not have the requested data item.
3. Next, Peer 1 contacts peer 3 for the data item. Peer 3 also does not have the data item.
4. Next, peer 1 contacts peer 4 for the data item. Peer 4 has the requested data item in its local cache.

5. Peer 4 replies back to peer 1 with the requested data item. Now, what happens in Expiration age scheme is, the responder cache i.e. Peer 4's local cache has an expiration age of 15. The requestor cache i.e. Peer 1's local cache has an expiration age of peer 10. Since, peer 4's cache's expiration age is greater than peer 1 cache's expiration; the data item will not be copied in Peer 1's cache since the data item is likely to stay for a longer time in the local cache of peer 4.
6. Peer 1 replies back to the client with the data item.

4. Proposed approach: Modified Expiration age scheme

The modified expiration age scheme which I have implemented and evaluated in this project contains an additional factor of frequency of requests apart from the expiration age parameter which was explained above.

The idea behind the Modified Expiration age approach is that if the number of requests at a cache for an object is greater than a specified threshold, it would be more beneficial to copy the object to that cache, regardless of the expiration age. The threshold will basically be the total number of requests divided by the server data size. Hence, there will be two factors governing the document placement scheme i.e. the frequency of requests for objects and the expiration age of the cache.

Consider the same scenario as explained in the Expiration age scheme. The scenario explained above showed that the data item "Arsenal" was not copied to the local cache of Peer 1 since the expiration age of Peer 1 was less than the expiration age of Peer 4. This indicated that the data item is more likely to stay for a longer time at Peer 4 than Peer 1. However, consider the case where Peer 1 receives a large number of client requests for the same data item "Arsenal". In this case, since the expiration age of Peer 1 is less than that of Peer 4 (which is the holder of the requested data item), the data item will never be copied to the local cache of Peer 1. Thus, every time Peer 1 receives a request for data item "Arsenal", it will follow the same procedure every time i.e. first check its local cache, then it will check the local caches of other peers until it finds the data item in the local cache of Peer 4. This will induce some delay in the response time. Thus, it would be advantageous to copy the data item from the responder cache(in our example, Peer 4) to the requestor cache(in our example, Peer 1) irrespective of the expiration age if the number of requests for that data item is greater than a specified threshold thus eliminating the small delay in response time.

Why is this scheme better than the expiration age scheme?

Though the Expiration Age scheme performs better than the Ad hoc Scheme, it does fail to take into account the frequency factor. Each peer spends some amount of time in processing the request and to lookup for the data item which is being requested. This can be termed as the cache processing time. Even though the processing and lookup time is small, it will make a significant difference if the number of requests for that data item is above a specific threshold. In that case, it is beneficial to copy the data item in the local cache of the requestor irrespective of the expiration age.

5. Simulation Methodology

5.a Description of Simulation Environment

Design of Simulation

The following classes have been implemented for this project:

a. **Peers**:
   Peers are responsible for generating requests within the system. Every peer has an associated unique ID with it. Whenever a peer enters the system, it gets associated with a cache. Every request generated from that peer is searched for in its local cache first.

b. **Caches:**
   Caches are responsible for serving requests within the system. Every peer in the system gets associated with a single cache. A cache consists of a cache queue which holds all incoming requests and the cache itself which is a repository of data present. If the requested data is found within the local cache it is returned to the peer thereby eliminating the need to go to other caches in the group or to the server.
   In cooperative caching, each request is searched for in a cache group. A cache group is the list of all the caches in a system. This helps in getting better response time. A cache also contains a Hashmap which stores the request name and its associated frequency. This is used for the Modified Expiration Age scheme.

   This class implements the logic for expiration age. It maintains the time when the data items entered the cache and also the time of their eviction. Thus, periodically it computes the expiration age parameter of the cache. This is done by computing the mean of the times of the documents which were evicted from the cache during a

specific time interval. This expiration age is then used for document placement purposes.

This class also implements the Least Recently Used (LRU) cache replacement algorithm which is used when a cache is full and needs to evict objects from the cache.

c. **Requests:** Requests are generated within the system based on the application scenarios. Every request has an associated ID and some data which is the request itself. The rate at which requests arrive in the system and the number of requests is constant and is taken from the input file. Requests are either served by the server or by the caches in the system. If a request is served by the server, the data requested is then added to the local cache so that any further similar requests can be directly served by the cache instead of going to the server.

d. **Server:** The server is just a gateway to the Internet. Any request that cannot be served from the cache is forwarded to the server which in reality means that the data which is requested is served from the Internet. In this system the server consists of a large set of data. This data set is the set of all requested objects in the system. The server also consists of a request queue which holds all the incoming requests being forwarded to the server. The server processing time is set at the start of the simulation and is taken from the input file. We assume that if a request comes to the server it will be served after a certain time, i.e. the server will always serve incoming requests.

e. **Generator:** This class is responsible for generating requests in the system and searching for data in the caches and the server.

f. **DocPlacMain:** This component is responsible for creating the above topology and initializing all of the entities mentioned above. Simulations for both the systems are run by taking in certain user parameters.

Other Classes used: Class Simulation and Class Event from the Parallel Java Library.

Class Event: It is the abstract class for an event for a discrete event simulation.

Class Simulation: This class provides the simulation object for the discrete event simulation.

Flow of the Simulation

- The only command line argument which the Simulation takes is the filename which contains the various parameters and their values. The various parameters used in the simulation are inter arrival rate, number of requests, number of peers, number of caches, cache processing time, server cache processing time, server disk processing time, data file which contains the data items in the form of strings, seed for random generation, another seedReq for random requests generation, server data size to indicate the amount of data items present in the server and whichScenario which indicates whether the scenario being tested are the base cases or application specific scenario.
- If the value of whichScenario parameter is “baseCases”, the simulation is run for four cases i.e. when all data items are found in local cache, when all data items are found in remote cache, when all data items are found in server cache and when all the data items are found in the server disk.
- There are two for loops in the main class. The outer for loop runs for different sizes of cache group and the inner for loop runs for the three different schemes i.e. Ad hoc scheme, Expiration Age scheme and Modified Expiration Age scheme.
- Inside the inner for loop, the Server Class object is setup. Also, the Caches class objects are also initialized depending on the number of caches taken into consideration.
- After the cache objects are setup, Peers are associated randomly to caches.
- The Simulation class object is also made and each cache object calls the computeExpirationAge() function. The implementation of this function is present in the Caches class and this function runs on a separate thread to calculate the expiration age of the Cache on a periodic basis.
- The computeExpirationAge() function has a scheduler which is scheduled to run every 100 milliseconds. This means that the expiration age of the cache is calculated every 100 milliseconds. I have a ListSeries which stores the time for a data item stayed in the cache when it is being evicted. The expiration age of the cache is the mean of this listSeries. Thus, periodically i.e. every 100 milliseconds, the mean of the ListSeries are calculated and then the ListSeries is cleared for the next periodic computation.
- The main function then creates an object of the Generator Class which is responsible for generating requests. Based on the scenario, the requests are generated. There are two scenarios used in this project. The “sequential” scenario is when a peer accesses data items sequentially i.e. a peer requests for a data item and sequentially accesses the next data items till a specified counter. This application scenario can be viewed as streaming a movie on Netflix or watching a video on YouTube where blocks or data items are accessed sequentially. The next scenario is the “repetitive” scenario where certain

random peers access the same data item. Few peers access the same data item, and then other few peers access a different data item and so on. This application scenario can be viewed as friends accessing or looking up the same photo on Facebook or users looking at the same item on Amazon or Ebay. Also, whatever the scenario, the concept of popular data items is also taken into consideration. Half of the requests are requests for a specified small set of data items which are the "popular" items and the rest of the requests are for the remaining set of data items which are the unpopular items.

- In the Generator Class, requests are generated from peers and the requests are added to the cache queue of the local cache. Local cache is the cache with which the peer from where the request originated is associated.
- As the requests are added to the respective cache queues, a searchCache() method is called.
- This searchCache() method searches the local cache first. If not found, it searches the remote caches in a round robin fashion. A small amount of delay is introduced in between searching the remote caches.
    - If the scheme in consideration is Ad Hoc scheme and if the data item is found in the remote cache, the data item is copied to local cache's cacheDataList (An arrayList which stores the data items. It is the actual cache itself). Also, in the remote cache where the data item was found, the data item is given a new lease of life. This is done by removing the data item from the remote cache's cacheDataList and adding the same data item back to the remote cache's cacheDataList. This adds the data item to the end of the arrayList thus indicating that the data item was recently accessed. If not found in any of the remote caches, it contacts the server. If found in the server cache, then the data item is returned else the data item is obtained from the server disk. A higher amount of delay is introduced when contacting the server since the links between the cache group and the server are low bandwidth links also known as backbone links.
    - If the scheme in consideration is Expiration Age scheme and if the data item is found in the remote cache, the data item is copied only if the expiration age of the local cache is greater than the expiration age of the remote cache. The expiration age of the cache is calculated on a periodic basis separately irrespective of the main flow of the program. If the expiration age of the local cache is less than the expiration age of the remote cache, then the data item isn't copied to the local cache. In this case, the data item is updated in the remote cache's cacheDataList by removing it and re-adding it.
    - If the scheme in consideration is Modified Expiration Age scheme and if the data item is found in the remote cache, the data item is copied only if the expiration age of the local cache is greater than the expiration age of the remote cache or if

the frequency of that data item is more than a threshold. The threshold in my experiment is calculated by dividing the total number of requests in the experiment by the total size of the server. This value is optimal because ideally if all the data items are requested one by one, then the number of times a data item is requested is given exactly by the total number of requests by the server data size. However, in reality, popular data items are requested more often i.e. more than the threshold, thus we want those data items to be considered in the modified expiration age scheme. If the expiration age of the local cache is less than the expiration age of the remote cache, then the data item isn't copied to the local cache. In this case, the data item is updated in the remote cache's cacheDataList by removing it and re-adding it.

- Cache replacement which is used in the project is LRU (Least recently used). For this, whenever a data item is accessed, it is removed from the cache's cacheDataList and added again. Thus, the data item is added to the end of the cacheDataList. During cache replacement, the data item at the top of the cacheDataList is removed since it is the least recently used.
- Also, during cache replacement, the removed data item's end time is noted and the lifetime of that data item is calculated since we know the start time of the data item i.e. the time when the data item was added in the cache. This calculated lifetime of the data item is used in the computation of expiration age of the cache.
- In the Server class, if the data items are not found in the server cache, then they are obtained from the server disk. The data item is then copied to the local cache from where the request originated and also copied to the server cache for future access purposes.
- The various metrics are calculated and graphs of average latency v/s cache group sizes, document hit ratio v/s cache group sizes and disk usage efficiency v/s cache group sizes are plotted. Also, the number of local hits, remote hits, server cache hits and server disk hits are shown.

5.b Evaluation Metrics

In order to evaluate the performance of the proposed scheme, it has been compared against the Ad hoc scheme and Expiration age scheme based on the following parameters:

a. **Disk Usage Efficiency** – It is defined as the percentage of unique items in the total disk space available. This means that if all the data items in the cache group are unique, then the disk usage efficiency of the cache group is said to be 100%. It is calculated by the number of unique documents in the cache group divided by the total number of documents in the cache group.

It should be noted that higher the disk usage efficiency, better will be the system performance.

b. Document Hit Ratio – It is defined as the total number of hits for requests in the cache group by the total number of requests. The higher the document hit ratio, the better will be the performance of the system.

c. **Average Latency** – Response time of a request is defined as the time taken when it is added to the cache's queue till the time it is finished processing. Average Latency can be calculated as the mean of the response times of all the requests in the system. The lower the average latency, the better will be the system performance. The unit of average latency is in milliseconds.

d. **Local Hits, Remote Hits and Server Hits** - Local Hits can be defined as the number of hits encountered in a peer's local cache whereas Remote Hits can be defined as the number of hits encountered in other peer's cache of the cache group. Server Hits can be defined as the number of requests served by the server since it cannot be served by the caches in the cache group.

6. Test Cases and Results

6.a Base Cases

Below are the values of the parameters used to study the base cases:

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	4
Seed	142857
SeedReq	21556
whichScenario	baseCases

Table 1 – Parameters for the test of base cases

6.a.1 All Data Items found in the Local Cache

Following are the results obtained when all the data items were found in the local cache. For this, a request for a data item say D1 is originated from a cache say C1 and a request for another data item say D2 is originated from a different cache say C2.

To get the results of this base case, all the subsequent requests are for data items D1 and D2. Thus, initially these two data items are obtained from the server disk and stored in the local caches. All the subsequent requests for that data item are obtained from the local cache itself.

Below is the output obtained for a cache group size of 1000.

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	9998	0	0	2	99.98%	100.00%	85.526
Expiration	9998	0	0	2	99.98%	100.00%	85.526
Mod Exp	9998	0	0	2	99.98%	100.00%	85.526

Table 2 – Results for the three schemes for base case 1.

The results obtained for other cache group sizes i.e. 1000, 1200, 1500, 1800, 2000 and 2500 are the same.

6.a.2 All Data Items found in the Remote Caches

Following are the results obtained when all the data items were found in the remote caches. For this scenario, a request for a data item say D1 is originated from a cache say C1 and a request for another data item say D2 is originated from a different cache say C2.

To obtain the result of this base case, C1 gets a request for data item D2 whereas C2 gets a request for data item D1. Thus, each data item can be found in the remote cache. Only to get the result for this scenario, a condition was applied that data item found in the remote cache was not copied to the local cache.

Below is the output obtained for cache group size of 1000.

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	0	9998	0	2	99.98%	100.00%	7581.719
Expiration	0	9998	0	2	99.98%	100.00%	7581.719
Mod Exp	0	9998	0	2	99.98%	100.00%	7581.719

Table 3 – Results for the three schemes for base case 2.

The results obtained for other cache group sizes i.e. 1000, 1200, 1500, 1800, 2000 and 2500 are the same.

6.a.3 All Data Items found in the Server Cache

Following are the results obtained when all the data items were found in the server cache. For this scenario, a request for a data item say D1 is originated from a cache say C1 and a request for another data item say D2 is originated from a different cache say C2. These two data items are initially obtained from the server disk. Usually what happens is the two data items are copied to the server cache and to the local caches respectively for future purposes. However, for this scenario, I just copy the data items to the server cache and not to the local caches. Thus, subsequent requests for the two data items will be obtained from the server cache since the server cache is always checked before the server disk.

To obtain the result of this base case, requests are generated for the D1 and D2 data items and these data items are found and obtained from the server cache.

Below is the output obtained for cache group size of 1000. The results obtained for other cache group sizes i.e. 1000, 1200, 1500, 1800, 2000 and 2500 are the same.

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	0	0	9998	2	0.0%	NaN%	60124.385
Expiration	0	0	9998	2	0.0%	NaN%	60124.385
Mod Exp	0	0	9998	2	0.0%	NaN%	60124.385

Table 4 – Results for the three schemes for base case 3

The Document Hit Ratio is 0.0% in this base case since doc hit ratio is the number of requests which were served by the caches divided by the total number of requests. However, as none of the requests were served by the caches of the cache group, the document hit ratio is zero.

Also, the disk usage efficiency in this base case is NaN since disk usage efficiency is the number of unique items in the cache group i.e. the number of unique data items divided by the number of data items in the cache group. However, the number of unique data items and the number of data items are both zero in this case. Hence, the disk usage efficiency is NaN.

6.a.4 All Data Items found in the Server Disk

Following are the results obtained when all the data items were found in the server disk. To obtain the results of this case, all the requests for data items were fetched from the server disk. This happened on the condition that none of the data items were copied to the local cache or the server cache. Thus, all the data items were found from the server disk.

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	0	0	0	10000	0.0%	NaN%	70122.385
Expiration	0	0	0	10000	0.0%	NaN%	70122.385

Mod Exp	0	0	0	10000	0.0%	NaN%	70122.385

Table 5 – Results for the three schemes for base case 4

Similar to the previous base case, the document hit ratio and the disk usage efficiency is 0.0% and NaN% respectively.

Below is the graph of the average latency v/s the cache group sizes for the base cases. As it can be seen, the average latency for the data items found in the local cache only is the least whereas the average latency for the data items found in the server disk only is the highest.

The graphs for document hit ratio v/s cache group sizes and disk usage efficiency v/s cache group sizes were not plotted for the base cases since it is 0.0 and NaN% respectively for two base cases i.e. all data items found in server cache and server disk.

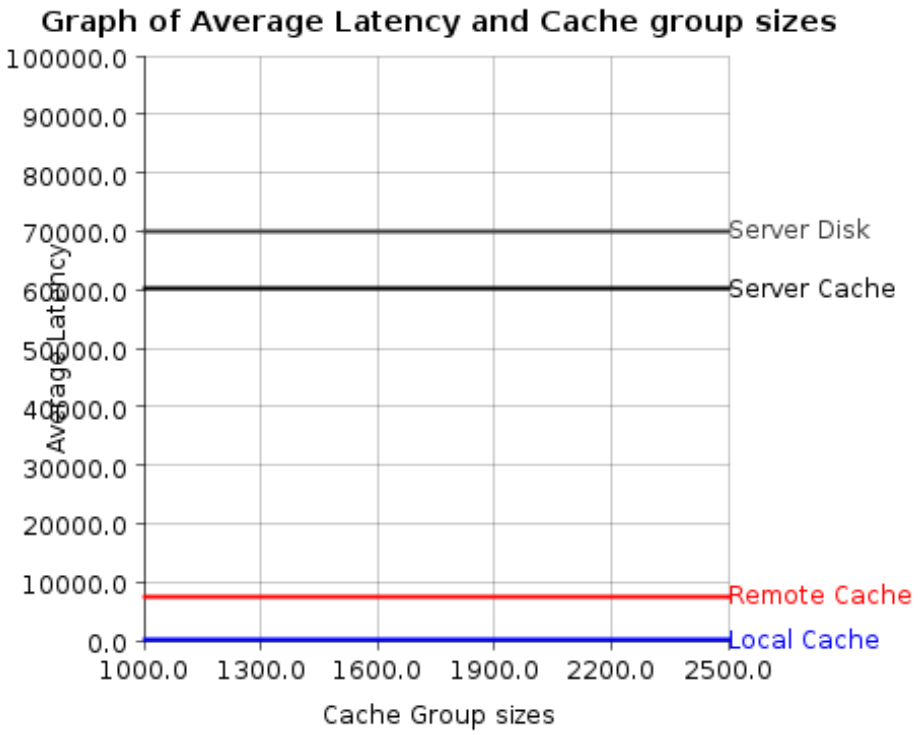

Figure 5 - Graph of Average Latency v/s cache group sizes for the base cases

6.b Vary the Inter arrival Rate of requests

By varying the inter arrival rate, we can observe a difference in the average latency for all the three schemes. This is because, if the inter arrival rate is high that means that the requests come in the system slowly i.e. low traffic. Thus, the caches and the server process the requests

and are idle most of the time. In other words, the cache queues or the server queue does not get filled up fast. Hence, we get low average latency. However, if the inter arrival rate is low as compared to the cache and server processing time, the requests come in the system at a very high rate thus filling up the cache queues and server queue quickly. This leads to an increase in the average latency since the response time for a request is started when it enters the queue and since the requests come in fast, the requests stay in the queue for a longer time.

Below is the first experiment in varying inter-arrival rate. The inter-arrival rate in this experiment is between the cache processing time and the server processing time.

interArrivalRate	**3**
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	4
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 6 – Parameter for test of varying inter arrival rate test 1

The results obtained are as follows.

1000 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	2624	3510	539	3327	61.34%	73.3%	4241.747
Expiration	1926	5193	69	2812	71.19%	87.3%	2534.610
Mod Exp	1939	5196	75	2790	71.35%	84.6%	2485.316

1200 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	2849	3630	499	3022	64.79%	73.08%	3532.108
Expiration	1995	5279	81	2645	72.74%	83.17%	2161.282
Mod Exp	1995	5326	76	2603	73.21%	79.25%	2071.497

1500 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	3679	3333	331	2657	70.12%	61.53%	2618.635
Expiration	2081	5511	82	2326	75.92%	79.53%	1538.534
Mod Exp	1981	5559	51	2409	75.4%	76.0%	**1655.291**

1800 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	3850	3533	270	2347	73.83%	54.83%	1878.848
Expiration	2148	5638	58	2156	77.86%	70.33%	1249.764
Mod Exp	2096	5719	90	2095	78.15%	68.44%	1208.103

2000 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	3950	3514	334	2202	74.64%	60.1%	1706.913
Expiration	2263	5603	19	2115	78.66%	72.75%	1165.765
Mod Exp	2165	5691	49	2095	78.56%	70.95%	1165.140

2500 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	4308	3381	222	2089	76.89%	50.64%	1355.211
Expiration	2262	5692	0	2046	79.54%	82.97%	1063.420
Mod Exp	2262	5692	0	2046	79.54%	84.07%	1062.548

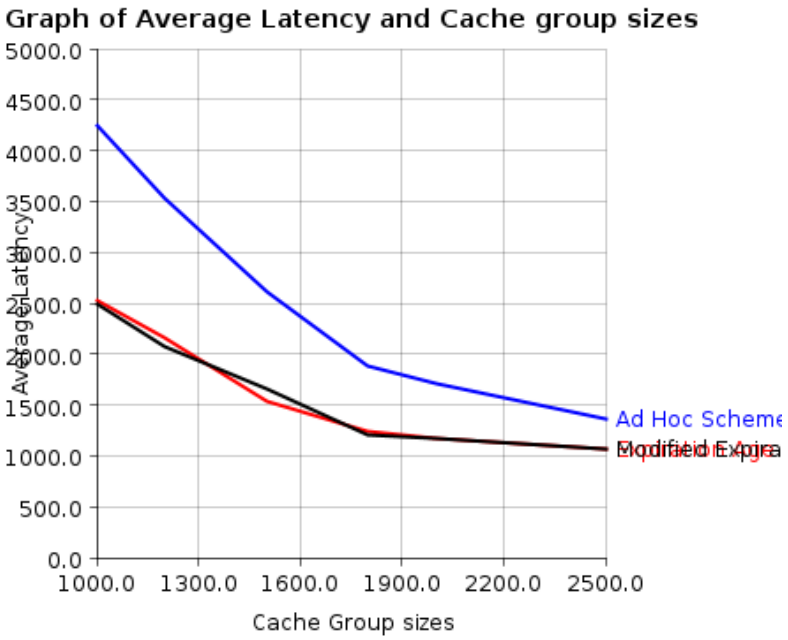

Figure 6 – Graph of average latency v/s cache group sizes for varying inter arrival rate test 1

In the average latency v/s cache group sizes graph, the average latency of modified expiration age scheme is less than that of expiration age scheme in most of the cases. In few cases, the average latency is slightly higher since the number of hits was less. However, the average

latency of modified expiration age and expiration age scheme was lower in all cache group sizes than the ad hoc scheme.

Below is the graph of document hit ratio v/s cache group sizes. The document hit ratio of modified expiration age is slightly higher than expiration age scheme for most of the cases since the number of hits encountered in local cache as well as remote caches is more.

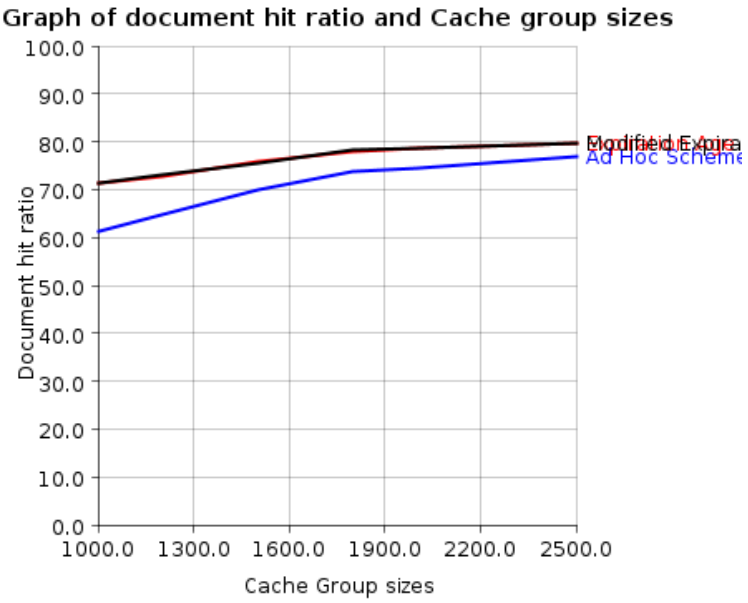

Figure 7 – Graph of doc hit ratio v/s cache group sizes for varying inter arrival rate test 1

Also, disk usage efficiency of modified expiration age is less than the expiration age since the number of unique items in the case of expiration age is more. This is because; we tend to copy more data items to the local cache in modified expiration age scheme because of the frequency factor. Thus, the result is as expected.

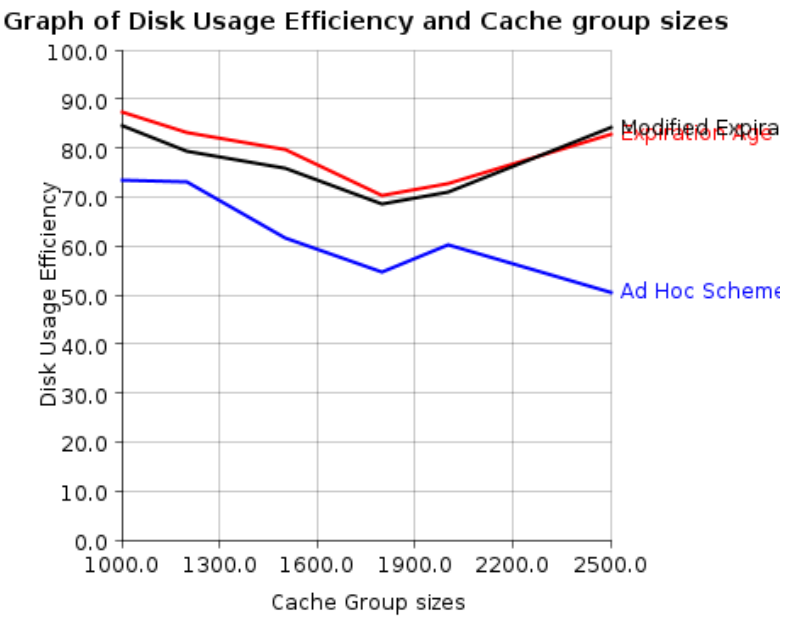

Figure 8 – Graph of disk usage efficiency v/s cache group sizes for varying inter arrival rate test 1

Below is the second expiration in the varying inter-arrival rate. Here, the inter arrival rate is equal to the cache processing time i.e. less than the previous case, thus we expect the average latency to increase as compared to the first experiment.

interArrivalRate	**1**
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	4
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 7 – Parameter for test of varying inter arrival rate test 2

The results obtained are as follows.

1000 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	2642	3510	540	3308	61.52%	74.7%	8686.288
Expiration	1809	4926	111	3154	67.35%	83.9%	6192.675
Mod Exp	1835	4814	134	3217	66.49%	81.8%	6535.012

1200 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	2905	3543	482	3070	64.48%	72.17%	7235.795
Expiration	1998	5184	77	2741	71.82%	82.83%	4540.010
Mod Exp	2035	5227	84	2654	72.62%	84.58%	4280.615

1500 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	3699	3304	311	2686	70.03%	64.47%	4985.174
Expiration	2122	5410	128	2340	75.32%	83.53%	3468.415
Mod Exp	2083	5519	96	2302	76.02%	84.2%	3279.906

1800 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	3842	3470	317	2371	73.12%	54.83%	3911.732
Expiration	2116	5631	132	2121	77.47%	73.67%	2893.196
Mod Exp	2137	5629	139	2095	77.66%	70.71%	2847.613

2000 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	4028	3431	347	2194	74.59%	60.2%	3468.575
Expiration	2092	5731	105	2072	78.23%	66.0%	2720.492
Mod Exp	2105	5758	65	2072	78.63%	66.15%	2626.340

2500 in the cache group

	Local Hits	Remote Hits	SvCache Hits	SvDisk Hits	DocHitRatio	DiskUsageEff	Avg. Latency
Ad hoc	4408	3259	249	2089	76.62%	50.92%	2916.426
Expiration	2312	5564	78	2046	78.76%	73.83%	2603.634
Mod Exp	2350	5561	43	2046	79.11%	72.82%	2522.802

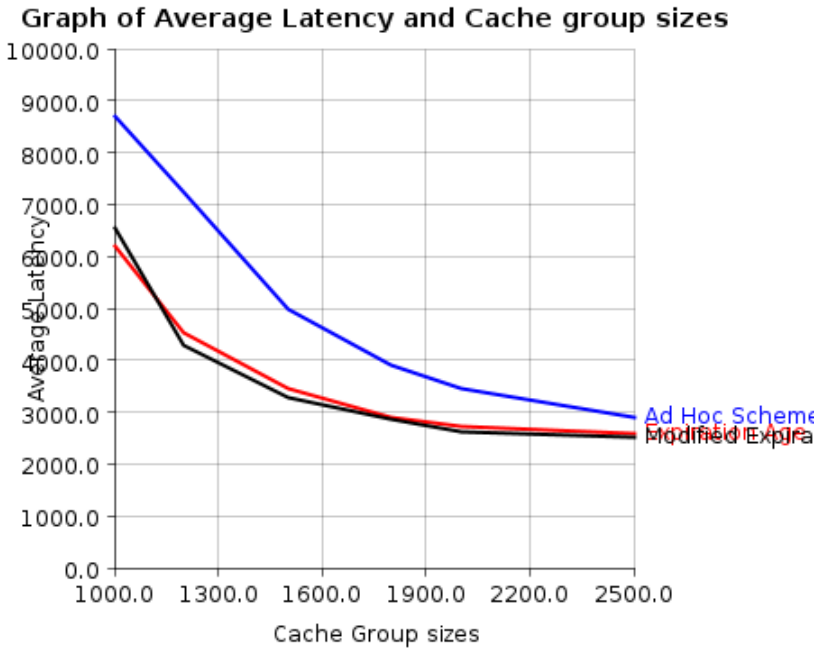

Figure 9 – Graph of average latency v/s cache group sizes for varying inter arrival rate test 2

As it can be seen, the average latency of modified expiration age is less than the expiration age scheme for all cache group sizes except one. Also, the average latency for this experiment i.e. low inter arrival rate is more as compared to the average latency in the previous experiment i.e. high inter arrival rate.

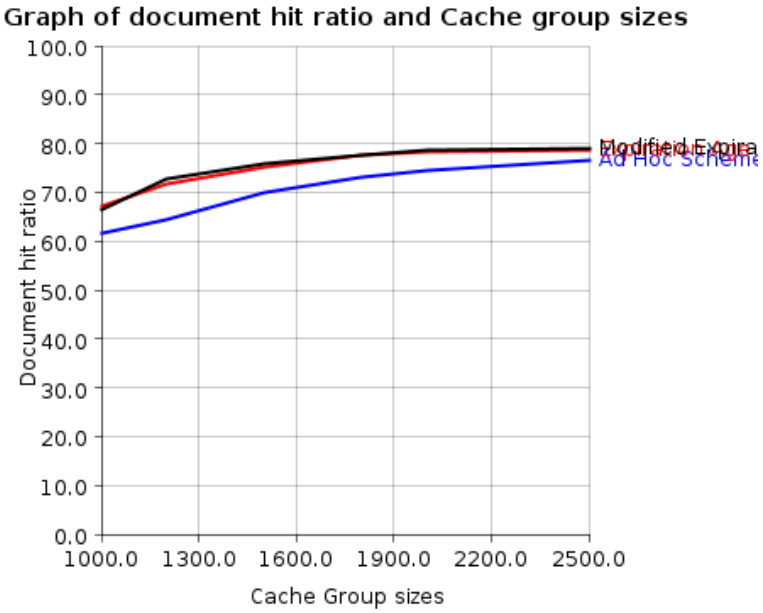

Figure 10 – Graph of doc hit ratio v/s cache group sizes for varying inter arrival rate test 2

Not much difference was observed in the document hit ratio graphs of the two experiments because the hits encountered in both the experiments were approximately similar.

6.c Vary the number of caches in the cache group

The next test case is varying the number of caches in the cache group. As the number of caches in the system increases, the average latency for all the three schemes will increase. This is because the chances of finding the data item in the local cache decreases thus resulting in an increase in the average latency encountered.

Below are the parameters for cache group consisting of 2 caches.

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	**2**
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 8 – Parameter for test of varying no. of caches test 1

I have shown the average latencies for all the cases since it is the metric in consideration here. There isn't much difference in the document hit ratio's and disk usage efficiency since the number of total hits is approximately the same. The difference would be in average latency since as the number of caches increases, there are more caches to search in collaborative caching thus increasing overall average latency.

Statistics for average latency are given below:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	3832.423	2856.032	2311.152	1966.658	1609.882	1404.184
Expiration	2935.678	2552.753	1981.800	1542.857	1453.370	1468.598
Mod Exp	2961.422	2387.380	1993.333	1509.875	1442.154	1458.722

Figure 6 – Graph of average latency v/s cache group sizes for varying inter arrival rate test 1

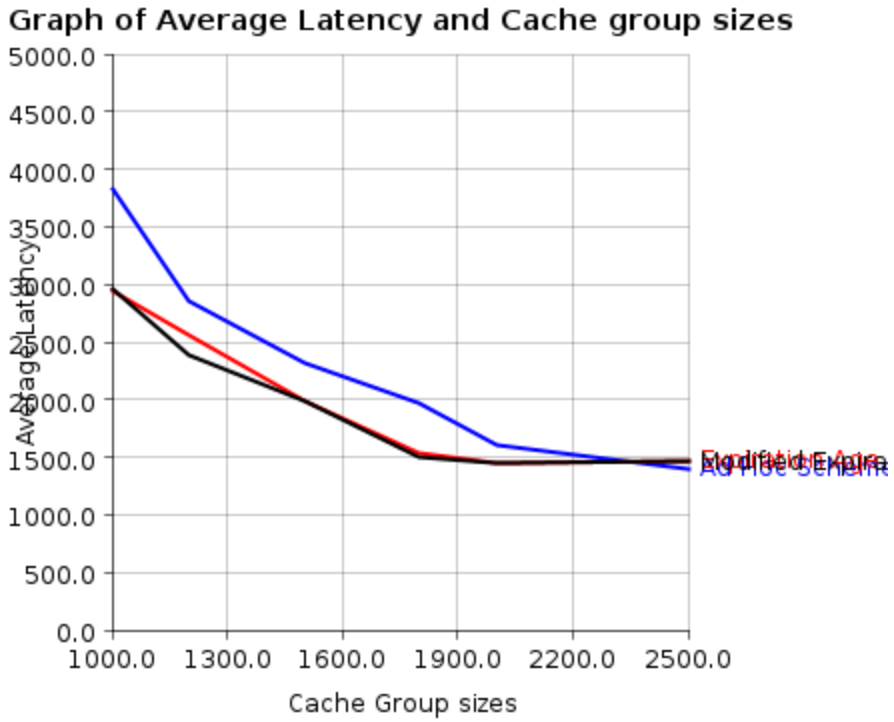

Figure 11 – Graph of average latency v/s cache group sizes for varying no. of caches test 1

Below are the parameters for cache group consisting of 4 caches.

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	**4**
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 9 – Parameter for test of varying no. of caches test 2

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	8192.259	6125.572	4594.949	3546.530	3105.214	2375.499
Expiration	4566.954	4176.081	3435.127	2557.735	2245.135	2147.497
Mod Exp	5164.704	3941.455	3289.382	2576.980	2224.251	2138.980

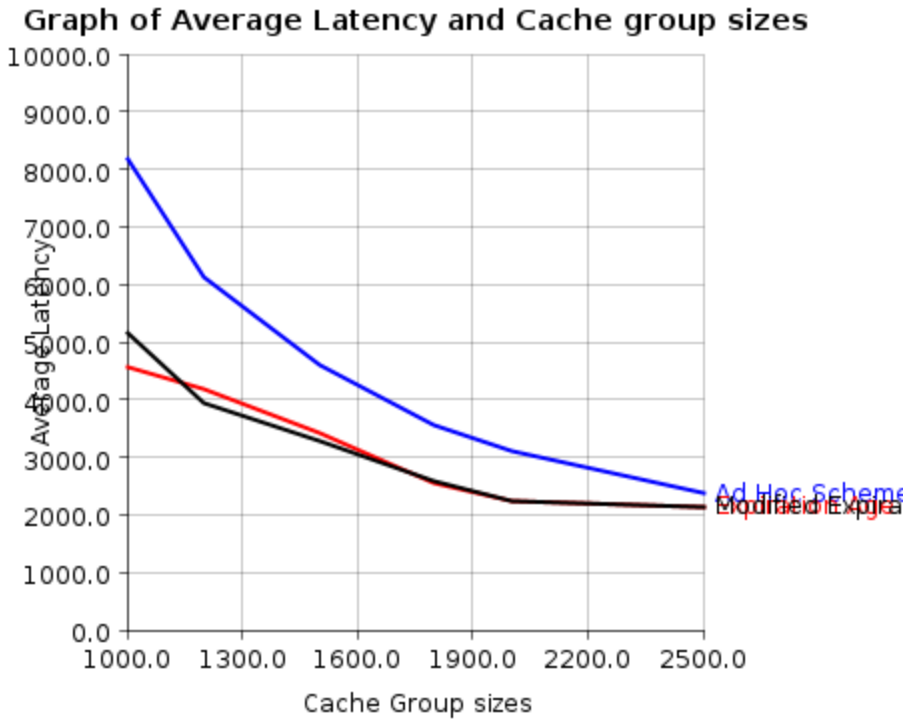

Figure 12 – Graph of average latency v/s cache group sizes for varying no. of caches test 2

Below are the parameters for cache group consisting of 8 caches.

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	**8**
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 10 – Parameter for test of varying no. of caches test 3

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	15059.605	11284.704	8427.418	6620.984	5808.405	4607.534
Expiration	8682.959	7657.317	5578.3299	4621.828	4186.298	3619.717
Mod Exp	8967.070	7194.376	5569.373	4452.400	4098.470	3597.960

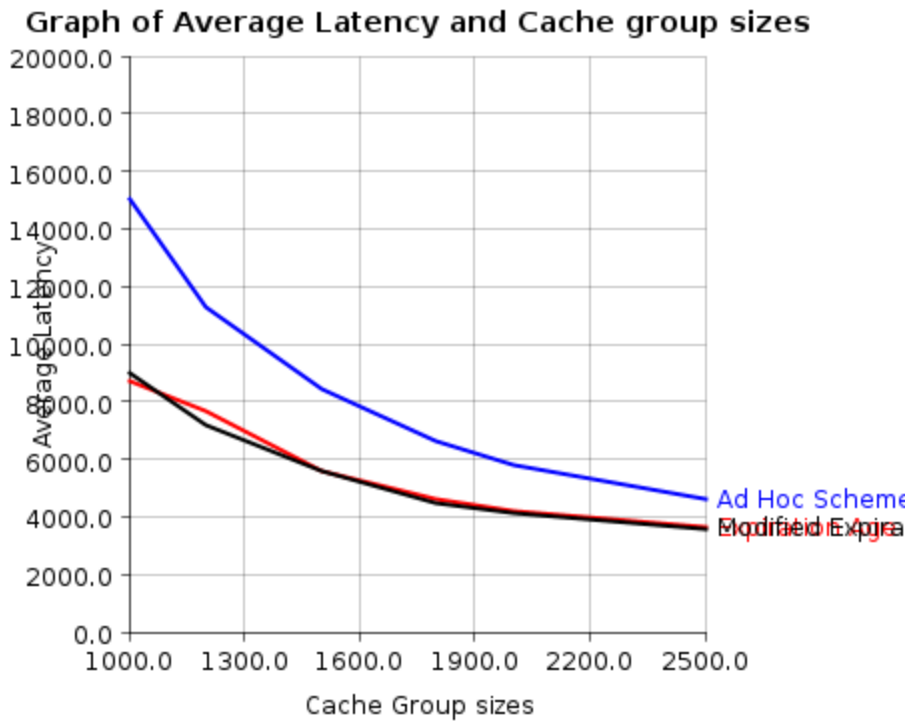

Figure 13 – Graph of average latency v/s cache group sizes for varying no. of caches test 3

Thus, as it can be observed from the results, the average latencies in case of 8 caches in cache group are the highest in all the three schemes as compared to 2 caches and 4 caches in the cache group. Thus, the results obtained were as expected.

6.d Vary the number of requests in the cache group

The next test case is varying the number of requests in the system. The number of requests has a direct relation to the average latency. Also, the number of requests needs to be large as compared to the size of the cache group. This is because, expiration age of a cache is calculated when data items are evicted from the cache. For the eviction to take place, the number of requests has to be large as compared to the size of the cache group. Also, higher number of requests is needed because of the frequency factor which is required for Modified Expiration Age Scheme.

If the number of requests is small as compared to the cache group size, average latency shows weird values since the caches don't get filled up due to which expiration age of a cache is never calculated. Also, the disk usage efficiency increases abruptly since the no. of requests are less i.e. requests for more unique items are less.

Consider this case where the number of requests is very much higher than the cache group sizes. The results were obtained from the test case above i.e. from varying number of caches test experiment.

interArrivalRate	1
noOfRequests	**10000**
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	4
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 11 – Parameter for test of varying no. of requests test 1

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	8192.259	6125.572	4594.949	3546.530	3105.214	2375.499
Expiration	4566.954	4176.081	3435.127	2557.735	2245.135	2147.497
Mod Exp	5164.704	3941.455	3289.382	2576.980	2224.251	2138.980

Consider the case below where the number of requests is less i.e. 4000 as compared to the previous case where the number of requests is significantly larger than the cache group sizes.

interArrivalRate	1
noOfRequests	**4000**
cacheProcTime	1
serverCacheProcTime	6
serverDiskProcTime	8
Filename	Testdata
noOfPeers	100
noOfCaches	4
Seed	142857
SeedReq	21556
whichScenario	Sequential

Table 12 – Parameter for test of varying no. of requests test 2

Statistic of average latency is given below.

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	4592.910	4240.025	3834.154	3737.493	3737.493	3737.493
Expiration	4588.358	3887.970	3775.133	3775.890	3775.890	3775.890
Mod Exp	4395.889	4100.693	3775.494	3775.531	3775.531	3775.531

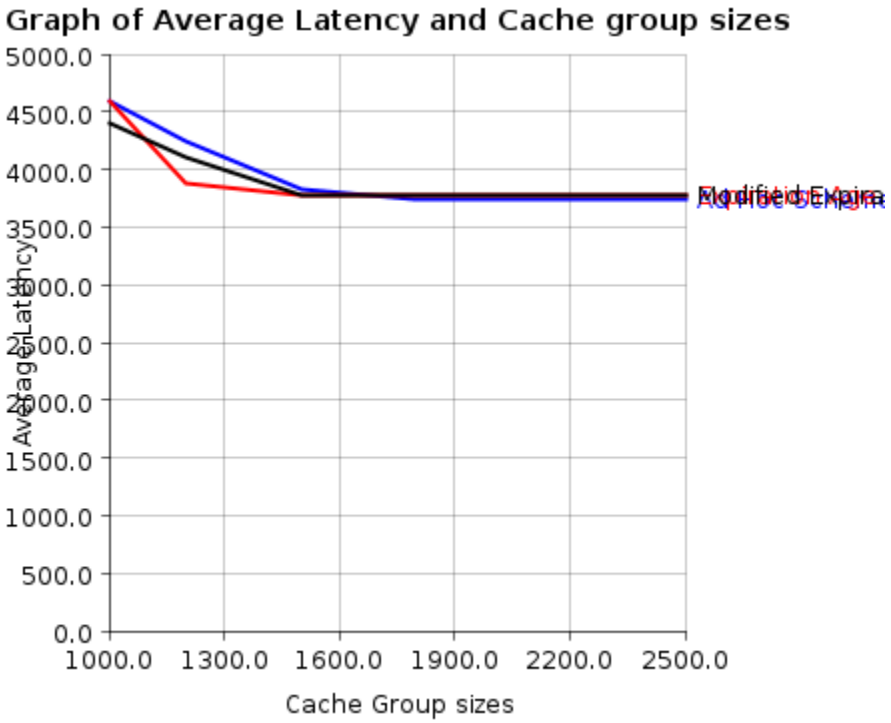

Figure 14 – Graph of average latency v/s cache group sizes for varying no. of requests test 1

As it can be observed above, the latencies of the three schemes are almost the same. This is because of the same reason that the expiration age of the caches is never calculated since they do not get overloaded. Thus, there is no difference between the schemes.

Statistics of disk usage efficiency are given below.

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	69.5%	65.67%	62.67%	65.61%	63.95%	56.84%
Expiration	76.3%	84.42%	93.11%	95.83%	100.0%	100.0%
Mod Exp	80.3%	75.92%	89.48%	95.29%	98.34%	98.34%

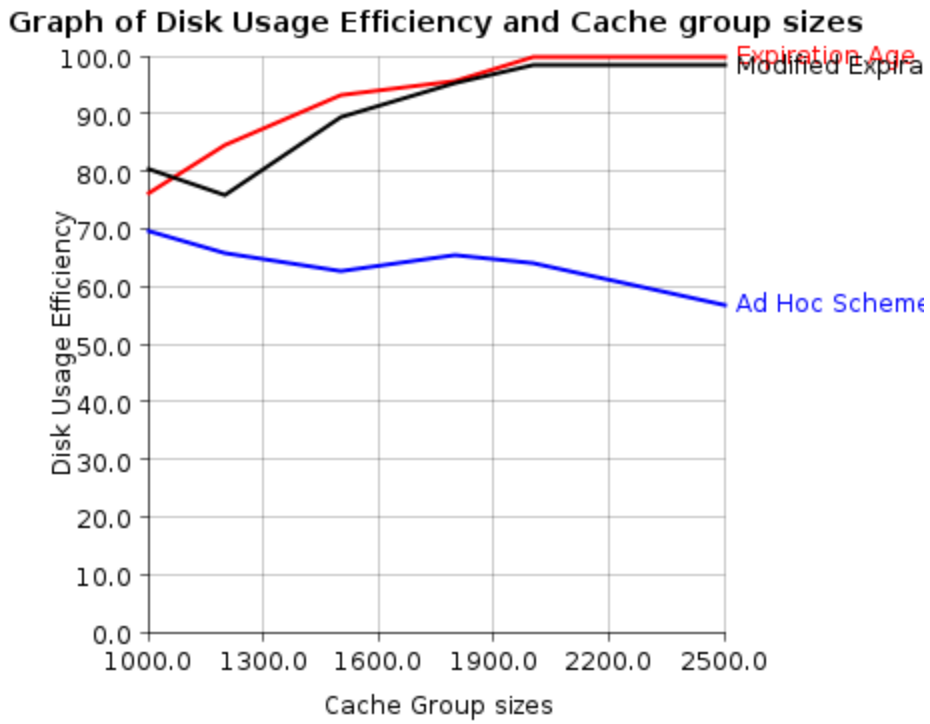

Figure 15 – Graph of disk usage efficiency v/s cache group sizes for varying no. of requests test 1

The reason why the disk usage efficiency of the expiration age and modified expiration age scheme increases and touches 100% is because as the cache group size increases, all the unique data items are already present in the cache. The disk usage efficiency of expiration age scheme goes till 100%. However, the modified expiration age does not go till 100% since in modified expiration age scheme, copies of the data items are created if the frequency is above the threshold. Thus, in that scheme, there are multiple copies present of few data items hence the usage efficiency does not touch 100%. Ad hoc scheme has low disk efficiency since it creates copies of every data item in the local cache which was found in any of the remote caches.

6.e Vary the number of peers in the cache group

The next test case is varying the number of peers in the system. Changing the number of peers in the system would not make much of a difference since it is the number of cache which matters and not the number of peers. Consider a scenario in which there are 4 caches in the system and say 100 peers. Each peer randomly gets associated with a cache. Each request is generated from a peer and it is checked in the local cache. In this scenario, there are around 25 peers associated with each cache on an average. Consider another scenario in which there are 4 caches in the system and say 1000 peers. Similar to the previous scenario, peers randomly get associated with a cache. There are around 250 peers associated with each cache on an average.

Now, the number of peers doesn't matter because requests originate from peers and are first checked in the local cache and then in remote caches. Thus, the number of caches which are present will make a difference and not the number of peers.

Consider the following experiment with 100 peers. Also, "repetitive" is used for the whichScenario parameter. Repetitive scenario is when set of peer's access the same data item and so on.

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	8
serverDiskProcTime	10
Filename	Testdata
noOfPeers	**100**
noOfCaches	8
Seed	142857
SeedReq	21556
whichScenario	Repetitive

Table 13 – Parameter for test of varying no. of peers test 1

Statistic for average latency:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	3748.698	3208.939	2638.309	2256.461	2128.121	**1938.965**
Expiration	2335.171	2220.337	2042.978	1994.441	1999.441	2004.839
Mod Exp	2325.577	2227.965	2036.489	1994.254	1994.254	1999.964

The red mark is to show that the average latency of ad hoc scheme comes out to be less than the expiration and modified expiration age scheme. This might be because the cache group size is 2500 which above the server data size i.e. 2000. Also, the scenario taken into consideration is repetitive because of which same data items are accessed by different peers. Due to this, the ad hoc scheme creates copies thus resulting in large number of local hits.

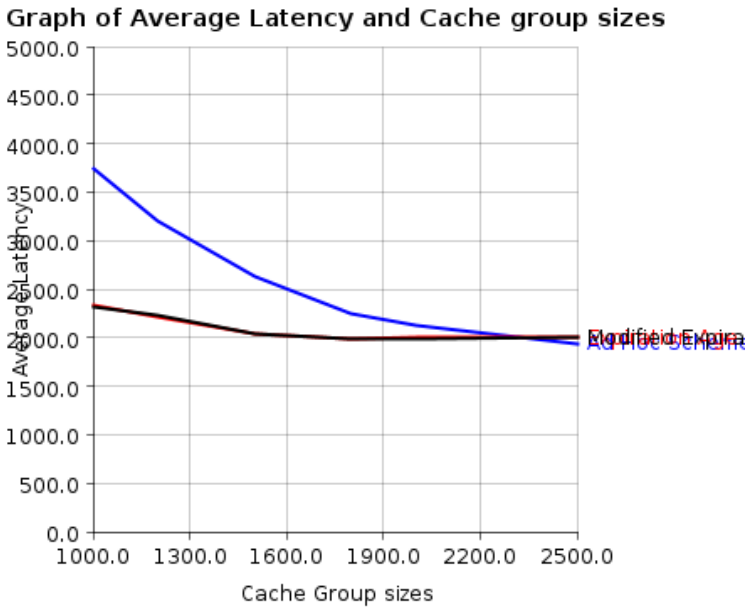

Figure 16 – Graph of average latency v/s cache group sizes for varying no. of peers test 1

Statistics for document hit ratio:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	79.47%	80.79%	82.33%	83.5%	83.92%	84.59%
Expiration	83.61%	84.08%	84.78%	85.02%	85.03%	85.03%
Mod Exp	83.67%	84.01%	84.77%	85.03%	85.03%	85.03%

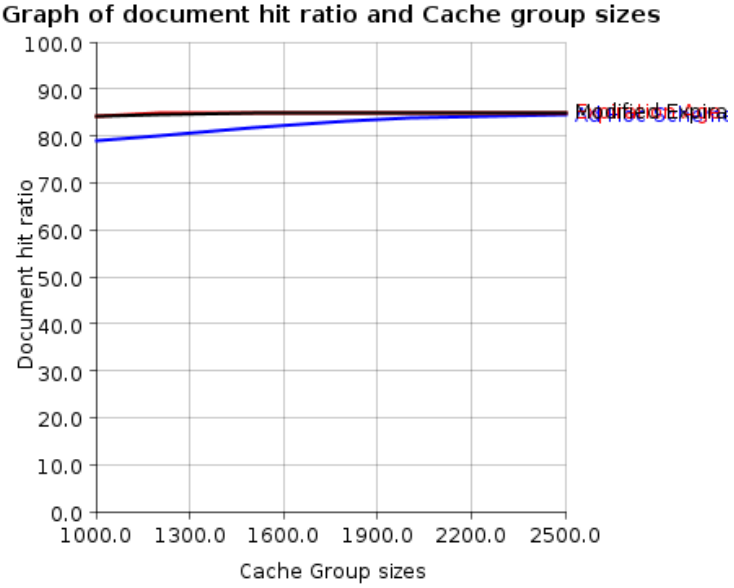

Figure 17 – Graph of doc hit ratio v/s cache group sizes for varying no. of peers test 1

The second experiment is with 1000 peers.

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	8
serverDiskProcTime	10
Filename	Testdata
noOfPeers	**1000**
noOfCaches	8
Seed	142857
SeedReq	21556
whichScenario	Repetitive

Table 14 – Parameter for test of varying no. of peers test 2

Statistics for average latency:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	4016.814	3462.865	2813.278	2346.754	2168.660	1980.394
Expiration	2012.963	1917.806	1843.347	1843.837	1843.837	1843.837
Mod Exp	2092.468	1960.288	1843.714	1843.543	1843.543	1843.543

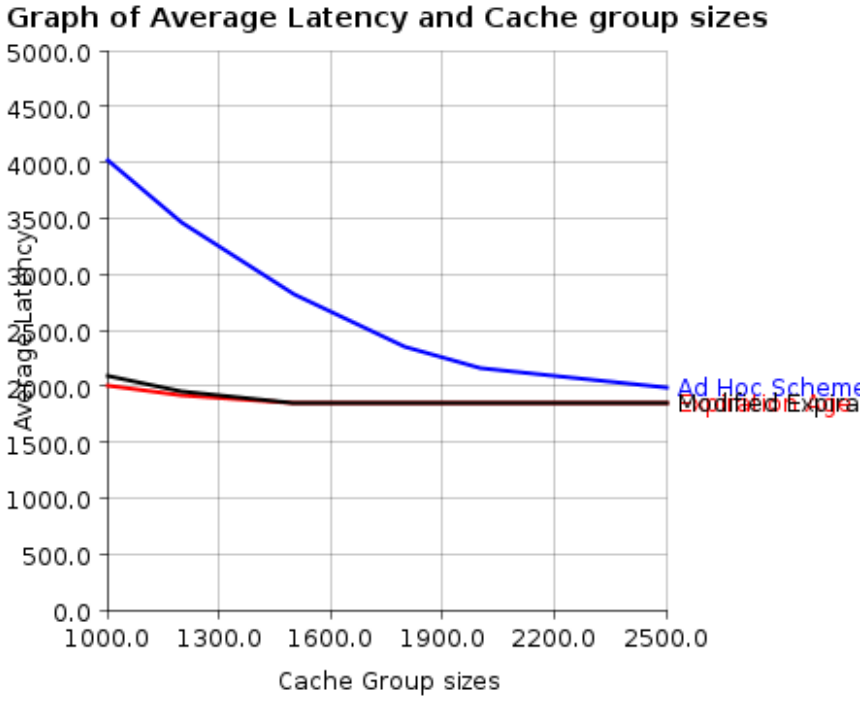

Figure 18 – Graph of average latency v/s cache group sizes for varying no. of peers test 2

Statistics for document hit ratio:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	78.83%	80.13%	81.81%	83.19%	83.77%	84.43%
Expiration	84.37%	84.73%	85.03%	85.03%	85.03%	85.03%
Mod Exp	84.08%	84.57%	85.03%	85.03%	85.03%	85.03%

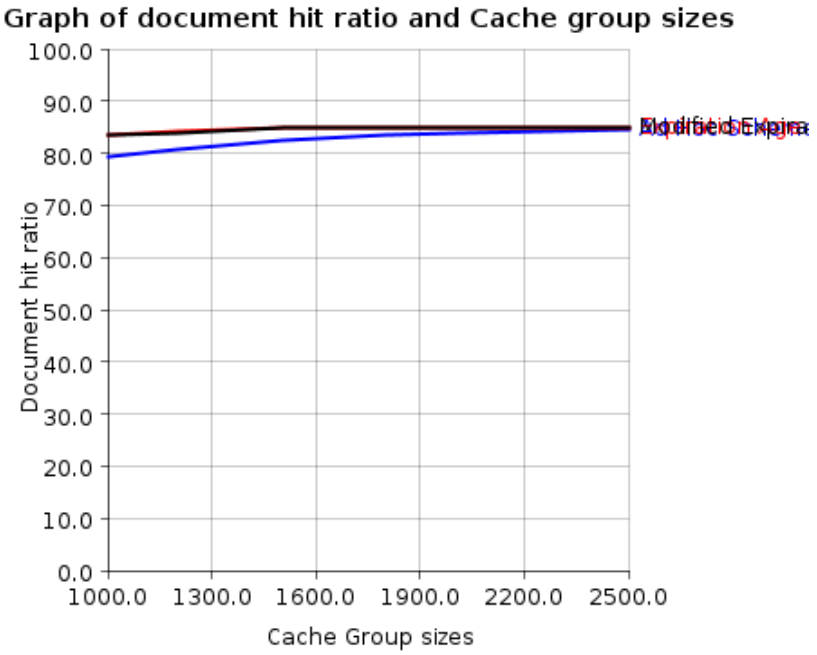

Figure 19 – Graph of doc hit ratio v/s cache group sizes for varying no. of peers test 2

As, it can be seen, the graphs are almost similar in different cases of number of peers present in the system. The few differences in the graphs are because of different amount of peers associated with caches in both experiments. To show this, below is the statistic of the number of peers per cache in both experiments.

100 Peers	1000 peers
Peers in cache 1 = 6	Peers in cache 1 = 114
Peers in cache 2 = 13	Peers in cache 2 = 116
Peers in cache 3 = 17	Peers in cache 3 = 129
Peers in cache 4 = 13	Peers in cache 4 = 117
Peers in cache 5 = 12	Peers in cache 5 = 130
Peers in cache 6 = 16	Peers in cache 6 = 143
Peers in cache 7 = 11	Peers in cache 7 = 128
Peers in cache 8 = 12	Peers in cache 8 = 123

In the first case, cache 3 has the highest number of peers associated with it whereas cache 6 has the highest number of peers associated with it in the second case. Higher the number of peers associated with a cache, more are the requests generated from them. This happens since peers are randomly assigned to caches and since requests originate from peers, we observe some differences in the average latency.

6.f Vary the server data size in the system

The next test case is varying the server data size in the system. Server data size means the amount of data items present in the server. Usually, any data item not found in the cache group is fetched from the server and the server is assumed to have all the data items. The server is nothing but a gateway to the internet where any data item can be found. However, in this project, the server has a specified data size. Peers request for data items from the specified data list. Thus, an interesting case to notice would be to vary the server data size in the system.

The server data size is kept much greater than the cache group size in this project. If the server data size is small, the number of data items in the cache group will be less. Due to this, caches won't get overloaded which will affect in the computation of expiration age of the cache.

The values for the parameters in the first experiment are as follows:

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	8
serverDiskProcTime	10
Filename	Testdata
noOfPeers	1000
noOfCaches	8
Seed	142857
SeedReq	21556
whichScenario	Repetitive
serverDataSize	**1000**

Table 15 – Parameter for test of varying server data size test 1

Statistics for average latency are as follows:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	1995.951	1529.549	1103.175	**866.943**	**794.109**	**730.959**

Expiration	893.991	880.241	896.413	906.924	906.924	906.924
Mod Exp	886.703	856.924	885.505	900.394	900.394	900.394

The results in red indicate the average latency of ad hoc scheme is less because the server data size is less. As the cache group size increases, it can accommodate more and more data items in the caches whereas the server data size is less thus all the data items are present in the caches probably multiple copies. Thus, ad hoc scheme yields less latency.

Also, the average latency for expiration age and modified expiration age scheme gives improper results i.e. is nearly constant as the caches don't get filled up due to which expiration age is never calculated.

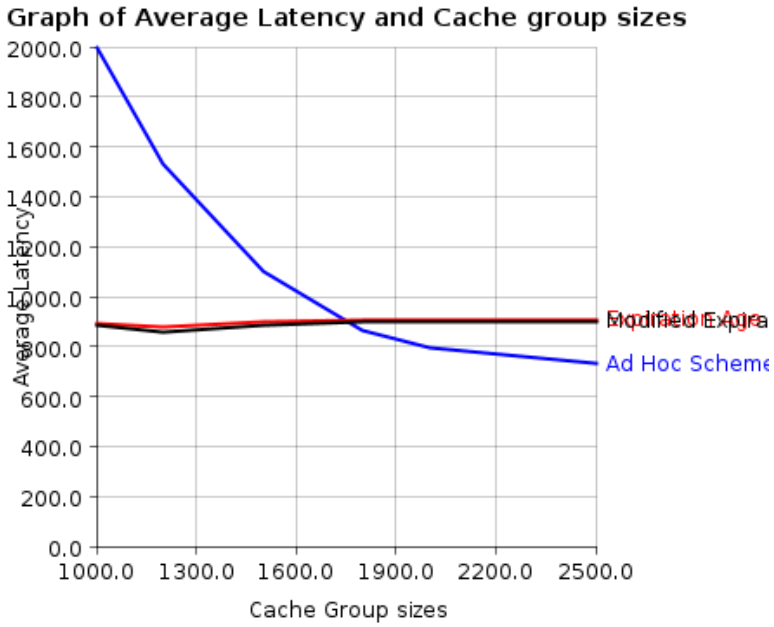

Figure 20 – Graph of average latency v/s cache group sizes for varying server data size test 1

Statistics for disk usage efficiency are as follows:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	30.2%	28.17%	26.2%	25.0%	24.55%	23.2%
Expiration	43.7%	55.91%	75.24%	**100.0%**	**100.0%**	**100.0%**
Mod Exp	41.7%	38%	69.31%	96.23%	96.23%	96.23%

The disk usage efficiency abruptly increases because of the same reason that the server data size is less thus there are less unique items in the system. Expiration age is never calculated in this case, thus no copies of data items are made. This means that caches contain only one copy of the data item thus we get 100% disk usage efficiency as cache group size increases.

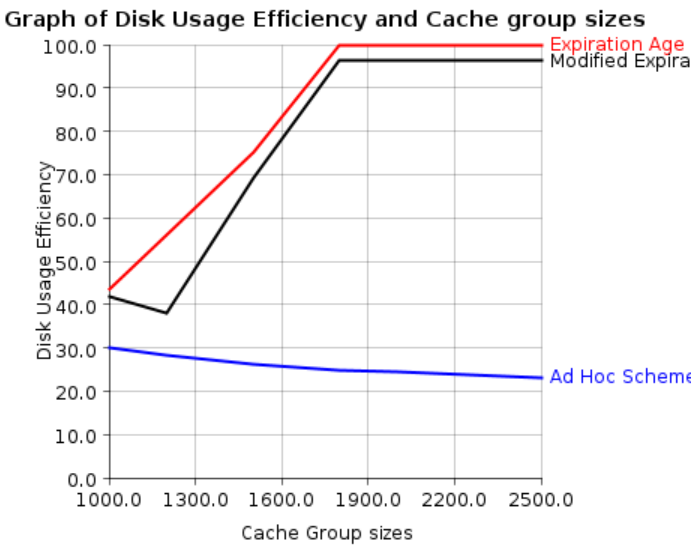

Figure 21 – Graph of disk usage efficiency v/s cache group sizes for varying server data size test 1

The values for the parameters in the second experiment are as follows:

interArrivalRate	1
noOfRequests	10000
cacheProcTime	1
serverCacheProcTime	8
serverDiskProcTime	10
Filename	Testdata
noOfPeers	1000
noOfCaches	8
Seed	142857
SeedReq	21556
whichScenario	Repetitive
serverDataSize	**2000**

Table 16 – Parameter for test of varying server data size test 2

Statistics for average latency are as follows:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	3748.698	3208.393	2638.309	2256.461	2128.121	1938.965
Expiration	2329.728	2178.798	2072.833	1992.776	1998.037	2004.839
Mod Exp	2295.204	2173.564	2067.170	1990.251	1995.680	1999.964

The average latency of ad hoc doesn't exceed except for the last one. More the server data size, more the unique data items in the system. Thus, ad hoc scheme performs poorly as expected.

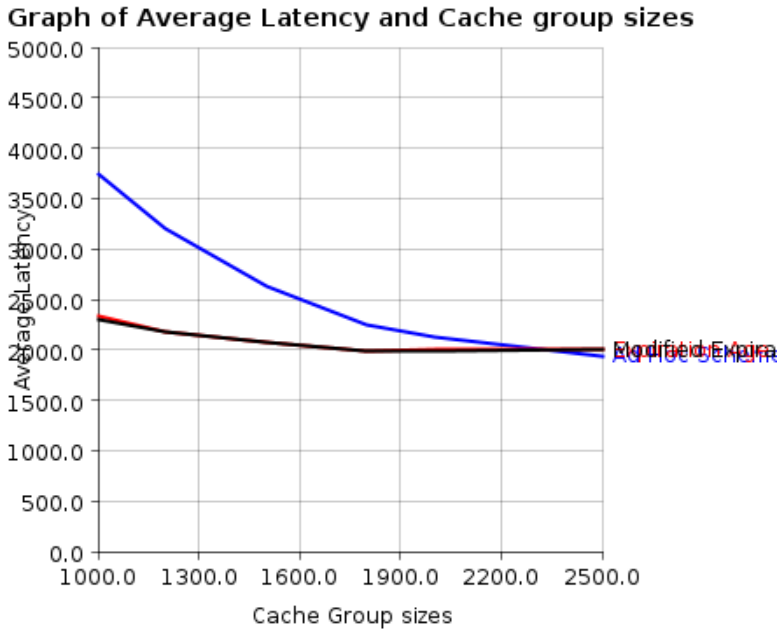

Figure 22 – Graph of average latency v/s cache group sizes for varying server data size test 2

Statistics for disk usage efficiency are as follows:

	1000 size	1200 Size	1500 Size	1800 Size	2000 Size	2500 Size
Ad Hoc	38.0%	37.17%	36.23%	35.56%	35.2%	34.5%
Expiration	56.0%	58.92%	52.74%	65.39%	74.75%	100.0%
Mod Exp	64.3%	59.92%	53.41%	75.25%	80.42%	98.55%

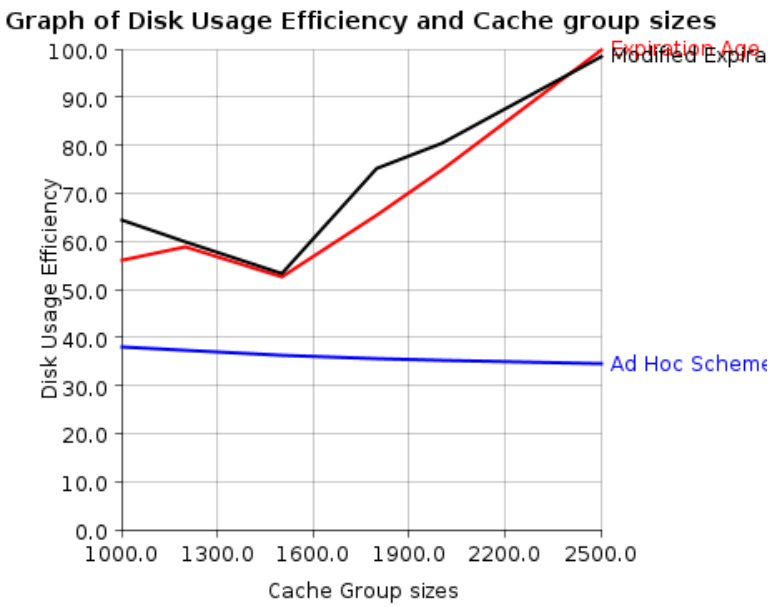

Figure 23 – Graph of disk usage efficiency v/s cache group sizes for varying server data size test 2

The disk usage efficiency does not increase abruptly in this case as compared to the previous experiment.

As it can be concluded, decreasing the server data size makes the disk usage efficiency to abruptly increase as the cache group size increases. This is because, the number of unique items is less as the server data size is less. Also, the average latency gives improper results as the caches don't get filled up due to which expiration age will never be calculated.

7. Conclusion

The Modified Expiration Age scheme performs better than Expiration Age in most of the cases because of the frequency factor. The overall hits encountered are more than the Expiration Age scheme hence reduced average latency. The reason why in some cases it doesn't perform better is because the Modified Expiration Age scheme copies the data item if the frequency of requests for that data item is over a threshold. Due to this, some data items are also evicted from the cache. Now, if there is a request for the evicted data item in the future, it will not be found in the caches and it has to be fetched from the server thus increasing the overall average latency by some value. Ad hoc scheme performs the worse as it can be seen in the results since it creates multiple copies of data items which are not necessary. To conclude, Expiration Age is a smart scheme which copies data items only when it's needed hence it performs way better than ad hoc scheme. However, as it can be seen in the results, Modified Expiration Age performs even better than Expiration Age scheme because of the additional factor taken into consideration.

8. Future Work

The possible future work which I think could be done in this project is as follows:

- The data items are of fixed size in this project. However, in the future, data items with varying sizes can be considered. Also, segments of the same data item can be done. In this way, each peer could contain certain segments of the data item rather than the whole data item itself.

- Analysis of a Heterogeneous system of this project can be studied as part of the future work instead of homogeneous system.

- Caches present in this system are of the same size. It would be interesting to study test cases when the caches in the system are of varying size.

- It is assumed in this project that there are no peer failures or peer additions to the system. Future implementation could take the peer failures or peer joining into account.

9. References

[1] Ramaswamy, L.; Ling Liu, "**An expiration age-based document placement scheme for cooperative Web caching**" *Knowledge and Data Engineering, IEEE Transactions on* , vol.16, no.5, pp.585,600, May 2004 doi: 10.1109/TKDE.2004.1277819

[2] Hefeeda, M.; Noorizadeh, B., "**On the Benefits of Cooperative Proxy Caching for Peer- to-Peer Traffic**" Parallel and Distributed Systems, IEEE Transactions on, vol.21, no.7, pp.998, 1010, July 2010 doi: 10.1109/TPDS.2009.130

[3] Ramaswamy, L.; Ling Liu; Iyengar, A., "**Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks**" *Distributed Computing Systems, 2005. ICDCS 2005 Proceedings. 25th IEEE International Conference on*, vol., no., pp.229, 238, 10-10 June 2005 doi: 10.1109/ICDCS.2005.16

[4] Ramaswamy, L.; Ling Liu, "**A new document placement scheme for cooperative caching on the Internet**" *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on* , vol., no., pp.95,103, 2002
doi: 10.1109/ICDCS.2002.1022246

[5] Hefeeda M. ; Noorizadeh B. "**Cooperative caching the case for p2p traffic**" Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on Digital Object Identifier: 10.1109/LCN.2008.4664146 Publication Year: 2008 , Page(s): 12 – 19

[6] **Parallel Java Library** by Alan Kaminsky – http://www.cs.rit.edu/~ark/pj.shtml