Spring 2011

Improving the Performance of a Proxy Server using Web log mining

Akshay Shenoy
San Jose State University

Improving the Performance of a Proxy Server using Web log mining

A Writing Project

Presented to

The Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

by

Akshay Shenoy

May 2011

ABSTRACT

Web caching techniques have been widely used with the objective of caching as many web pages and web objects in the proxy server cache as possible to improve network performance. Web pre-fetching schemes have also been widely discussed where web pages and web objects are pre-fetched into the proxy server cache. This paper presents an approach that integrates web caching and web pre-fetching approach to improve the performance of proxy server's cache.

ACKNOWLEDGEMENTS

I am thankful to my advisor, Dr. Robert Chun, for his constant guidance and support. When I began working on this project, I had very little knowledge about the topic. Dr. Robert Chun was a constant source of support and encouragement. Dr. Robert Chun provided me with enough time to understand the subject. He encouraged me to keep researching this subject. Dr. Chun is very polite and courteous. He stays until late to help students. I would like to thank him for his constant confidence in me.

I very much appreciate Dr. Teoh Soon Tee and Mr. Kartik Shah's participation as thesis committee members. My committee has provided enlightening insight, guidance and polished the work presented in this report. The three of you are precious assets to society.

It has been a challenging, yet rewarding journey, which I could not have completed alone and am grateful for your support.

Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

There has been tremendous amount of research and development in computer technology and the Internet has emerged as a platform for the sharing and exchange of information. There has been a growing demand for internet-based applications that generates high network traffic and putting much demand on the limited network infrastructure. A possible solution to the problems of growing network traffic is addition of new resources to the network infrastructure and distribution of the traffic across more resources.

Proxy servers have been used widely to reduce the network traffic by caching frequently requested web pages by using web caching (Pallis, Vakali & Pokorny, 2007). Proxy server acts as an intermediary between the web server and the web user requesting the web page. The proxy servers try to serve as many requests at the proxy server level. Proxy servers first fetch the requested web pages from the origin web servers and store the web pages in the proxy server's cache. If a user makes a request to a web page already stored in the cache, the proxy server accesses the local copy of the web page stored in the cache and serves it to the user who requested the web page. The proxy server's cache has limited capacity in terms of size of web pages that can be stored in the cache at any given time. Once the cache capacity is reached, the temporally stale web pages in the cache are discarded and replaced by newly requested web pages. The web pages stored in the proxy server cache are managed by the cache replacement algorithms. This approach of caching is called as web caching.

1.1 Problem Addressed

Web caching has been used to reduce the network traffic by caching web pages at the proxy server level. The work presented in this paper seeks to explore an analysis based pre-fetching scheme to improve the performance of the proxy server. The pre-fetching scheme interprets the user's access pattern to form a cluster of closely related pages based on the analysis of the requests from the proxy server's log files. When the user requests a web page that is part of such a cluster, other related web pages in the same cluster can be pre-fetched into the proxy server's cache in the expectation that the next set of web pages requested by the web user would be from the pre-fetched web pages. The approach presented in this paper integrates the pre-fetching approach with the web caching scheme with the objective of improving performance of the proxy server. The integrated scheme would boost the performance of the proxy server in terms of the Hit Ratio and the Byte Hit Ratio as opposed to a plain web caching approach.

2. 0 RELATED WORK

2.1 Introduction

The Internet has evolved from a system that delivered simple and static html pages to one that provides advanced services of e-learning, video conferencing and e-commerce (Pallis, Vakali & Pokorny, 2007). With the development and evolution of these services and rapid growth in the number of users, the demand for such services has been ever increasing. These services have put a great demand on the limited network infrastructure that provides support for these services. The higher demand for services and the limited network infrastructure has caused an inferior experience for the Internet users in terms of higher latency. The simplest solution to this problem is to add network resources and increase the total bandwidth of the network (Pallis, Vakali & Pokorny, 2007). However, such a solution only encourages the development of applications that consume a higher network bandwidth and deliver a richer user experience thus causing network congestion.

The possible approaches to reduce network congestion are discussed below:

2.2 Web Caching

The Web caching approach is an implementation of the page replacement concepts used in operating systems. Web caching is a passive caching approach that is implemented by the proxy server. When a web user requests a page that is not present in the proxy server cache, the request is forwarded to the origin web server, which serves the requested web page to the proxy server. The proxy server stores the web page in the

proxy server's cache and then serves the page to the web user. The web pages present in the proxy server cache are managed by the cache replacement techniques and its algorithms. The performance and effectiveness of such an approach is entirely dependent on the cache replacement techniques that are used by the proxy server. According to (Lee, An & Kim, 2009), although the web caching scheme does reduce the network traffic and overall bandwidth consumption, however the scheme has low hit rate.

2.3 Web Pre-fetching

Web pre-fetching schemes, pre-fetch web pages into the cache in the expectation that the user would request these pages in the future requests. Web pre-fetching is performed based on analysis and in a measured way. In (Lee, An, & Kim, 2009) the authors present a brief discussion about the pre-fetching schemes. Pre-fetching schemes can be classified into two types: short-term pre-fetching schemes and long-term pre-fetching schemes. A brief summary of the two approaches is given below:

Short-term pre-fetching scheme

The short-term pre-fetching scheme pre-fetches web pages in the cache by analyzing the web cache's recent access history. Based on the analysis, the scheme computes the cluster of closely related web pages and pre-fetches clusters of web pages from the origin web servers (Chen, Qiu, Chen, Nguyen & Katz, 2003). Various approaches have been suggested for the short-term pre-fetching scheme. In (Lee, An, & Kim, 2009), the authors discuss the N-th order Markov model approach in order to

predict future user requests. In (Palpanas & Mendelzon, 1999), the authors disccuss a Partial-Match (P.P.M) model.

Long-term pre-fetching scheme

In Long-term pre-fetching scheme, the popular web pages are identified by analyzing the global access pattern for the web pages (Lee, An, & Kim, 2009). In this scheme, objects with higher access frequencies and without longer update time intervals are more likely to be pre-fetched.

Thus, Web pre-fetching is a pro-active approach where the web pages are pre-fetched into the proxy server cache from the origin web servers. If the web pre-fetching results in the pre-fetching of too many web pages, it may result in a performance decrease, rather than a performance improvement (Lee, An, & Kim, 2009).

2.4 Web Usage Access Pattern Analysis

In (Lou, Liu, Lu &Yang,2002) the authors present an approach to construct a graph of the access patterns from a proxy server log file to identify closely related websites for client IP addresses. The approach identifies the IP transactions from a proxy log file by dividing the proxy log's file into transactions for each client IP address. In this approach, it is ensured that the time difference between the two subsequent requests for a client IP address is no more than 30 minutes. Furthermore, when constructing the web navigational graph, the specific page information about the page that was accessed within the website is discarded and only the website information is saved. Using all such transactions identified for a client IP address, a web navigational graph is constructed. It

details the number of transitions between different websites for each client IP address. In (Pallis, Vakali & Pokorny, 2007), the authors use an approach which identifies and pre-fetches web pages into the web cache in order to improve the performance of the cache. Web pre-fetching is based on the analysis of proxy server log files. The web pre-fetching approach can be integrated with web caching to achieve performance improvement. However, it is important to ensure that the number and size of pages that have been pre-fetched should not have an adverse impact on the performance of the cache.

The web caching approach seeks to exploit the temporal locality of the web pages already stored in the cache and they would be requested by the users. Web pre-fetching on the other hand, exploits the spatial locality of web pages that were pre-fetched beforehand based on the fact that a closely related web page had been requested.

2.5 Integration of Web caching and Web pre-fetching

Web caching and Web pre-fetching schemes have been studied in (Podlipnig & Boszormenyi, 2003) and (Teng, Chang & Chen, 2005). The studies point out that integrating the two approaches would be of great use. Another approach implements the extension of the existing cache's replacement policies used in the proxy servers (Pallis, Vakali & Pokorny, 2007). This approach is based on the clustering based pre-fetching scheme. This cluster of related web pages based on the user’s access patterns is fetched into the proxy server cache. It is notable that the web pages may belong to different websites. There are many different clustering algorithms designed to cluster the intra-website web pages or the web pages that belong to the same website. However, when

such algorithms are applied to cluster the inter-website web pages, they are not effective (Pallis, Vakali & Pokorny, 2007). Another approach discussed is one that combines the pre-fetching scheme with the web caching scheme. Such an approach helps combine the benefits of a temporal location for the Web caching approach and the spatial location benefits for the pre-fetching approach.

The short-term pre-fetching scheme discussed above, if implemented independently, is ineffective for several reasons. First, without a carefully designed scheme, the pre-fetching scheme can result in excessive requests to the web server thus creating greater network traffic (Lee, An, & Kim, 2009). It will make ineffective use of the cache memory.

2.6 Need for a Clustering based Approach

It is possible to group together all the requests made from the Client's IP addresses and assign the most popular web pages in a cluster for each Client IP address. In (Qiu, Chen, Nguyen, & Katz, 2003), the authors state that the popularity of each web object varies considerably. Also, it is difficult to select popular objects before-hand and also predict the popularity threshold. Thus, authors in (Pallis, Vakali, & Pokorny, 2007) argue that a graph based approach to cluster the web pages would be more effective.

2.7 Session Based Approach

Session based approaches have been used to infer the frequently requested web pages within a web site by analyzing the web server's access logs. In (Xiao & Zhang, 2001), the authors present an approach that obtains user's sessions from a web server log file and identifies similar sessions. In (Chen, Fu & Tong, 2003) the authors discuss two different approaches in order to identify user access session from very large web server log files based on temporal parameters. Session based measures have been used to identify frequently requested web pages within a website by analysis of a web server's access log file.

An algorithm called Rough Set Clustering which identifies sessions from a web proxy log file and clusters the users' sessions is discussed in (Jyoti, Sharma & Goel, 2009). The data gathered in the sessions can later be used to formulate the knowledge base of rules for how the next page to be accessed could be pre-fetched. A session time out policy is used. It requires that the difference between two subsequent requests be not greater than 30 minutes.

3.0 NEW APPROACH

3.1 Overview

In (Lou, Liu, Lu & Yang,2002) the authors present an approach which analyzes the user's access pattern between different websites and identifies the closely related web sites belonging to different web servers. A proxy server log file contains information about accesses to various web pages belonging to the same website, as well as web pages belonging to different websites. A web proxy server log file does not contain information about the individual users that requested the web pages. It only contains information about which web page was accessed, the timestamp it was requested, and the client IP address that requested the web page. The individual users cannot be identified based on the information present in the proxy server log file (Pallis, Vakali & Pokorny, 2007). The approach presented in (Lou, Liu, Lu & Yang, 2002) creates a graph of access sequences for each unique client IP address. The constructed graph is pruned by using confidence and support measures to identify the closely related web sites.

In (Pallis, Vakali & Pokorny, 2007), the authors evaluate the performance of the proxy server cache by implementation of web caching schemes and then attempt to improve the performance of the proxy server cache by integrating web caching with access pattern based pre-fetching of web pages. For the access pattern analysis, a graph-based approach is used, where a web navigational graph is constructed using the information obtained from the proxy server log file. The authors in (Pallis, Vakali & Pokorny, 2007) use an approach similar to the one discussed in (Lou, Liu, Lu & Yang, 2002) to cluster inter-website pages. The distinction, however, in the former paper is that they group together the similar requesting client IP addresses based on their domain and

then a web navigational graph is constructed for each client IP group as a whole. The more important difference in this approach is that they preserve only a unique set of transitions between web pages for each client IP address. The approach discussed in this paper is closer to the one discussed in the latter analysis (Lou, Liu, Lu & Yang, 2002).

According to the authors in (Lou, Liu, Lu & Yang, 2002), confidence measure is an indicator that helps in identification of closely related websites. The distribution of page references on the internet can be highly skewed in the website dimension. In addition, the use of support as a single measure for pruning the graph will result in large number of the websites being connected to a website that has large number of hits (Lou, Liu, Lu & Yang, 2002). The approach presented in this paper attempts to extend this work presented in (Lou, Liu, Lu & Yang, 2002) by identifying a strongly related set of website URLs and applying a session based approach in order to identify frequently requested intra-website pages or pages within the website. Thus, an integrated approach is discussed which first identifies the patterns of accesses between the different website URLs and uses this information to identify frequently requested web pages within the website for each website URL identified for each of the client IP addresses. The information gathered is then used to pre-fetch web pages into the proxy server cache with the objective to improve the performance of the proxy server. The complete approach is discussed in detail as follows:

3.2 Preprocessing of the proxy server Log File

The raw proxy server log files are unsuitable for access pattern analysis. The proxy server log requires effective preprocessing to remove irrelevant data from the

proxy server log file for analysis. It is important to remove all the requests from the web proxy log file that are not explicitly requested by the user.

When a user requests any page using the browser, there are a number of log entries created in the log file as a page contains other web objects like, images, javascript files and cascading style sheets apart, from the main HTML page. The requests for these web objects are made by the browser. We analyze the web user's behavior in the proxy server log mining; the requests to the web objects of images, javascript files that the user did not explicitly request are removed. Thus, we remove all log entries that are transactions about images, cascading stylesheets and javascript files. We also remove un-cacheable requests from the web proxy log file, requests that contain queries. Typically query requests contain the character "?" (Pallis, Vakali & Pokorny, 2007). Invalid requests from the proxy log file that refer to either Internal Server Errors with code and Server Side Errors are also removed from the proxy server log files.

A snapshot of the proxy server log file is presented below:

It lists the requests made to the proxy server arranged in a temporal order: with each entry having information about the originating client IP address that made the request; the web page that was requested, the time the proxy server spent in processing the request.

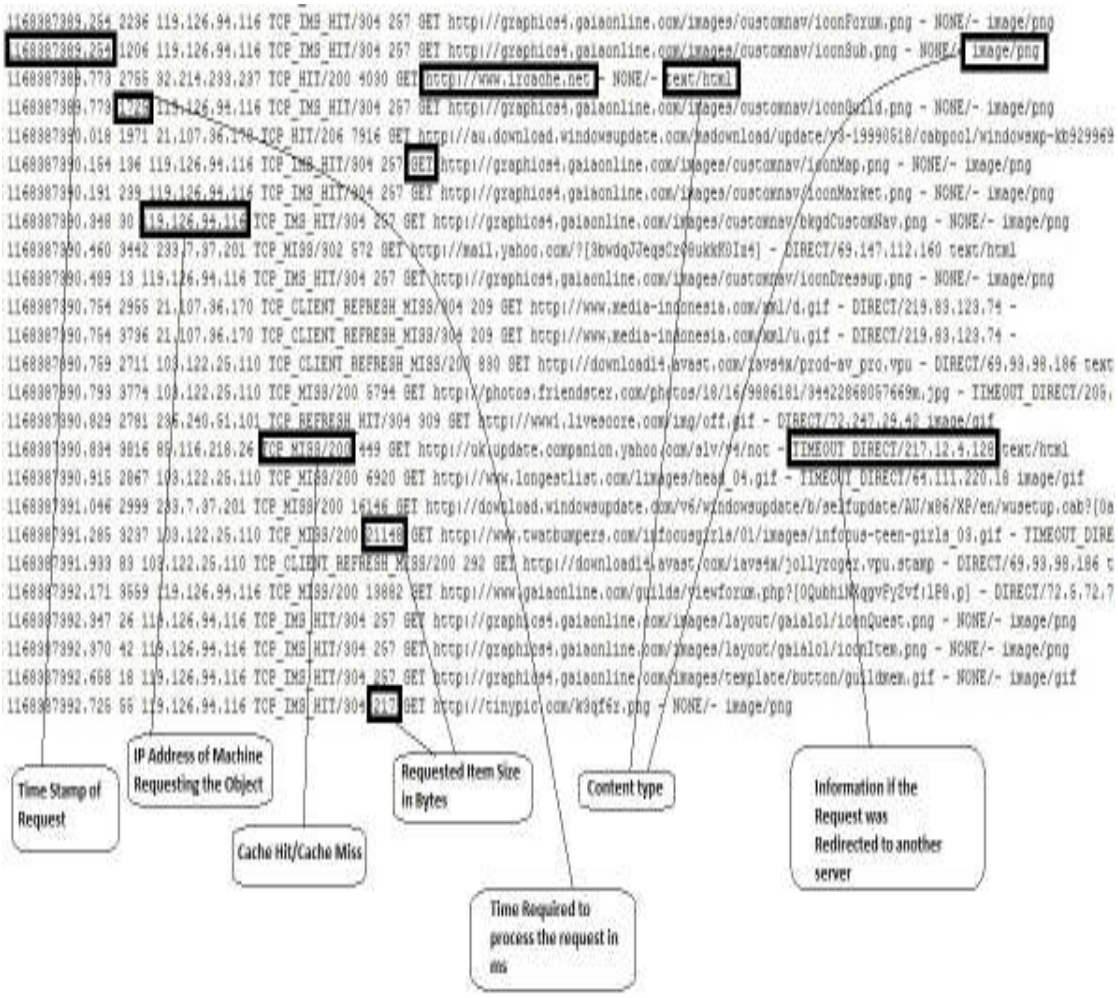

Figure 1: Snapshot of an Actual Proxy Server Log File

3.3 Segregation of the Proxy Server Log File

Once we have obtained a processed log file by removing all the irrelevant entries in the web proxy log file, the web pages frequently requested have to be identified. This is a two step process in which, we first identify the closely related websites for each client IP address and then identify the frequently requested web pages within the websites. To achieve this, we need to segregate the web proxy log file in two different ways:

1) Create a separate proxy log file for each client IP address, which stores all the web requests originating from a particular client IP address. This log file helps in identification of inter-website access patterns for each client IP address by using the inter-website clustering algorithm. Using this algorithm we identify the closely related inter-website URLs for each client IP address.

2) We need to identify the frequently requested web pages within a website by each client IP address. To achieve this, the proxy log file obtained in 1) is divided into separate log files for each website URL, which would list all the web pages visited within a website URL by the client IP address.

Log files for identifying closely related websites URLs

Inter-website cluster refers to closely related website URLs that belong to the different web servers. To achieve the objective of identifying inter-website clusters for each of the client IP addresses, we first need to identify the transactions for the client IP addresses. Thus, we segregate the web proxy log file according to the originating client IP address. The proxy server log file obtained in this step is used to construct a website traversal graph for the requests made by the client IP address.

For example, all the transactions associated with the client IP address *175.161.101.208* are grouped in a separate proxy server log file. This helps in identification of the client IP's access pattern between the different website URLs eg: www.cnn.com and news.gooogle.com are two different website URLs. We repeat this

step for each Client IP address that occurs in the processed proxy log file and create a separate web proxy log file for each client IP address.

Proxy log files for identifying intra-website pages

To achieve the objective of identifying frequently requested intra-website pages or frequently requested web pages within a website for each of the website URLs identified in the inter-website cluster, it is necessary to divide the requests made by client IP address according to the website URLs to which the requests were made for the client IP address. For example: for client IP address *175.161.101.208* all the requests made to the website URL www.cnn.com would be saved into one particular log file in order to analyze the frequently requested pages within the website URL www.cnn.com for the client IP address *175.161.101.108*. We repeat this step for each website URL to which the client IP address made requests and we create separate log files for each unique website URL that the Client IP requests.

3.4 Methodology for Clustering

We follow a two-staged approach for clustering the user access patterns.

3.4.1 Identify an inter-website cluster from the proxy server access logs using a graph based approach

3.4.2 Identify the frequently requested intra-website web pages for each of the website URLs identified in the inter-website cluster using session based measures

3.4.1 Identifying the inter-website cluster from Proxy Server Access Logs

We follow a two step process for identifying the inter-website cluster of web pages from the proxy server log file:

3.4.1.1 Construction of a website traversal graph

Based the approach mentioned in (Lou, Liu, Lu & Yang, 2002), we construct a website traversal graph for each Client IP address. A graph consists of nodes and edges that connect the nodes. A node in the graph represents a base website URL, eg: www.google.com or www.cnn.com and the edges in the graph record the number of transitions between the various website URLs. It should be noted that this website traversal graph does not save or represent any information of the web page accesses by the user within a particular website URL. There might be several web pages visited within a website URL www.cnn.com, the page information is not represented in the graph. Based on the log entries in the web proxy log file, we construct a graph that stores the information about the transitions between the different website URLs.

An Illustration for the construction of website traversal graph

Suppose an IP transaction log for a particular Client IP *175.161.101.208* contains the following page references: (*URL*1.*page1, URL*2.*page1, URL*2.*page3, URL1.page5, URL*1.*page2, and URL2.page3*), where *URLi* is a website information and page1, page2, page3 etc. are the pages visited within a website URL. By removing the page information, the resulting site access sequence looks like: (*URL*1*, URL*2*, URL*2*, URL*1*, URL*1*, URL*2). The sequence is then edited to remove subsequent requests to the same

website URL, thus further abbreviated to (*URL*1*, URL*2*, URL*1*, URL*2). Now, a website traversal graph is constructed from this transaction, it shows that the number of

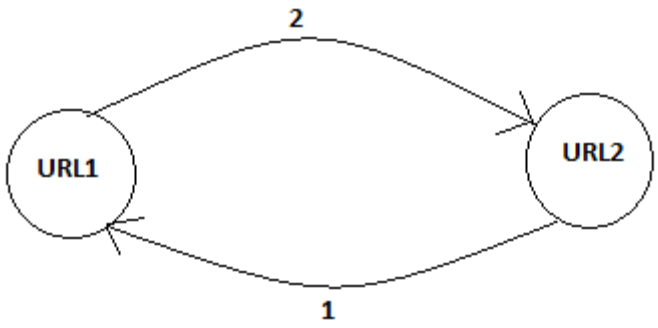

Figure 2: A Website Traversal Graph between website URLs

transitions between the website URL1 and the website URL2, originating at the website URL1 for client IP address *175.161.101.208* is 2. The number of transitions between the website URL2 and website URL1, originating at the website URL1 is 1.

Consider a website traversal graph with more transactions to identify inter-website clusters. There are 5 different website URLs in the graph, and the graph records the number of transitions between the different website URLs. The graph is labeled with popularity for nodes (a website URL) and support for the edges that connect the various website URL nodes labeled on the edges.

Popularity for a Node

Popularity for a Node is defined as sum of the number of edges leading into the node. In the above website traversal graph presented below, the popularity for the node URL2 is 5, because there are 5 transitions leading into the node URL2.

Pop(URL2) = 5

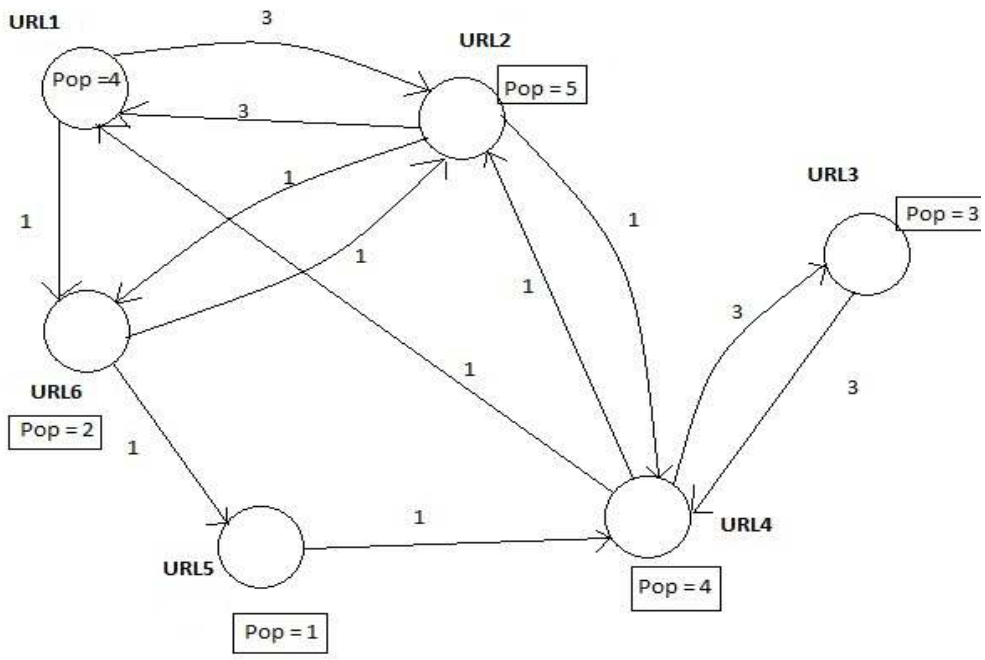

Figure 3: A Website traversal graph with popularity and Support

Support for an Edge

Support for an edge is defined as the number of transitions from starting at one node and leading to another node. For example, the support for the edge starting at the website URL5 node and leading into the website URL4 node is 1.

Supp(URL5,URL4) = freq(URL5,URL4) = 1

Confidence Measure for Edges

The next step in the process is to estimate the confidence measures for each of the edges. A revised version of the above graph with labeled confidence measures is shown

below:

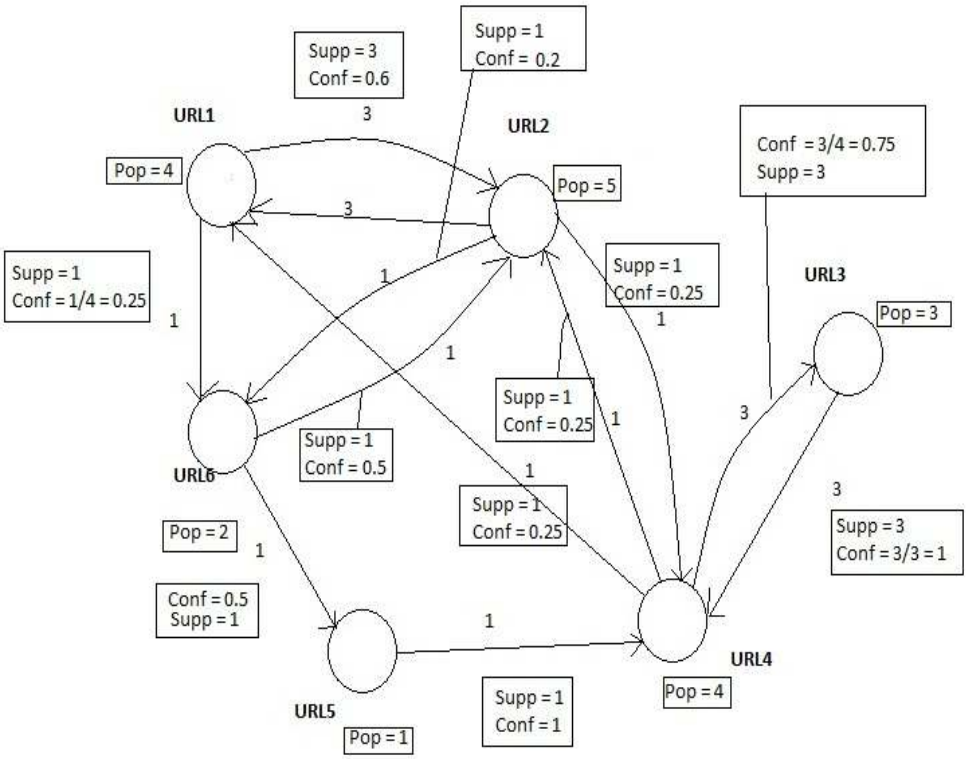

Figure 4: A Website Traversal Graph with Confidence measure

Confidence of edge (URL1, URL6) = frequency (URL1, URL6) / pop (URL1) = ¼= 0.25. Similarly, the estimation is made of confidence measures for all of the edges is estimated in the graph.

3.4.1.2 Identify the inter-website clusters from the website traversal graph

Once we have constructed the website traversal graph, we execute a clustering algorithm on the graph to identify strongly related website URLs in the graph; named as

inter-website clustering algorithm. The inter-website clustering algorithm is as explained below:

1) Construct a inter-website traversal graph G(U,V) for each client IP address where U represents a node and V represents the edges with number of transitions between edges (between website URLs)

2) For each website traversal graph, by using support measure for edges and populairty mesasure for nodes, compute the confidence measure for the edges.

3) Prune the graph by using confidence measure for edges(remove all edges from the graph whose confidence measure is less than the confidence threshold)

4) Prune the resulting graph in step 3, by using support measure for edges (remove all the edges with support measure less than the support threshold)

5) For the resulting graph in 4, apply the Breath First Traversal algorithm to identify closely related website URLs.

6) Step 5 results in sub-graphs that represent closely related inter-website URLs

Figure 5: Inter-website Clustering Algorithm

As an example, consider the above website traversal graph in Figure 4, labeled with support and confidence measures. We then apply edge filtering on the graph by filtering out all the edges from the graph that have a confidence value below a certain threshold value.

We set the minimum confidence threshold value as **0.5** and remove all the edges from the graph that do not satisfy the confidence threshold value for the edges. Since, the support measure for the edge connecting the nodes URL1 and URL2 is **0.25** and it is below the confidence threshold of **0.5,** this edge is discarded from the graph. Similarly, all the edges that have the support measures less than the confidence threshold value of

0.5 are discarded from the graph. The modified graph after applying the confidence filtering measure for all of the edges is as shown below in the figure:

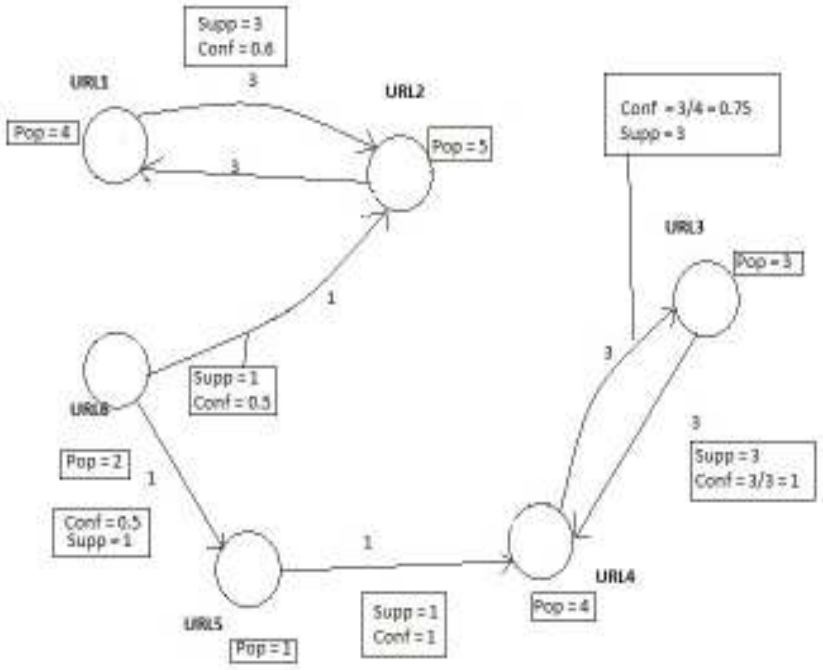

Figure 6: Graph after applying the confidence threshold

The next step in the process is to now filter all the edges, from the remnant graph with a support measure for all the edges. We set the minimum support for the edges as **2**, thus remove all the edges from the graph that do not satisfy the minimum support threshold. When the graph is edited with the support measures the modified graph is as presented below:

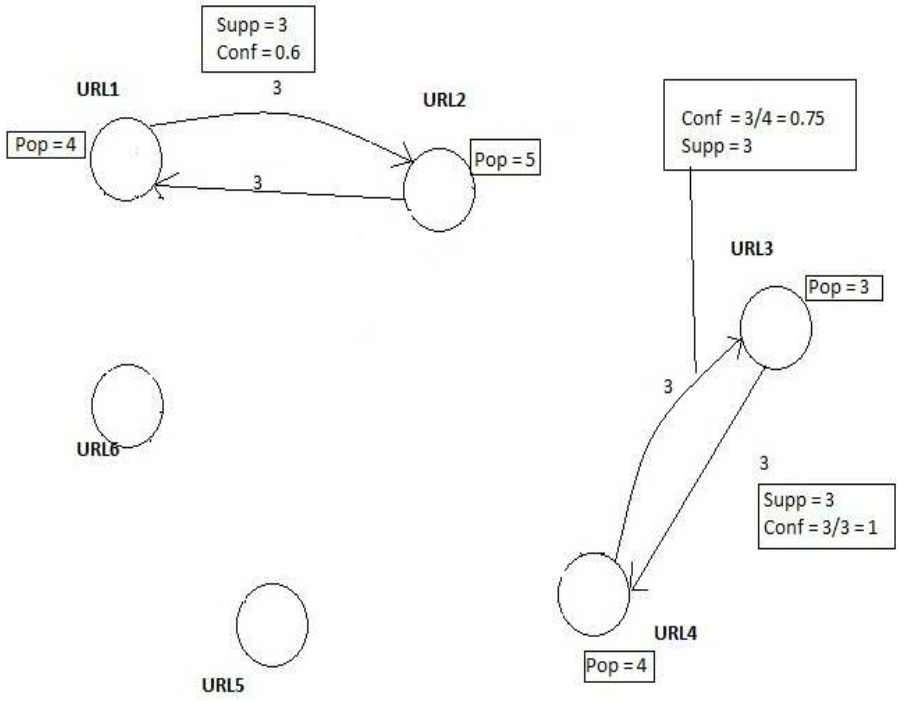

Figure 7: Graph after applying Support Threshold

Now, we apply the breadth first traversal or BFS algorithm on the graph to find out all the connected nodes in the graph. The breadth first traversal algorithm is as explained below: For the remnant graph, we process one node of the graph at a time. We start with a node say n1, mark the node n1 as visited and find all unvisited neighbors for the node n1. Add all those nodes in a sub graph T that are not connected to any other node in the graph are discarded and are not part of any inter-site cluster. The reason for discarding these nodes is because these website URLs are not strongly related to any of the other website URLs.

```
Input = Graph(U,V)
Choose some starting node n1
mark n1 as visited
List l1 = n1;
subgraph T = n1;
while l1 is nonempty;
{
 choose node n2 from front of the list;
 visit n2;
}
for each unmarked neighbor x
{
 mark x;
 add it to end of l1;
 add the edge n2x to subgraph T;
}
find all the neighbhours of node n1
visit each neighbhour and mark the visited Node
```

Figure 8: Breadth First Search Algorithm

We repeat this procedure until the graph does not contain any unvisited nodes. Each of the sub-graphs that the Breadth First Search Procedure outputs, consists of closely related inter-website URLs. Thus, as illustration inter-website cluster of closely related website URLs for the above graph is as shown in the figure below. There are two inter-site clusters for the graph above; website URL1 and website URL2 constitute inter-website cluster1 for the above graph. Website URL3 and website URL4 constitute another inter-website cluster from the above website traversal graph. Since, website URL6 and website URL5 are the nodes that are not connected to any other nodes in the graph and are not part of any inter-website clusters and they are discarded.

Thus we obtain an inter-website cluster from the original website traversal graph for each of client IP addresses. This procedure is iterated for the access pattern for every client IP address in the proxy server log file.

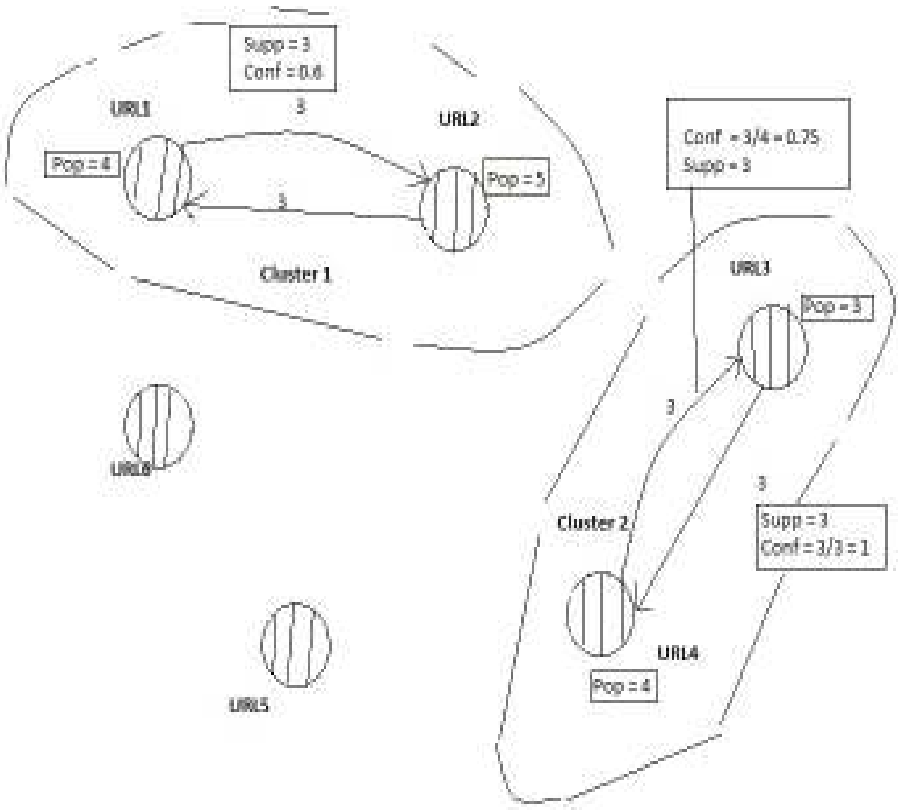

Figure 9: Graph after using the Breadth First Search

Thus we identify closely related website URLs belonging to different web sites for every client IP address. Now, the next step in the process of clustering is to identify the actual web pages requested frequently within the website URL. It should be noted that every client IP address could have a different number of inter-website clusters. For our experiments we have used inter-website support measure as 2 and the inter-website

confidence measure as 0.2. Low support and confidence measures are used to prevent interesting transitions between website URLs from being discarded from analysis.

3.4.2 Identify the intra-website pages for each inter-website URL identified

Once we have identified the closely related website URLs by using the inter-website clustering algorithm, the next step in the process is to actually identify the web pages requested within a website URL. We use the algorithm discussed below to identify the intra-website cluster of web pages for a website URL.

Intra-website Clustering Algorithm

Input - Client IP,website URL obtained from the Inter-website Clustering Algorithm, Support Threshold, Probability Threshold

Steps:

1) By using the Client IP and the URL of the Web-site, obtain the Intra-website access Log file for the the URL

2) Cut the Intra-site log file in terms of Sessions, where the time difference between the two subsequent requested item is not more than 30 min.

3) Identify all the unique intra-site pages that occur accross all the sessions and put these intra-site pages in unique object set S

4) Count the total Number of sessions

5) Count the number of times each intra-website page occurs across all the sessions.

6) Discard all the intra-site pages that have support values less than the Support Threshold.

7) Compute the probability score for each of the remaining intra-siter page by obtaining ratio of number of times the intra-web siter page occurs across session by the total number of session.

8) Now include only those intra-site pages, that satisfy the probabilty threshold.

9) Put all of the items in the intra-site Cluster of pages for the URL

Figure 10: Intra-website Clustering Algorithm

For each website URL identified in the inter-website cluster, the customized proxy log file for the website URL is accessed. In (Jyoti, Sharma & Goel,2009), the authors present an approach to divide log file is in sessions, where the time difference between the previous item's requested time and the current item's requested time is not more than 30 minutes. The next step is to construct a set of unique pages that store all the unique set of intra-website pages that occur across these various sessions. Now, for each of the unique intra-website web pages, a count is created for the number of times the intra-website page occurs across the various identified sessions.

To identify the frequently requested pages within a website URL, we use the support measure and probability measure. This involves removing all the intra-website web pages that have support measure less than a minimum support threshold. For example, all the intra-website pages that occur less than 2 times across the various sessions are removed. Now for the remaining intra-website pages, a probability score is computed by obtaining the ratio of the number of times the intra-website page occurs across all the sessions by the total number of sessions. If the web page within the URL has a probability of more than specified probability, it stores the intra-site page in the intra-site cluster for the website URL. This procedure is repeated for each of the closely related website URLs identified in the inter-website URL cluster in order to obtain the intra-website pages requested within the website URL.

For our implementation for the intra-site clustering algorithm we have used support measure as 2 and probability measure as 0.3. Support measure is chosen as 2, to make sure the web page occurs, at least twice across sessions and occurs in atleast one-third of all sessions. Since, we divide the intra-website URL proxy log for each client in

terms of sessions where the time difference between two subsequent requests is not more than 30 minutes; it is possible to have only one session for the proxy URL log for the client's IP. In that event, we identify the pages within the requested intra-website URL that satisfy the minimum support threshold.

3.5 Methods for Performance Evaluation

We test out the scheme of pre-fetching by integrating the web caching and the web pre-fetching approach and evaluate the caching performance by comparing the Hit Ratio (HR) and Byte Hit Ratio (BHR). First, the implementation of Least Recently Used (LRU) and Least Frequently Used (LFU) are tested with the testing dataset and the statistics of the Hit Ratio (HR) and Byte Hit Ratio (BHR) are computed. We then integrate the pre-fetching with web caching for the LRU and LFU based cache. A brief discussion about the various algorithms is presented below:

Least Recently Used (LRU)

In the Least Recent Used scheme, only the most recently used pages are preserved in the web cache. When the cache is full and a new page is to be stored in the cache, the least recently used page from the cache is evicted first from the cache.

Least Frequently Used (LFU)

In the Least Frequently Used scheme, only the most frequently requested items are preserved in the cache. When the cache is full and a new page is to be stored in the

cache, the least frequently used items are evicted first from the cache to create space for the new page.

Pre-fetching based Least Recently Used (LRU) and Least Frequently Used (LFU)

In this scheme, website and web page information within the website is obtained from the request that was made to the proxy server; the client IP address that made the request is also identified. In the next step, we find out if the there are any pre-computed inter-website URL clusters for the client IP address. If the client IP address has does not have any pre-computed inter-site clusters, the request is processed using the routine LRU or LFU scheme for caching. If the webpage of the website, which is requested by the client IP address does occur in the pre-computed clusters for the client IP address, the algorithm explained below pre-fetches web pages into the cache. Once the pages are brought into the cache, the cache replacement algorithms of LRU and LFU then manage these pages

Algorithm for pre-fetching of web pages

1) Identify the client IP address making the request for the web page.
2) Identify the requested website URL information from the request.
3) Obtain the page information from the website URL.
4) Search if the requested website URL occurs in any of the inter-website clusters for the client IP address. If the website URL occurs in the inter-website cluster, obtain other inter-website URLs within the same cluster.

5) Obtain the frequently requested intra-website pages for the requested website URL. Check if the web page requested by the client IP occurs in the intra-website cluster for the requested website URL. If not, the request is handled as a routine web request.
6) If the requested web page occurs in the intra-website cluster of web pages for the website URL, the scheme pre-fetches all the intra-website pages for the other inter-website URLs. Once all the pages are pre-fetched into the cache, the Least Recently Used (LRU) and Least Frequently Used (LFU) algorithms manage these web pages.

4.0 DESIGN

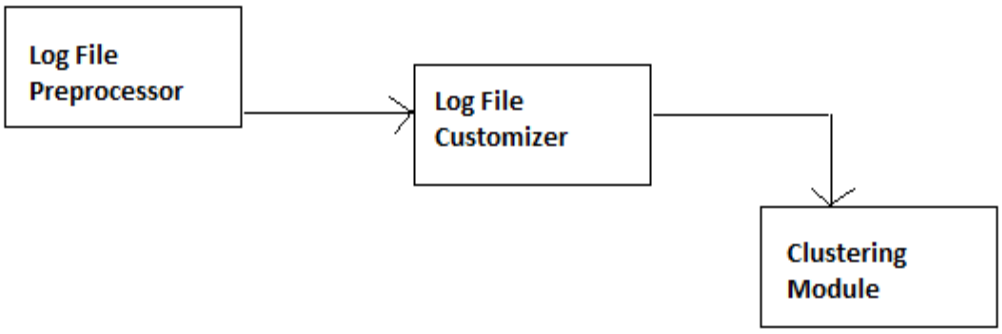

Figure 11: Design for clustering module

The complete block diagram for the clustering scheme is as shown in the diagram above. The key components in the scheme are the proxy server log file preprocessor, customized log file processor, and the clustering module.

Log File Processor

The proxy server log file processor processes the raw proxy server log file to obtain a processed log file. All the proxy server logs requests that refer to the image, css and javascript files are deleted from the log file and only those entries like web pages are explicitly requested by the user and are retained in the processed log file. The Log file processor outputs a processed proxy log file. 70% of the requests for the cleaned log file are used for sampling and 30% of the requests are used for testing.

Log File Customizer

Log File Customizer uses the data obtained from the cleaned proxy log file for sampling. A sub module of the log file customizer segregates the log files into separate log files based on the client's IP address that made the request. The second of the modules divides the client's IP based log files further into customized log files, based on the website URLs to which the requests were made.

Clustering Module

The clustering module then processes the customized log files obtained from the customized log file processor to obtain closely related websites using the graph based approach discussed and identifies the frequently requested intra-website pages within the website that are frequently requested.

The above process identifies the most popular web pages for each client IP by analyzing the sampling dataset. Once these have been identified, we run the algorithms of Least Recently Used (LRU), pre-fetching based Least Recently Used (LRU) and Least Frequently Used (LFU) and pre-fetching based Least Frequently Used (LFU) algorithms to obtain the results.

5.0 SOFTWARE TOOLS AND DETAILS ABOUT DATASET

Software Tools

All the classes were developed using JAVA programming language and Java Development Kit (JDK) 1.6. An open-source JAVA API called Java Universal Network/Graph (JUNG) was used in the implementation; it is used for analysis and modeling of the data which can be represented in terms of a graph or a network.

Details about the Dataset

The data sets for testing have been obtained from http://www.ircache.net/. IRCache is a NLANR (National Laboratory of Applied Network Research) project that encourages web caching and provides data for researchers. The files that have been used for the testing of the scheme can be obtained under: ftp://ftp.ircache.net/Traces/DITL-2007-01-09/. The proxy server log files used for testing are rtp.sanitized-access.20070109.gz and rtp.sanitized-access.20070110.gz. These represent the traces for proxy server installation at Research Triangle Park, North Carolina for the dates 01/09/2007 and 01/10/2007. Each of the dataset contains access patterns for about 160 different client IP addresses.

6.0 EXPERIMENTAL RESULTS

The scheme explained in the paper was tested with 2 different datasets. These datasets are obtained from a proxy server installation ftp://ftp.ircache.net/Traces/DITL-2007-01-09/. The filenames of the datasets used for testing of the schemes are rtp.sanitized-access.20070109.gz (Dataset 2) and rtp.sanitized-access.20070110.gz (Dataset 1) under the website.

Details for the Dataset 1

The raw proxy server log file for Dataset 1 contained the details of more than 3 million requests. After the log cleaning process was applied on the dataset, the dataset contained about 610,634 entries fit for analysis.

The finer details about the data set are as explained below:

- Total Number of Items: 610634
- Total Size of Bytes for Dataset: ~ 49 GB
- Total Number of Items used for *Clustering*: 427443
- Total Size of Bytes Used for *Clustering*: ~36 GB
- Total Number of Items used for *Testing*: 183191
- Total Size of Bytes Used for *Testing*: ~ 13GB

Researchers have used 70% of all the requests ordered by time for the user's access pattern analysis, sampling and testing (Pallis, Vakali, & Pokorny, 2007). And the remaining 30% of the requests were used for testing the scheme. We used about 427,443

requests for analysis and obtaining clusters for client IPs and about 183,191 requests for testing of the scheme.

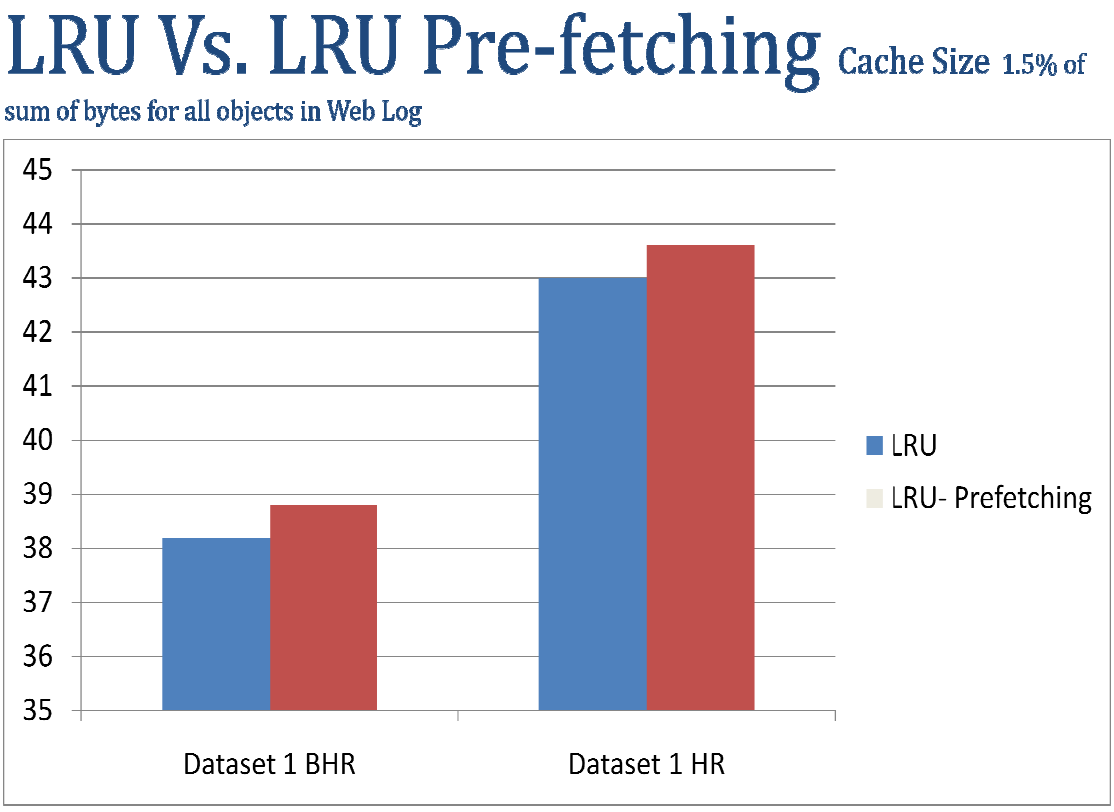

Figure 12: Dataset 1: LRU Vs Pre-fetching based LRU

We plotted a graph for Hit Rates and Byte Hit Rates obtained for the LRU and pre-fetching the LRU after testing our scheme. It is observed that the Byte Hit Ratio improves to 38.8% for pre-fetching based LRU when compared to 38.2% for a plain LRU and there is an improvement in the Hit Ratio for improves from 43.0% for LRU to 43.6% for pre-fetching based LRU for the same test data set.

Cost of pre-fetching

The total number of items that were pre-fetched into the cache was 3771. Out of these, about 2197 items did not generate cache hits. The remaining pre-fetched items generated about 10,777 cache hits. The total size of the items pre-fetched into the cache was about 1.27% of the total size of the items that the cache was tested against.

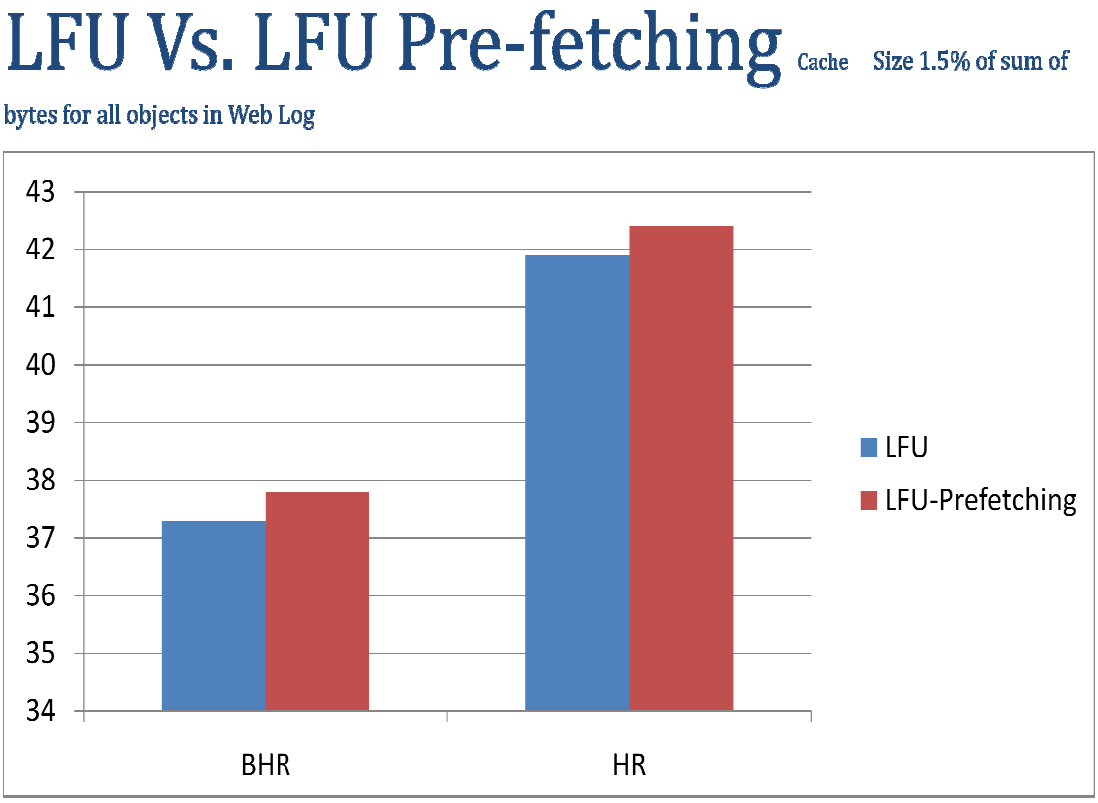

Figure 13: Dataset 1: LFU Vs Pre-fetching based LFU

The scheme has also been examined for the LFU and pre-fetching based LFU schemes and a graph is plotted for the performance metric of the Hit Ratio and the Byte Hit Ratio. It is as plotted in the figure above. The Hit Ratio of a passive LFU implementation is 37.3% against a pre-fetching based LFU where the Hit Ratio is 37.8%.

In terms of the Hit Ratio, the hit ratio for the LFU is 41.9% against that of pre-fetching based LFU is 42.4% for the same test data set.

Cost of pre-fetching

The total number of items that were pre-fetched into the cache was 4499. The total size of the items was about 197MB. Out of these, about 2304 items did not generate cache hits. The remaining pre-fetched items generated about 6772 cache hits. The total size of the items pre-fetched into the cache was about 1.48% of the total size of the items that the cache was tested against.

The schemes were also tested with another dataset. The details of the dataset are as explained below:

Details for the Dataset 2

- Total Number of Items: 694419
- Total Size of Bytes for Dataset: ~ 58GB
- Total Number of Items used for *Clustering*: 486093
- Total Size of Bytes Used for *Clustering*: ~ 40 GB
- Total Number of Items used for *Testing*: 208326
- Total Size of Bytes Used for *Testing*: ~ 18GB

The dataset 2 obtained after pre-processing the raw log file contained about all 694,419 entries that were fit for analysis and the testing of the scheme. We used about

486,093 requests for analysis and obtaining clusters for each client IP and about 208,326 requests for the testing of the scheme.

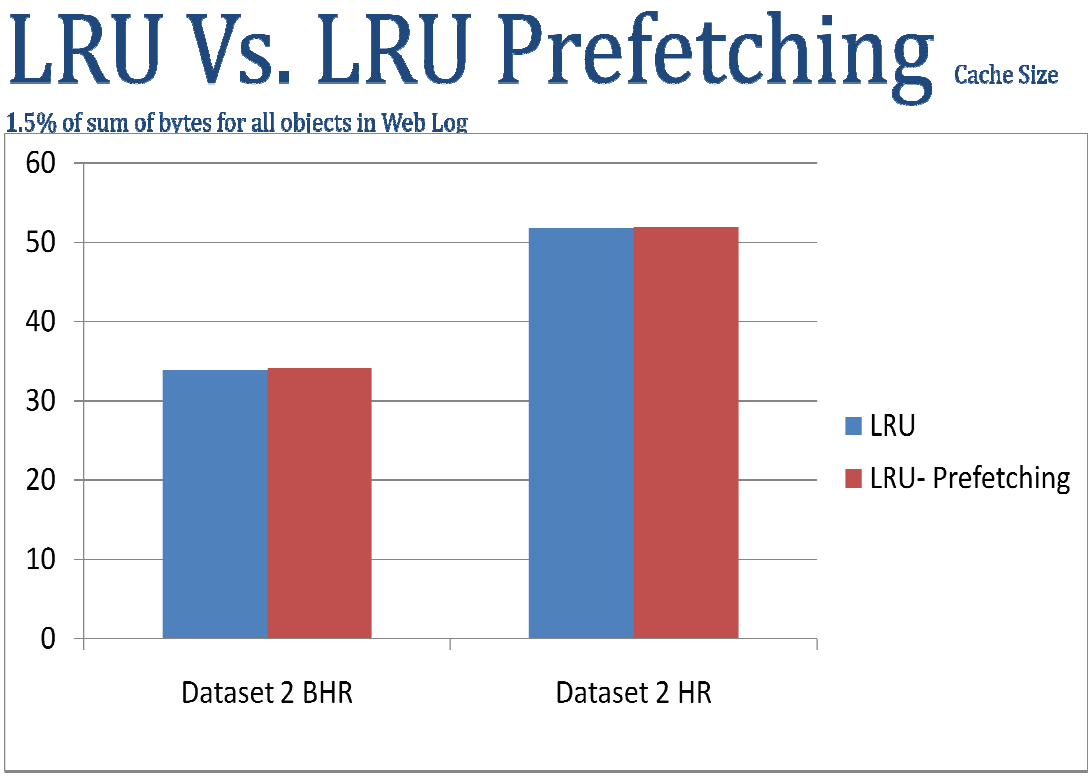

Figure 14: Dataset 2: LRU Vs Pre-fetching based LRU

We plotted a graph for the Hit Rates and Byte Hit Rates obtained for the LRU and pre-fetching LRU after testing our scheme. It is observed that the Byte Hit Ratio improves from 33.9% for LRU to 34.0% for pre-fetching based LRU and the Hit Ratio improves from 51.8% for LRU to 52.0% for pre-fetching based LRU for the same test data set.

Cost of pre-fetching

The total number of items that were pre-fetched into the cache was 6106. The total size of the items was about 275 MB. Out of these, about 3091 items did not generate cache hits. The remaining items generated 16674 cache hits. The total size of the items pre-fetched into the cache was about 1.52% of the total size of the items that the cache was tested against.

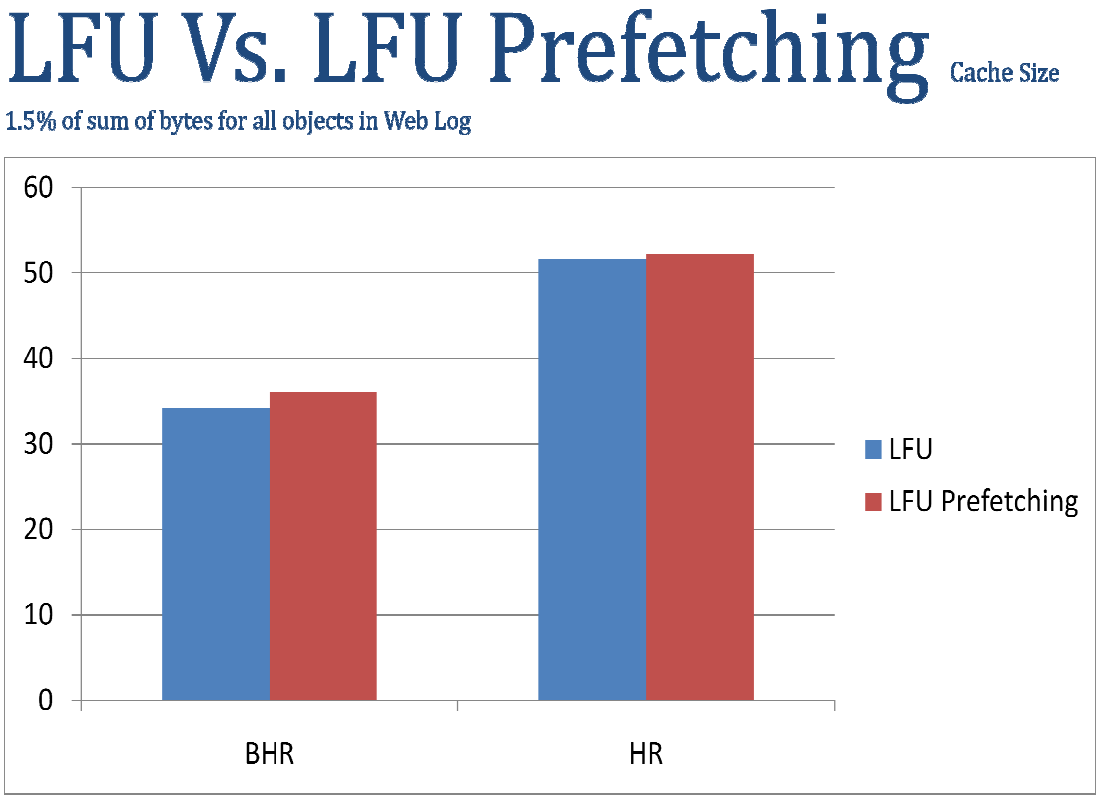

Figure 15: Dataset 2: LFU Vs Pre-fetching based LFU

The scheme has also been examined for the LFU and pre-fetching LFU and a graph is plotted for the performance metric of the Hit Ratio and Byte Hit Ratio. It is as plotted in the figure below. The Hit Ratio of a passive LFU implementation is 51% against a pre-fetching based LFU where the Hit Ratio is 52%. In terms of the Byte Hit

Ratio, the byte hit ratio for the LFU is 34.2%, against that of the pre-fetching based LFU which is 36.1% for the same test data set.

The objective of the scheme is to improve the performance of the proxy server's cache by analyzing the user's access pattern analysis for data obtained from the proxy server log. It also is to obtain clusters of popular objects from the proxy server log file for each client IP. This information is used to proactively load the cache with possible items to improve the hit ratio and the byte hit ratio for the cache.

Cost of pre-fetching

The total number of items that were pre-fetched into the cache was 6762. The total size of the items was about 430 MB. Out of these, about 3360 items did not generate cache hits. The remaining items generated 12473 cache hits. The total size of the items pre-fetched into the cache was about 2.2 % of the total size of the items that the cache was tested against.

On initial analysis there was a decrease in the hit ratio performance for the dataset 2 for the pre-fetching based LRU; on further investigation of the dataset, it was learned that the dataset was biased towards a Client's IP address. The dataset was edited to remove entries for the Client's IP address to obtain Dataset 3.

Details for the Dataset 3

- Total Number of Items: 275301
- Total Size of Bytes for Dataset: ~24GB
- Total Number of Items used for *Clustering*: 192710
- Total Size of Bytes Used for *Clustering*: ~ 18 GB
- Total Number of Items used for *Testing*: 82591
- Total Size of Bytes Used for *Testing*: ~ 6GB

We used about 192,710 requests for analysis and obtaining clusters for sampling to obtain clusters and about 82,591 requests for the testing of the scheme. Once the edited dataset was obtained, the scheme was tested with this dataset for both the pre-fetching based LRU and the pre-fetching based LFU scheme. The performance statistics for the tests are as presented below:

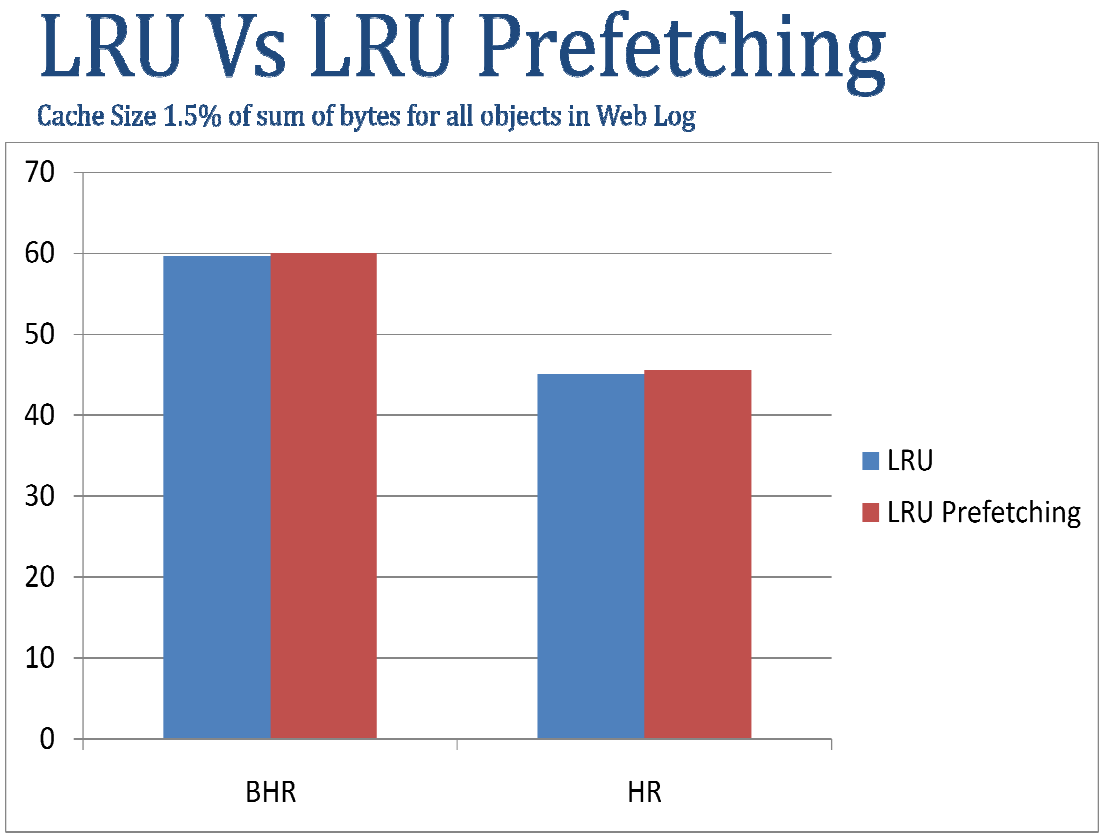

Figure 16: Dataset 3: LRU Vs Pre-fetching based LRU

The Byte Hit Ratio for the pre-fetching based LRU improved to 60%, as compared to the Byte Hit Ratio of 59.8% for LRU. In terms of the Hit Ratio, the pre-fetching based LRU had an improved Hit Ratio of 45.6% as compared to Hit Ratio of 45.1% for the LRU.

Cost of pre-fetching

The total number of items that were pre-fetched when the pre-fetching based LRU scheme was 2504. Out of these about 1515 items did not generate any cache hits. The remaining 989 items generated 5342 hits. The total size of items pre-fetched was about

47MB. The total size of the items pre-fetched was about 0.8% of total dataset size used for testing. The performance graph for LFU is as shown below:

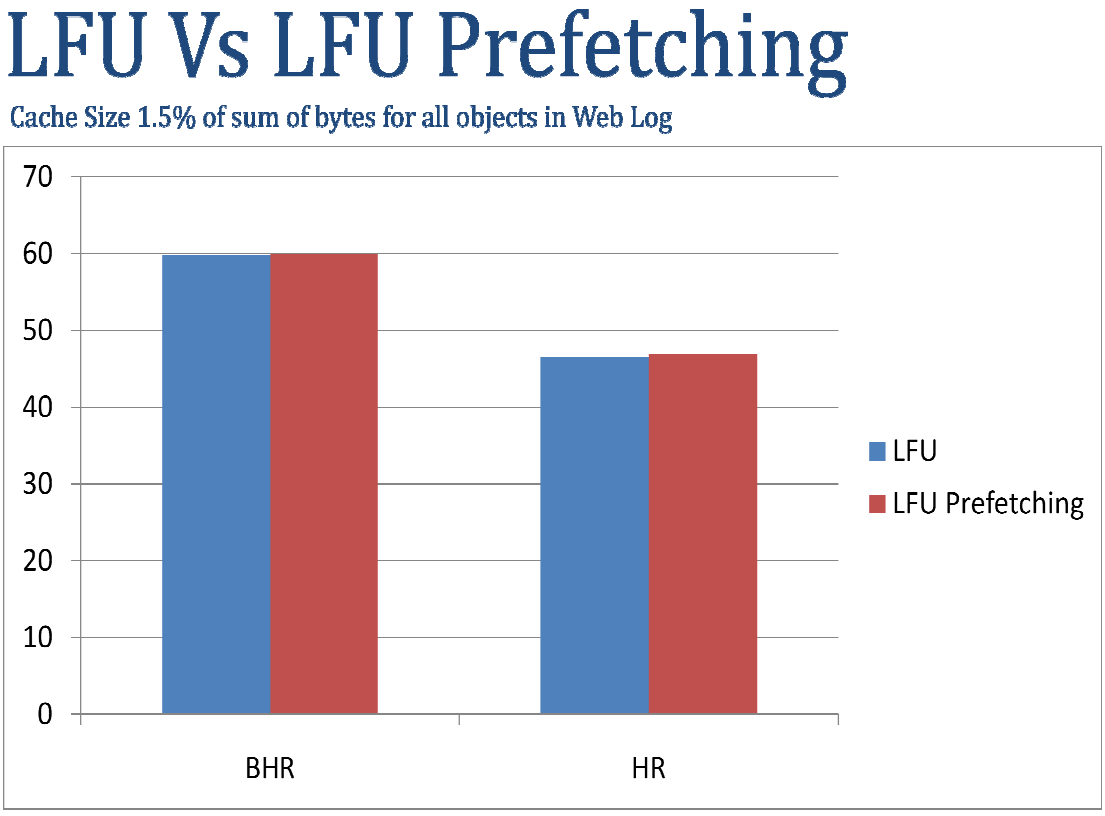

Figure 17: Dataset 3 LFU Vs Pre-fetching based LFU

The Byte Hit Ratio for pre-fetching based LFU improved to 60% from 59.8% as compared to a passive LRU implementation. The Hit Ratio improved to 45.9% for pre-fetching based LRU as opposed to 45.5% for a passive LFU scheme.

Cost of pre-fetching

The total number of items that were pre-fetched was 2641. Out of those 2124 items did not generate any cache hits. The remaining 517 items generated about 2898

cache hits. The total size of the items that were pre-fetched was about 60.4 MB. The total size of the items that was pre-fetched was about 1% of total dataset size used for testing.

Another dataset was obtained from the dataset 2 that had entries only for the Client IP address 136.75.171.165. The dataset was edited to remove entries for the Client's IP address. We used about 293,382 requests for analysis for sampling to obtain clusters and about 125,736 requests for the testing of the scheme. Details about the dataset are as presented below:

Details for the Dataset 4

- Total Number of Items: 419118
- Total Size of Bytes for Dataset: ~32GB
- Total Number of Items used for *Clustering*: 293382
- Total Size of Bytes Used for *Clustering*: ~ 20 GB
- Total Number of Items used for *Testing*: 125736
- Total Size of Bytes Used for *Testing*: ~ 12GB

Once the edited dataset was obtained, the scheme was tested with this dataset for the LRU, pre-fetching based LRU and the LFU, the pre-fetching based LFU schemes. The performance statistics for the tests are as presented below:

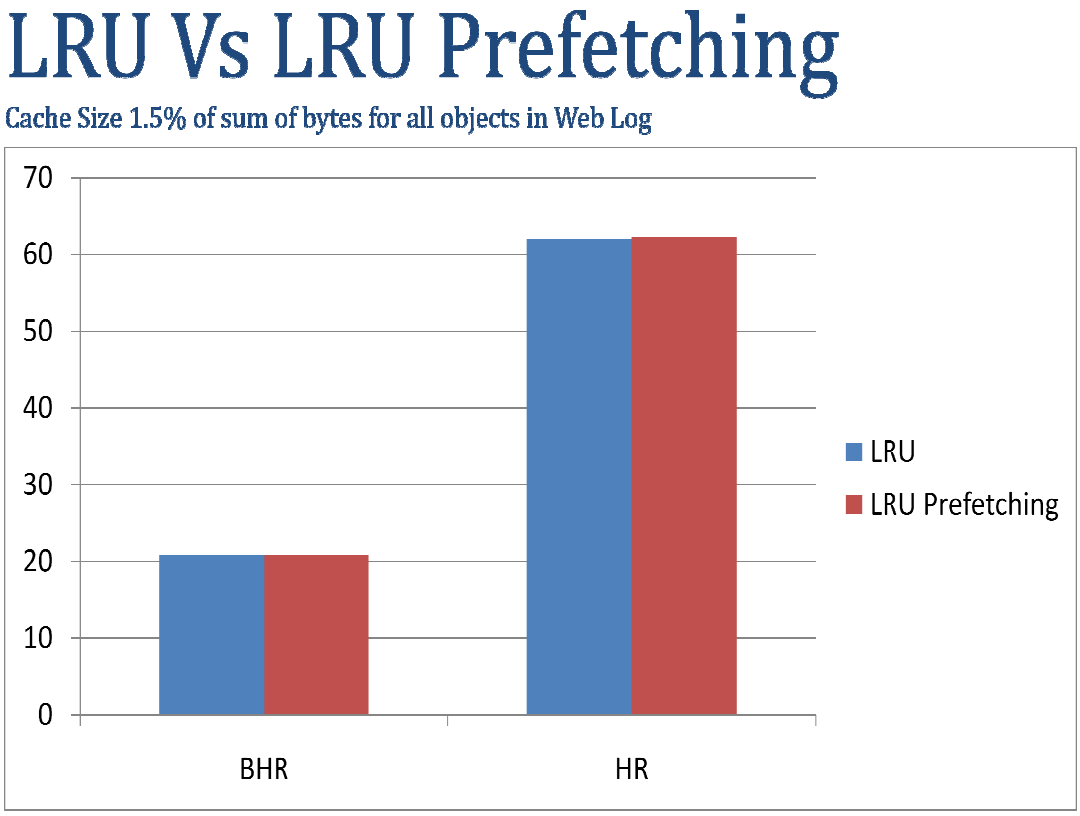

Figure 18: Dataset 4 LRU Vs Pre-fetching based LRU

The Byte Hit Ratio for pre-fetching based LRU improved to 20.9%, as compared to the Byte Hit Ratio of 20.8 % for the LRU. In terms of the Hit Ratio the pre-fetching based LRU scheme had improved Hit Ratio of 62.35%, as compared to Hit Ratio of 62.0% for LRU.

Cost of pre-fetching

The total number of items that were pre-fetched was 4331. Out of those 2153 items did not generate any cache hits. The remaining 2178 items generated about 13430 cache hits. The total size of the items that were pre-fetched was about 280 MB. The total

size of the items that was pre-fetched was about 2.2% of total dataset size used for testing.

The performance graph for LFU is as shown below:

The Byte Hit Ratio for pre-fetching based LFU improved to 21.3% from 21.2% for a passive LFU implementation. The Hit Ratio improved to 62.7% for pre-fetching based LFU as compared to Hit Ratio of 62.5% for a LFU implementation.

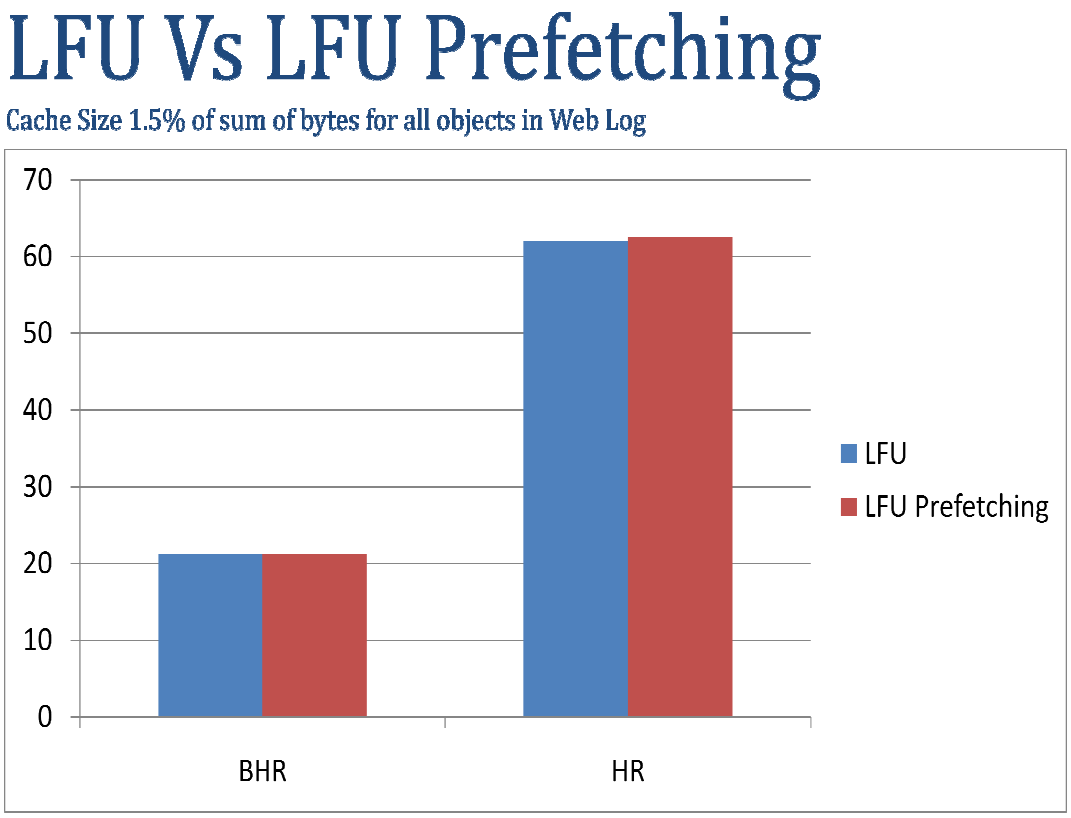

Figure 19: Dataset 4: LFU Vs Pre-fetching based LFU

Cost of pre-fetching

The total number of items that were pre-fetched was 5130. Out of those 2243 items did not generate any cache hits. The remaining 2887 items generated about 13430 cache hits. The total size of the items that were pre-fetched was about 415 MB. The total size of the items that was pre-fetched was about 3.3 % of total dataset size used for testing.

7.0 CONCLUSION

The work presented in the paper discusses a comprehensive approach to analyze web access pattern for users by using the information present in the proxy server log files. Using the approach, we identify frequently requested web pages by the users and integrate the pre-fetching scheme with the web caching to achieve performance improvement for the proxy server cache. In most of the experiments, we could observe that there was performance improvement in terms of the Hit Ratio and the Byte Hit Ratio. We also learn that the information obtained from the web proxy log file can contain large number requests of for a particular client IP address. This causes greater pre-fetching for a particular client IP address, which can result in decrease in the performance of the proxy server cache.

8.0 FUTURE WORK

First and foremost, we need to implement a mechanism that would perform an analysis of the dataset and find out if the dataset used is skewed or not. Furthermore, in our experiments we divided our dataset in terms of sampling and testing dataset, we used the about 70% of the dataset for sampling and about 30% for testing. There is scope of come up with an approach that would find out if there is an optimal way to divide the data set into sampling and testing dataset to enable the faster learning for the clustering mechanism.

9.0 REFERENCES

Pallis, G, Vakali, A, & Pokorny, J. (2008). A clustering-based prefetching scheme on a web cache environment. *Science Direct, Computer & Electrical Engineering*, *34*(4), 309-323.

Pallis, G, Angelis, L, & Vakali, A. (2007). Validation and interpretation of web users' sessions clusters. *Science Direct, Information Processing & Management*, *43*(5), 1348-1367.

Chen, Y, Qiu, L, Chen, W, Nguyen, L, & Katz, RH. (2003). Efficient and adaptive web replication using content clustering. *Selected Areas in Communications*, *IEEE Journal on 21*(6), 979-994

Ten, W, Chang, CY, & Chen, MS. (2005). Integrating web caching and web prefetching in client-side proxies. *Parallel and Distributed Systems*, *IEEE Transactions on 16*(5), 444-455

Podlipnig, S, & Boszormenyi, L. (2003). A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, *35*(4), 374-398

Palpanas T, Mendelzon A (1999). Web prefetching using partial match prediction. In: *Proceedings of the 4th International Web Caching workshop.*

Lou, W, Liu, G, Lu, H, & Yang, Q. (2002). Cut-and-pick transactions for proxy log mining. In*: Proceedings of the 8th international conference on extending database technology (EDBT 2002)*, 88–105.

Jyoti, Sharma, A, & Goel, A. (2009). A novel approach for clustering web user sessions using *RST*. *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT*, *2*(1), 656-661.

Xiao, J, & Zhang, Y. (2001). Clustering of web users using session-based similarity measures. *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*, *2*(1), 223-228.

Chen, Z, Fu, AW, & Tong, FC. (2003). Optimal algorithms for finding user access sessions from very large web logs. *World Wide Web*, *6*(3), 259-279.

Chen, W, & Zhang, X. (2002). Popularity-based PPM: An Effective web prefetching technique for high accuracy and low storage. *Parallel Processing, 2002. Proceedings. International Conference on*, *2*(1), 296-305.

Silberschatz, A, & Galvin, P. (1994). *Operating system concepts*. Addison-Wesley Publishing Company.

10.0 APPENDIX

CODE

```
package com.sjsu.edu.weblogpreprocessor;

import java.io.BufferedInputStream;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.IntrasiteRequestProcessor;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.commonhelpers.IntersiteRequestProcessor;

/**
 *
 * The component reads each line in the Proxy Server Log file
 * and determines if the requested object is
 * <br/> 1) image  - .png/.jpeg/.jpg/.gif/
 * <br/> 2) cascading style sheet css - .css
 * <br/> 3) javascript js - .js
 * <br/> 4) cgi - .cgi
 * <br/> 5) contains a query - '?'
 *
 * and is thrown out of the file as such objects are
 * useful for web usage pattern analysis
 *
 * @author Administrator
 *
 */
public class WebLogPreprocessor {
	private String inputFileName;
	private String outputFileName;
	private String baseDir;
	private Map<String,Long> uniqueClientIP;
	private Map<String,Long> uniqueDomainNames;
	private String datasetDetailsFileName;

	public WebLogPreprocessor()
	{
		uniqueClientIP = new HashMap<String, Long>();
		uniqueDomainNames = new HashMap<String, Long>();
		inputFileName = null;
		outputFileName = null;
		baseDir = null;
		datasetDetailsFileName = null;
```

```
}

/**
 * Helper to Initialize File name
 *
 * @param inputFileName
 */
public void initializeFileName(String inputFileName,
                        String outputFileName,String baseDir,String datasetDetailsFileName)
{
            this.inputFileName = inputFileName;
            this.outputFileName = outputFileName;
            this.baseDir = baseDir;
            this.datasetDetailsFileName = datasetDetailsFileName;
}

/**
 * Helper that initiates the Web Log File Reading Process
 * @throws IOException
 */
public void readFile() throws IOException
{
            long totalNoOfRequsts = Configuration.ZERO;
            File f = new File(inputFileName);

            FileInputStream fileInputStream =
                        new FileInputStream(f);

            BufferedInputStream bufferedInputStream =
                        new BufferedInputStream(fileInputStream);

            DataInputStream dataInputStream =
                        new DataInputStream(bufferedInputStream);
                        while(dataInputStream.available()!=Configuration.ZERO)
                        {
                         String logLine = dataInputStream.readLine();
                         /** Read a Line in the Log File */
                        // Request request = readLine(logLine);
                         Request request = IntrasiteRequestProcessor.processRequest(logLine);
                         String URL = request.getRequestedURL();
                         String customizedURL = request.getRequestedURL().toLowerCase().trim();
                         String customizedRequestType = request.getRequestedObjectType().trim();
                         String tcpCode = request.getTcpCode();
                          /**
                          * If the URL request is for a css,js,png,jpeg,jpg,png,gif
                          * invalidate the request
                          */

                                    if( !customizedURL.contains(Configuration.gif)
                                     && !customizedURL.contains(Configuration.jpeg)
                                     && !customizedURL.contains(Configuration.jpg)
                                     && !customizedURL.contains(Configuration.png)
                                     && !customizedURL.contains(Configuration.css)
                                     && !customizedURL.contains(Configuration.js)
                                     && !customizedURL.contains(Configuration.ico)
                                     && !customizedURL.contains("?")
                                     && !customizedURL.contains("cgi")
                                     && !customizedRequestType.contains("javascript")
                                     && !customizedRequestType.contains("image")
                                     && !customizedRequestType.contains("video")
                                     && !customizedRequestType.contains("xml")
                                     && !customizedRequestType.contains("audio")
```

```
                                        && !tcpCode.contains("4")
                                        && !tcpCode.contains("5")
                                        )
                                {

                                        /*if(request.getClientIPAddress().equals("136.75.171.165"))
                                                continue;
*/
                                        //** Write the Line to appropriate Domain File*//*
                                        writeLine(logLine, URL);

                                        totalNoOfRequsts+=Configuration.ONE;

                                        System.out.println("Number of Request:"
                                                        + totalNoOfRequsts);

                                        //+++++++++++++++++++++++++++++++++
                                        //Invoke Garbage Collection
                                        //+++++++++++++++++++++++++++++++++
                                        if(totalNoOfRequsts%Configuration.FIFTYTHOUSHAND
                                                        == Configuration.ZERO)
                                        {
                                                Runtime rc = Runtime.getRuntime();

                                                rc.gc();

                                                try
                                                {
                                                        Thread.sleep(Configuration.THOUSAND);
                                                }
                                                catch (InterruptedException e)
                                                {
                                                        e.printStackTrace();
                                                }

                                                rc.gc();
                                        }
                                        //+++++++++++++++++++++++++++++++++
                                        //Invoke Garbage Collection
                                        //+++++++++++++++++++++++++++++++++
                                }
                        }

                                printStats();
        }
        /**
         * Helper to read line in the file
         */
        private Request readLine(String logLine)
        {
                logLine = logLine.trim();
                Request request = processLine(logLine);
                return request;
        }
        /**
         * Helper to process each line in the file
         */
        private Request processLine(String logLine)
        {
                Request request = null;
                String[] fields = logLine.split(" ");
                request = createRequestObject(fields);
```

```
		//Process the ClientIP
		processClientIP(request.getcip());
		return request;
}

/**
 * Helper to process the URL
 */
private String processURL(String URL)
{
		//URL : http://www.gmail.com/skd/
		URL = URL.trim();
		int index = URL.indexOf("//");
		int index2 = -1;
		//www.gmail.com/skd/
		URL = URL.substring(index + Configuration.TWO);
		index2 = URL.indexOf("/");
		if(index2>-1)
		{
			URL =
				URL.substring(Configuration.ZERO,index2);
		}
		else
		{
			URL =
				URL.substring(Configuration.ZERO);
		}
		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");
			URL =
				URL.substring(Configuration.ZERO,index3);
		}
		if (uniqueDomainNames.containsKey(URL)) {

			Long count = uniqueDomainNames.get(URL);
			count+=1;
			uniqueDomainNames.put(URL, count);
		}
		else{
			Long count = new Long(Configuration.ONE);
			uniqueDomainNames.put(URL, count);
		}
		return URL;
}

/**
 * Helper to process the Client IP
 */
private void processClientIP(String clientIP)
{
		clientIP = clientIP.trim();
		if(uniqueClientIP.containsKey(clientIP))
		{
			Long count = uniqueClientIP.get(clientIP);
			count += Configuration.ONE;
			uniqueClientIP.put(clientIP, count);
		}
		else
		{
			Long count = new Long(Configuration.ONE);
			uniqueClientIP.put(clientIP, count);
```

```
		}
}

/**
 * Helper to write Line
 *
 * @param line
 * @throws IOException
 */
private void writeLine(String line,String URL) throws IOException
{
		String baseDirectory = baseDir;
		String pathName = null;
		String pathDir = null;
		FileWriter fileWriter = null;
		File outputFile = null;
		File outputDir = null;
		File baseParentDir = null;
		//pathName = baseDir + "\\" + URL + "\\" + URL + ".txt";
		pathName = this.outputFileName;
		pathDir = baseDir + "\\" + URL;
		baseParentDir = new File(baseDir);
		if(!baseParentDir.exists())
			baseParentDir.mkdir();
		outputDir = new File(pathDir);
		if(!outputDir.exists())
			outputDir.mkdir();
		outputFile = new File(pathName);
		fileWriter = new FileWriter(outputFile,true);
		if(outputFile.exists())
		{
			fileWriter.append(line + "\n");
		}
		else
		{
			fileWriter.write(line + "\n");
		}
		fileWriter.flush();
		fileWriter.close();
}

/**
 * Create a Request Object out of the Log File Line
 *
 * @param fields
 * @return
 */
private Request createRequestObject(String[] fields)
{
		Request request = new Request();
		//Long requestTime = new Long(fields[0].toString());
		String requestTimeString = fields[0].trim();
		Long requestTime = new Long(requestTimeString);
		Long requestProcessingTime = new Long(fields[1]);
		String clientIPAddress = fields[2].trim();
		String tcpCode = fields[3].trim();
		//Integer requestedItemSize = new Integer(fields[4]);
		String requestedItemSizeString = fields[4].trim();
		Long requestedItemSize = new Long(requestedItemSizeString);
		String requestedObject = fields[6].trim();
		//requestParam[7] --> '-'
		String timeOutRedirectionAttribute = fields[8].trim();
```

```
                String requestedObjectType = fields[9].trim();
                request.setRequestTime(requestTime);
                request.setRequestProcessingTime(requestProcessingTime);
                request.setClientIPAddress(clientIPAddress);
                request.setTcpCode(tcpCode);
                request.setRequestedItemSize(requestedItemSize);
                request.setRequestedURL(requestedObject);
                request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
                request.setRequestedObjectType(requestedObjectType);
                return request;
        }

        private void printStats() throws IOException
        {
                Set<String> uniqueDomain = uniqueDomainNames.keySet();
                File outputFile = new File(datasetDetailsFileName);
                FileWriter fileWriter = null;
                fileWriter = new FileWriter(outputFile);
                for (String string : uniqueDomain) {
                System.out.println(string);
                fileWriter.write(string + ",Count:" + uniqueDomainNames.get(string)+"\n");
        }

                Set<String> uniqueUserGroups = uniqueClientIP.keySet();
                fileWriter.write("-------------------------- \n");
                fileWriter.write("Unique User Groups \n");
                fileWriter.write("-------------------------- \n");
                for (String string : uniqueUserGroups)
                {
                        System.out.println(string);
                        fileWriter.write(string + ",Count:" + uniqueClientIP.get(string)+"\n");
                }
                fileWriter.close();
                System.out.println("Unique Domain Count:" + uniqueDomain.size());
        }
}

package com.sjsu.edu.helpers.logfilereader.clientIPURLbasedSplitter;

import java.io.BufferedInputStream;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.commonhelpers.IntersiteRequestProcessor;
public class WebLogClientIPURLSplitter
{
        String inputFileName;
        String outputFileName;
        String baseDir;
```

```
Map<String,Long> uniqueClientIP;
Map<String,Long> uniqueDomainNames;
Long seventyPercentRequests;
public WebLogClientIPURLSplitter()
{
	uniqueClientIP = new HashMap<String, Long>();
	uniqueDomainNames = new HashMap<String, Long>();
	inputFileName = null;
	outputFileName = null;
	baseDir = null;
	seventyPercentRequests = null;		}
/**
 * Helper to Initialize File name
 *
 * @param inputFileName
 */
public void initializeFileName(String inputFileName,
			String outputFileName,String baseDir, Long seventyPercentRequests)
{
	this.inputFileName = inputFileName;
	this.outputFileName = outputFileName;
	this.baseDir = baseDir;
	this.seventyPercentRequests = seventyPercentRequests;
}

/**
 * Helper that initiates the Web Log File Reading Process
 * @throws IOException
 */
public void readFile() throws IOException
{
	int totalNoOfRequsts = Configuration.ZERO;
	File f = new File(inputFileName);
	long numberOfrequests =0;
	FileInputStream fileInputStream = new FileInputStream(f);
	BufferedInputStream bufferedInputStream =
		new BufferedInputStream(fileInputStream);

	DataInputStream dataInputStream =
		new DataInputStream(bufferedInputStream);

		while(dataInputStream.available()!=Configuration.ZERO)
		{
			numberOfrequests = numberOfrequests + Configuration.ONE;
			if(!(numberOfrequests <= seventyPercentRequests))
			{
				break;
			}

		  String logLine = dataInputStream.readLine();

		  /** Read a Line in the Log File */
		  Request request = readLine(logLine);

		  String URL = request.getRequestedURL();

		  if(!URL.contains("?"))
		  {
			/** Write the Line to appropriate Domain File*/
			writeLine(logLine, URL,request);
			totalNoOfRequsts+=Configuration.ONE;
			System.out.println("Number of Request:"
```

```
                                        + totalNoOfRequsts);
                                if(totalNoOfRequsts%Configuration.FIFTYTHOUSHAND
                                        == Configuration.ZERO)
                                {
                                        Runtime rc = Runtime.getRuntime();
                                        rc.gc();
                                        try
                                        {
                                                Thread.sleep(Configuration.THOUSAND);
                                        }
                                        catch (InterruptedException e)
                                        {
                                                e.printStackTrace();
                                        }
                                        rc.gc();
                                }
                        }
                        else
                        {
                                System.out.println("URL Contains ?" + URL);
                        }
                }
                        printStats();
}

/**
 * Helper to read line in the file
 */
private Request readLine(String logLine)
{
        logLine = logLine.trim();
        Request request = processLine(logLine);
        return request;
}

/**
 * Helper to process each line in the file
 */
private Request processLine(String logLine)
{
        Request request = null;
        request = IntersiteRequestProcessor.processRequest(logLine);
        //Process the URL
        String URL = processURL(request.getRequestedURL());
        request.setRequestedURL(URL);
        //Process the ClientIP
        processClientIP(request.getcip());
        return request;
}

/**
 * Helper to process the URL
 */
private String processURL(String URL)
{
        /*//URL : http://www.gmail.com/skd/
        URL = URL.trim();

        int index = URL.indexOf("//");
        int index2 = -1;
        //www.gmail.com/skd/
        URL = URL.substring(index + CacheConstants.TWO);
```

```
		index2 = URL.indexOf("/");
		if(index2>-1)
		{
			URL =
				URL.substring(CacheConstants.ZERO,index2);
		}
		else
		{
			URL =
				URL.substring(CacheConstants.ZERO);
		}

		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");
			URL =
				URL.substring(CacheConstants.ZERO,index3);
		}
*/
		if (uniqueDomainNames.containsKey(URL)) {
			Long count = uniqueDomainNames.get(URL);
			count+=1;
			uniqueDomainNames.put(URL, count);
			}
		else{
			Long count = new Long(Configuration.ONE);
			uniqueDomainNames.put(URL, count);
		}
		return URL;
	}

	/**
	 * Helper to process the Client IP
	 */
	private void processClientIP(String clientIP)
	{
		clientIP = clientIP.trim();
		if(uniqueClientIP.containsKey(clientIP))
		{
			Long count = uniqueClientIP.get(clientIP);
			count += Configuration.ONE;
			uniqueClientIP.put(clientIP, count);
		}
		else
		{
			Long count = new Long(Configuration.ONE);
			uniqueClientIP.put(clientIP, count);
		}
	}
	/**
	 * Helper to write Line
	 *
	 * @param line
	 * @throws IOException
	 */
	private void writeLine(String line,String URL,Request request) throws IOException
	{
		String pathName = null;
		String pathDir = null;
		FileWriter fileWriter = null;
		File outputFile = null;
		File outputDir = null;
```

```
		File baseParentDir = null;

		/*pathName = baseDir + "\\" + URL + "\\" + URL + ".txt";
		pathDir = baseDir + "\\" + URL;*/
		String pathDirbase = baseDir + "\\" + request.getcip().trim();
		pathName = baseDir + "\\" + request.getcip().trim() + "\\"
		+ request.getRequestedURL() + "\\" + request.getRequestedURL() + ".txt";
		pathDir = baseDir + "\\" + request.getcip().trim()
							+ "\\" + request.getRequestedURL();
		baseParentDir = new File(baseDir);
		if(!baseParentDir.exists())
			baseParentDir.mkdir();
		outputDir = new File(pathDirbase);

		if(!outputDir.exists())
			outputDir.mkdir();

		outputDir = new File(pathDir);
		boolean outPut = false;

		if(!outputDir.exists())
			outPut = outputDir.mkdir();

		outputFile = new File(pathName);
		fileWriter = new FileWriter(outputFile,true);

		if(outputFile.exists())
		{
			fileWriter.append(line + "\n");
		}
		else
		{
			fileWriter.write(line + "\n");
		}

		fileWriter.flush();
		fileWriter.close();
}

/**
 * Create a Request Object out of the Log File Line
 *
 * @param fields
 * @return
 *//*
private Request createRequestObject(String[] fields)
{
		Request request = new Request();

		//Request Time
		String requestTimeString = fields[0].trim();

		Long requestTime = new Long(requestTimeString);

		//Request Processing Time
		Long requestProcessingTime = new Long(fields[1].trim());

		//Get the Client IP Address
		String clientIPAddress = fields[2].trim();

		//TCP Code
		String tcpCode = fields[3].trim();
```

```
		//Requested Item Size
		Long requestedItemSize = new Long(fields[4].trim());

		//Requested Web Page
		String requestedObject = fields[6].trim();

		//Time OUT Redirection Attribute
		String timeOutRedirectionAttribute = fields[8].trim();

		//Requested object type
		String requestedObjectType = fields[9].trim();


		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedURL(requestedObject);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);

		return request;
	}*/

	private void printStats() throws IOException
	{
		Set<String> uniqueDomain = uniqueDomainNames.keySet();

		File outputFile = new File(outputFileName);
		FileWriter fileWriter = null;
		fileWriter = new FileWriter(outputFile);

		for (String string : uniqueDomain) {

			System.out.println(string);
			fileWriter.write(string + ",Count:"
					+ uniqueDomainNames.get(string)+"\n");
		}

		Set<String> uniqueUserGroups = uniqueClientIP.keySet();

		fileWriter.write("-------------------------- \n");
		fileWriter.write("Unique User Groups \n");
		fileWriter.write("-------------------------- \n");

		for (String string : uniqueUserGroups) {
			System.out.println(string);
			fileWriter.write(string + ",Count:"
					+ uniqueClientIP.get(string)+"\n");
		}

		fileWriter.close();
		System.out.println("Unique Domain Count:" + uniqueDomain.size());

	}
}
package com.sjsu.edu.helpers.logfilereader.clientIPsplitter;

import java.io.BufferedInputStream;
import java.io.BufferedWriter;
```

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.commonhelpers.IntersiteRequestProcessor;


public class WebLogClientIPSplitter {

	String inputFileName;
	String outputFileName;
	String baseDir;
	Map<String,Long> uniqueClientIP;
	Map<String,Long> uniqueDomainNames;
	Long seventyPercentRequest;

	public WebLogClientIPSplitter()
	{
		uniqueClientIP = new HashMap<String, Long>();
		uniqueDomainNames = new HashMap<String, Long>();
		inputFileName = null;
		outputFileName = null;
		baseDir = null;
		seventyPercentRequest = null;
	}

	/**
	 * Helper to Initialize File name
	 *
	 * @param inputFileName
	 */
	public void initializeFileName(String inputFileName,
				String outputFileName,String baseDir, Long seventyPercentRequest)
	{
		this.inputFileName = inputFileName;
		this.outputFileName = outputFileName;
		this.baseDir = baseDir;
		this.seventyPercentRequest = seventyPercentRequest;
	}

	/**
	 * Helper that initiates the Web Log File Reading Process
	 * @throws IOException
	 */
	public void readFile() throws IOException
	{
		int totalNoOfRequsts = Configuration.ZERO;

		long numberOfRequests=0;
```

```
File f = new File(inputFileName);

FileInputStream fileInputStream =
        new FileInputStream(f);

BufferedInputStream bufferedInputStream =
        new BufferedInputStream(fileInputStream);

DataInputStream dataInputStream =
        new DataInputStream(bufferedInputStream);

        while(dataInputStream.available()!=Configuration.ZERO)
        {
                numberOfRequests = numberOfRequests + Configuration.ONE;

                if(!(numberOfRequests <= seventyPercentRequest))
                {
                        break;
                }

         String logLine = dataInputStream.readLine();

         /** Read a Line in the Log File */
         //Request request = readLine(logLine);

         Request request = IntersiteRequestProcessor.processRequest(logLine);

         String URL = request.getRequestedURL();

         if(!URL.contains("?"))
         {
                 /** Write the Line to appropriate Domain File*/
                 writeLine(logLine, URL,request);

                 totalNoOfRequsts+=Configuration.ONE;

                 System.out.println("Number of Request:"
                                 + totalNoOfRequsts);

                 //++++++++++++++++++++++++++++++++
                 //Invoke Garbage Collection
                 //++++++++++++++++++++++++++++++++
                 if(totalNoOfRequsts%Configuration.FIFTYTHOUSHAND
                                 == Configuration.ZERO)
                 {
                         Runtime rc = Runtime.getRuntime();

                         rc.gc();

                         try
                         {
                                 Thread.sleep(Configuration.THOUSAND);
                         }
                         catch (InterruptedException e)
                         {
                                 e.printStackTrace();
                         }

                         rc.gc();
                 }
                //++++++++++++++++++++++++++++++++
                //Invoke Garbage Collection
```

```
                                //+++++++++++++++++++++++++++++++
                        }
                        else
                        {
                                System.out.println("URL Contains ?" + URL);
                        }
                }

                        printStats();

}

/**
 * Helper to read line in the file
 */
private Request readLine(String logLine)
{
        logLine = logLine.trim();

        Request request = processLine(logLine);

        return request;
}

/**
 * Helper to process each line in the file
 */
private Request processLine(String logLine)
{
        Request request = null;

        String[] fields = logLine.split(" ");

        request = createRequestObject(fields);

        //Process the URL
        String URL = processURL(request.getRequestedURL());

        request.setRequestedURL(URL);

        //Process the ClientIP
        processClientIP(request.getcip());

        return request;

}

/**
 * Helper to process the URL
 */
private String processURL(String URL)
{
        //URL : http://www.gmail.com/skd/
        URL = URL.trim();

        int index = URL.indexOf("//");
        int index2 = -1;

        //www.gmail.com/skd/
        URL = URL.substring(index + Configuration.TWO);

        index2 = URL.indexOf("/");
```

```
		if(index2>-1)
		{
			URL =
				URL.substring(Configuration.ZERO,index2);
		}
		else
		{
			URL =
				URL.substring(Configuration.ZERO);
		}

		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");

			URL =
				URL.substring(Configuration.ZERO,index3);
		}

		if (uniqueDomainNames.containsKey(URL)) {

			Long count = uniqueDomainNames.get(URL);
			count+=1;
			uniqueDomainNames.put(URL, count);

		}else{

			Long count = new Long(Configuration.ONE);
			uniqueDomainNames.put(URL, count);
		}

		return URL;
}

/**
 * Helper to process the Client IP
 */
private void processClientIP(String clientIP)
{
		clientIP = clientIP.trim();

		if(uniqueClientIP.containsKey(clientIP))
		{
			Long count = uniqueClientIP.get(clientIP);
			count += Configuration.ONE;
			uniqueClientIP.put(clientIP, count);
		}
		else
		{
			Long count = new Long(Configuration.ONE);
			uniqueClientIP.put(clientIP, count);
		}
}

/**
 * Helper to write Line
 *
 * @param line
 * @throws IOException
 */
private void writeLine(String line,String URL,Request request) throws IOException
```

```
{
		String pathName = null;
		String pathDir = null;
		FileWriter fileWriter = null;
		File outputFile = null;
		File outputDir = null;
		File baseParentDir = null;

		/*pathName = baseDir + "\\" + URL + "\\" + URL + ".txt";
		pathDir = baseDir + "\\" + URL;*/

		pathName = baseDir + "\\" + request.getcip().trim() + "\\"
					+ request.getcip() + ".txt";

		pathDir = baseDir + "\\" + request.getcip().trim();

		baseParentDir = new File(baseDir);

		if(!baseParentDir.exists())
			baseParentDir.mkdir();

		outputDir = new File(pathDir);

		if(!outputDir.exists())
			outputDir.mkdir();

		outputFile = new File(pathName);
		fileWriter = new FileWriter(outputFile,true);

		if(outputFile.exists())
		{
			fileWriter.append(line + "\n");
		}
		else
		{
			fileWriter.write(line + "\n");
		}

		fileWriter.flush();
		fileWriter.close();
}

/**
 * Create a Request Object out of the Log File Line
 *
 * @param fields
 * @return
 */
private Request createRequestObject(String[] fields)
{
		Request request = new Request();

		//Request Time
		String requestTimeString = fields[0].trim();

		Long requestTime = new Long(requestTimeString);

		//Request Processing Time
		Long requestProcessingTime = new Long(fields[1].trim());

		//Get the Client IP Address
		String clientIPAddress = fields[2].trim();
```

```
		//TCP Code
		String tcpCode = fields[3].trim();

		//Requested Item Size
		Long requestedItemSize = new Long(fields[4].trim());

		//Requested Web Page
		String requestedObject = fields[6].trim();

		//Time OUT Redirection Attribute
		String timeOutRedirectionAttribute = fields[8].trim();

		//Requested object type
		String requestedObjectType = fields[9].trim();

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedURL(requestedObject);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);

		return request;
	}

	private void printStats() throws IOException
	{
		Set<String> uniqueDomain = uniqueDomainNames.keySet();

		File outputFile = new File(outputFileName);
		FileWriter fileWriter = null;
		fileWriter = new FileWriter(outputFile);

		for (String string : uniqueDomain) {

			System.out.println(string);
			fileWriter.write(string + ",Count:"
					+ uniqueDomainNames.get(string)+"\n");
		}

		Set<String> uniqueUserGroups = uniqueClientIP.keySet();

		fileWriter.write("-------------------------- \n");
		fileWriter.write("Unique User Groups \n");
		fileWriter.write("-------------------------- \n");

		for (String string : uniqueUserGroups) {
			System.out.println(string);
			fileWriter.write(string + ",Count:"
					+ uniqueClientIP.get(string)+"\n");
		}

		fileWriter.close();
		System.out.println("Unique Domain Count:" + uniqueDomain.size());

	}
}
package com.sjsu.edu.commonhelpers;
```

```
/**
 * HTTP Request Class attributes
 * @author Akshay
 *
 */
public class Request {

	/** requestTime: UNIX Timestamp for requestTime for the the Object */
	private Long requestTime;

	/** Processing time for the request in miliseconds */
	private Long requestProcessingTime;

	/** The IP Address of the Client */
	private String clientIPAddress;

	/** TCP Miss/ TCP Hit */
	private String tcpCode;

	/** Request Type : GET / POST */
	private String requestType;

	/** URL of the Requested Object as String */
	private String requestedURL;

	/** URL to which the request is forwarded in case of timeout */
	private String timeOutRedirectionAttribute;

	/** meta information of the requested Object eg: text/html */
	private String requestedObjectType;

	/** Size of the Requested Item in terms of byte */
	private Long requestedItemSize;

	public Long getRequestTime() {
		return requestTime;
	}

	public void setRequestTime(Long requestTime) {
		this.requestTime = requestTime;
	}

	public Long getRequestProcessingTime() {
		return requestProcessingTime;
	}

	public void setRequestProcessingTime(Long requestProcessingTime) {
		this.requestProcessingTime = requestProcessingTime;
	}

	public String getTcpCode() {
		return tcpCode;
	}

	public void setTcpCode(String tcpCode) {
		this.tcpCode = tcpCode;
	}

	public String getRequestType() {
		return requestType;
	}
```

```
	public void setRequestType(String requestType) {
		this.requestType = requestType;
	}

	public String getTimeOutRedirectionAttribute() {
		return timeOutRedirectionAttribute;
	}

	public void setTimeOutRedirectionAttribute(String timeOutRedirectionAttribute) {
		this.timeOutRedirectionAttribute = timeOutRedirectionAttribute;
	}

	public String getRequestedObjectType() {
		return requestedObjectType;
	}

	public void setRequestedObjectType(String requestedObjectType) {
		this.requestedObjectType = requestedObjectType;
	}

	public Long getRequestedItemSize() {
		return requestedItemSize;
	}

	public void setRequestedItemSize(Long requestedItemSize) {
		this.requestedItemSize = requestedItemSize;
	}

	public void setRequestedURL(String requestedObject) {
		this.requestedURL = requestedObject;
	}

	public String getRequestedURL() {
		return requestedURL;
	}

	public String getcip() {
		return clientIPAddress;
	}

	public void setClientIPAddress(String clientIPAddress) {
		this.clientIPAddress = clientIPAddress;
	}

}
package com.sjsu.edu.commonhelpers;

import com.sjsu.edu.commonconstants.Configuration;

public class IntrasiteRequestProcessor {

	public static Request processRequest(String logLine)
	{
		Request request = null;

		String[] fields = logLine.split(" ");

		request = createRequestObject(fields);

		return request;

	}
```

```
private static Request createRequestObject(String[] fields)
{
        Request request = new Request();

        //Long requestTime = new Long(fields[0].toString());
        int index = fields[0].indexOf(".");

        String requestTimeString = fields[0].trim().substring(0,index);

        Long requestTime = processRequestTime(requestTimeString);

        Long requestProcessingTime = new Long(fields[1]);
        String clientIPAddress = fields[2].trim();
        String tcpCode = fields[3].trim();
        Long requestedItemSize = new Long(fields[4].toString());
        String requestedObject = fields[6].trim();
        String URL = processURL(requestedObject);


        //requestParam[7] --> '-'
        String timeOutRedirectionAttribute = fields[8].trim();
        String requestedObjectType = fields[9].trim();

        request.setRequestTime(requestTime);
        request.setRequestProcessingTime(requestProcessingTime);
        request.setClientIPAddress(clientIPAddress);
        request.setTcpCode(tcpCode);
        request.setRequestedItemSize(requestedItemSize);
        request.setRequestedURL(URL);
        request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
        request.setRequestedObjectType(requestedObjectType);

        return request;
}

/**
 * Get the request time
 *
 * @param requestTimeStr
 * @return
 */
private static Long processRequestTime(String requestTimeStr)
{
        Double d = new Double(requestTimeStr);

        Long l = new Long((long) (d * Configuration.THOUSAND));

        return l;
}

/**
 * Helper to process the URL
 */
private static String processURL(String URL)
{
        //URL : http://www.gmail.com/skd/
        URL = URL.trim();

        int index = URL.indexOf("//");
        int index2 = -1;
```

```
                //www.gmail.com/skd/
                URL = URL.substring(index + Configuration.TWO);

                return URL;
        }
}
package com.sjsu.edu.commonhelpers;
import com.sjsu.edu.commonconstants.Configuration;
public class IntersiteRequestProcessor {

        public static Request processRequest(String logLine)
        {
                Request request = null;
                String[] fields = logLine.split(" ");
                request = createRequestObject(fields);
                return request;
        }

        private static Request createRequestObject(String[] fields)
        {
                Request request = new Request();
                //Long requestTime = new Long(fields[0].toString());
                int index = fields[0].indexOf(".");
                String requestTimeString = fields[0].trim().substring(0,index);
                Long requestTime = processRequestTime(requestTimeString);
                Long requestProcessingTime = new Long(fields[1]);
                String clientIPAddress = fields[2].trim();
                String tcpCode = fields[3].trim();
                Long requestedItemSize = new Long(fields[4].toString());
                String requestedObject = fields[6].trim();
                String URL = processURL(requestedObject);
                //requestParam[7] --> '-'
                String timeOutRedirectionAttribute = fields[8].trim();
                String requestedObjectType = fields[9].trim();
                request.setRequestTime(requestTime);
                request.setRequestProcessingTime(requestProcessingTime);
                request.setClientIPAddress(clientIPAddress);
                request.setTcpCode(tcpCode);
                request.setRequestedItemSize(requestedItemSize);
                request.setRequestedURL(URL);
                request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
                request.setRequestedObjectType(requestedObjectType);
                return request;
        }

        /**
         * Get the request time
         *
         * @param requestTimeStr
         * @return
         */
        private static Long processRequestTime(String requestTimeStr)
        {
                Double d = new Double(requestTimeStr);
                Long l = new Long((long) (d * Configuration.THOUSAND));
                return l;
        }
        /**
         * Helper to process the URL
         */
        private static String processURL(String URL)
        {
```

```
                //URL : http://www.gmail.com/skd/
                URL = URL.trim();
                int index = URL.indexOf("//");
                int index2 = -1;
                //www.gmail.com/skd/
                URL = URL.substring(index + Configuration.TWO);
                index2 = URL.indexOf("/");
                if(index2>-1)
                {
                        URL =
                                URL.substring(Configuration.ZERO,index2);
                }
                else
                {
                        URL =
                                URL.substring(Configuration.ZERO);
                }
                if(URL.contains(":"))
                {
                        int index3 = URL.indexOf(":");
                        URL =
                                URL.substring(Configuration.ZERO,index3);
                }
                return URL;
        }
}
package com.sjsu.edu.commonhelpers;

public class GarbageCollectorHelper {

        public static void invokeGC() throws InterruptedException
        {
                Runtime rc = Runtime.getRuntime();

                rc.gc();

                try
                {
                        Thread.sleep(5);
                }
                catch (InterruptedException e)
                {
                        e.printStackTrace();
                        throw new InterruptedException();
                }

                rc.gc();
        }

}
package com.sjsu.edu.commonconstants;

public class Configuration {

        /**
         * Numerical Value 10000
         */
        public static int TENTHOUSAND = 10000;

        /** Numerical Value 50000*/
        public static int FIFTYTHOUSHAND = 50000;
```

```
/** Numerical Value 2690*/
public static int TWOTHOUSANDSIXHUNDREDNINTY = 2690;

/** Numerical Value 50000*/
public static int TWENTYFIVETHOUSHAND = 25000;

/** Numerical Value 0 */
public static int ZERO = 0;

/** Numerical Value 2000 */
public static int TWOTHOUSAND = 2000;

/** Numerical Value 20095 */
public static int TWENTYTHOUSAND95 = 20095;

public static int TWENTYTHOUSAND596 = 20596;

public static int Seven916 = 7916;

/** Numerical Value 1 */
public static int ONE = 1;

/** Numerical Value 2 */
public static int TWO = 2;

/** Numerical Value 3 */
public static int THREE = 3;

/** Numerical Value 4 */
public static int FOUR = 4;

/** Numeric value 1000 */
public static int THOUSAND = 1000;

/** Numeric value 30 */
public static int THIRTY = 30;


/** Numeric value 60 */
public static int SIXTY = 60;

public static String dp = "136.75.171.165";

/**
 * for css extension
 */
public static String css = "css";

/**
 * for js extension
 */
public static String js = "js";

/**
 * for jpg extension
 */
public static String jpg = "jpg";

/**
 * for jpeg extension
 */
public static String jpeg = "jpeg";
```

```
/**
 * for gif extension
 */
public static String gif ="gif";

/**
 * for png extension
 */
public static String png="png";


/**
 * for png extension
 */
public static String ico="ico";

/**
 * Directory for Log Files Segregated based on the client IP
 *
 * <b>D:\ClientIPCustomizedLogFile</b>
 *
 */
public static String cIPLogFileBaseDir = "D:\\ClientIPCustomizedLogFile";

/**
 * Directory for Log files when client IP based log files are
 * segregated based on the URL requested
 *
 * <b>D:\CustomizedLogFile</b>
 *
 */
public static String cIPCustomFileBaseDir = "D:\\CustomizedLogFile";

/**
 * Directory for Client IP Intrasite Clusters
 *
 * <b>D:\ClusteredWebObjects\IntrasiteClusters</b>
 *
 */
public static String cIPIntrasiteClusterDir = "D:\\ClusteredWebObjects\\IntrasiteClusters";


/**
 * Directory for Client IP Intersite Clusters
 *
 * <b>D:\ClusteredWebObjects\IntersiteClusters</b>
 */
public static String cIPIntersiteClusterDir = "D:\\ClusteredWebObjects\\IntersiteClusters";

/**
 * Data set Path 1:<b>processed.rtp.sanitized-access.20070110</b>
 */
public static String dataSet1Path = "E:\\CentralServerLogDump\\rtp.sanitized-access.20070110\\" +
                    "processed.rtp.sanitized-access.20070110";

/**
 * <b>Intersite Support Threshold
 */
public static Integer intraSiteSupport1Threshold = new Integer(15);

/**
```

```
 * Size of the Cache for the Dataset 1
 *
 * <b>Size: 788127966 bytes</b>
 */
public static Long dataSet1CacheSize = new Long(788127966);

/**
 * Size of the Cache for the Dataset 2
 *
 * <b>Size: 788127966 bytes</b>
 */
public static Long dataSet2CacheSize = new Long(930621808);

public static Long dataSet4CacheSize = new Long(521426634);
public static Long dataSet3CacheSize = new Long(389476788);

/**
 * Data set Path 2
 *
 * <b>processed.rtp.sanitized-access.20070109</b>
 */
public static String dataSet2Path = "E:\\CentralServerLogDump\\rtp.sanitized-access.20070109\\" +
                    "Actualprocessed.rtp.sanitized-access.20070109";

public static String dataSet4Path = "E:\\CentralServerLogDump\\rtp.sanitized-access.20070109\\" +
"clientIPprocessed.rtp.sanitized-access.txt";

/**
 * <b>Data set Path 4, for Other Client IP than 136.75.171.165</b>
 */
public static String dataSet3Path = "E:\\CentralServerLogDump\\rtp.sanitized-access.20070109" +
"\\other.processed.rtp.sanitized-access.txt";

/**
 *
 * <b> 70% Requests for Data Set 3</b>
 */
public static Long seventyPercentDataSet4 = new Long(293380);

/**
 *
 * <b> 70% Requests for Data Set 3</b>
 */
public static Long seventyPercentDataSet3 = new Long(192710);


/**
 * File Path to Store the Result of the Analysis
 *
 * <b>E:\CentralServerLogDump\results.txt</b>
 */
public static String resultsPath = "E:\\CentralServerLogDump\\results.txt";

/**
 * <b> 70% Requests for Data Set 1</b>
 *
 */
public static Long seventyPercentDataSet1 = new Long(427443);

/**
 * <b> 70% Requests for Data Set 2</b>
 */
```

```
	public static Long seventyPercentDataSet2 = new Long(486094);

	/**
	 * <b>Intersite Confidence Threshold
	 */
	public static Double interSiteConfidenceThreshold = new Double(0.2);

	/**
	 * <b>Intersite Support Threshold
	 */
	public static Integer interSiteSupportThreshold = new Integer(2);

	/**
	 * <b>Intrasite Confidence Threshold
	 */
	public static Double intraSiteConfidenceThreshold = new Double(0.3);

	/**
	 * <b>Intersite Support Threshold
	 */
	public static Integer intraSiteSupportThreshold = new Integer(2);

	public static String cleanDataSet3Path = "E:\\CentralServerLogDump\\rtp.sanitized-access.20070109" +
	"\\all.cleaned.processed.rtp.sanitized-access.txt";
}


package com.sjsu.edu.domainprocessor;

import java.io.BufferedWriter;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Map;
import java.util.Set;

public class LogProcessorTest {

	public static void main(String[] args)
	{
		String baseDir = "D:\\CustomizedLogFile";

		File baseDirectory = new File(baseDir);

		String listOfDir[] = baseDirectory.list();

		for (String string : listOfDir) {

			File subDirFiles = new File(baseDir + "\\" + string);

			String[] listOfSubDir = subDirFiles.list();

			if(listOfSubDir!=null)
				for (String string2 : listOfSubDir) {

					String pathToFile = subDirFiles + "\\" + string2;
					LogProcessorTest.processCluster(pathToFile);
					System.out.println(pathToFile);

				}
		}
```

```
}

public static void processCluster(String inputWebLogFile)
{

	String inputFile =
		new String(inputWebLogFile);
	//new String("D:\\junk\\sprint.txt");
	//new String("D:\\junk\\sprint1.txt");
	//new String("D:\\CustomizedLogFile\\www.sprint.com\\www.sprint.com.txt");

	//"D:\\CustomizedLogFile\\157.91.28.89\\157.91.28.89.txt";
	//String outputFile = new String("E:\\CentralServerLogDump\\sprint.txt");

	String baseDir = "D:\\CustomizedLogFile";
	//Map<String,Integer> clusteredObjects = null;
	Map<String,Long> clusteredObjects = null;
	Set<String> clusteredObjectSet = null;
	//WebLogClientIPSplitter logFileReader = null;
	LogProcessor logFileReader = null;
	FileWriter fstream = null;
	BufferedWriter out = null;

	int index = inputFile.lastIndexOf("\\") + 1;
	int lastIndexOfDot = inputFile.lastIndexOf(".");

	String dmnString = inputFile.substring(index, lastIndexOfDot);
	String domainName = "http://" + dmnString ;
	String outputFile = "D:\\ClusteredWebObjects" + "\\" + dmnString + ".txt";

	/*logFileReader = new WebLogClientIPSplitter();
	logFileReader.initializeFileName(inputFile, outputFile, baseDir);*/

	logFileReader = new LogProcessor(inputFile, outputFile, baseDir);

	try {

		clusteredObjects = logFileReader.readFile();

		if(clusteredObjects!=null && clusteredObjects.size() > 0)
		{
			clusteredObjectSet = clusteredObjects.keySet();
			fstream = new FileWriter(outputFile);
		   out = new BufferedWriter(fstream);

		   for (String string : clusteredObjectSet)
		   {
				out.write(domainName + string + "," + clusteredObjects.get(string) +"\n");
			}

		   //Close the output stream
		   out.close();
		}
	}
	catch (IOException e)
	{
		e.printStackTrace();
	}

}
```

```
}package com.sjsu.edu.domainprocessor;

import java.io.BufferedInputStream;


import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.commonhelpers.IntrasiteRequestProcessor;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.domainprocessor.helpers.ClusterProcessor;



/**
 *
 * The Log processor is the main component that processes the
 * unique individual log files
 *
 * @author Administrator
 *
 */
public class LogProcessor {

	private String inputFileName;
	private String outputFileName;
	private String baseDir;
	private Map<String,Long> uniqueDomainNames;
	private ClusterProcessor clusterProcessor;

	public LogProcessor(String inputFile,String outputFile,
				String baseDir)
	{
		this.inputFileName = inputFile;
		this.outputFileName = outputFile;
		this.baseDir = baseDir;
		uniqueDomainNames = new HashMap<String, Long>();
		clusterProcessor = new ClusterProcessor();
	}

	/**
	 * Helper that initiates the Web Log File Reading Process
	 * @throws IOException
	 */
	//public Map<String,Integer> readFile() throws IOException
	public Map<String,Long> readFile() throws IOException
	{
		File f = new File(inputFileName);

		FileInputStream fileInputStream =
			new FileInputStream(f);
```

```
		BufferedInputStream bufferedInputStream =
			new BufferedInputStream(fileInputStream);

		DataInputStream dataInputStream =
			new DataInputStream(bufferedInputStream);

		 //readLines(dataInputStream);

		return readLines(dataInputStream);

		//printStats();

}

//private Map<String,Integer> readLines(DataInputStream dataInputStream) throws IOException
private Map<String,Long> readLines(DataInputStream dataInputStream) throws IOException
{
		StringBuffer requestedItemBuffer = null;
		List<StringBuffer> sessionList = new ArrayList<StringBuffer>();
		//Set<String> uniqueObjSet = new HashSet<String>();
		Map<String,Long> uniqueObjSet = new HashMap<String,Long>();
		Request request = null;
		String logLine = null;
		Long requestTime = null;
		Long requestTimePlus = null;
		String requestedObject = null;
		//Set<String> clusteredObjects = null;
		//Map<String,Double> clusteredObjects = null;
		Map<String,Long> clusteredObjects = null;

		int counter = Configuration.ZERO;

		while(dataInputStream.available()!=Configuration.ZERO)
		{
		 logLine = dataInputStream.readLine();

		 //request = processLine(logLine);

		 request = IntrasiteRequestProcessor.processRequest(logLine);

		 //Get the request Time for the first Line
		 requestTime = request.getRequestTime();

		 if(requestTimePlus == null)
		 {
			requestTimePlus =
				new Long(requestTime.longValue() +
					(Configuration.THIRTY *
						Configuration.SIXTY *
						Configuration.THOUSAND));
		 }

		 if(requestTime >= requestTimePlus
				&& counter != Configuration.ZERO)
		 {
			sessionList.add(requestedItemBuffer);

			requestedItemBuffer = null;

			counter = Configuration.ZERO;

			requestTimePlus = new Long(requestTime.longValue() +
```

```
                                        (Configuration.THIRTY *
                                                        Configuration.SIXTY *
                                                        Configuration.THOUSAND));
                }

                //Set the RequestTimePlus; Only if Counter is set to ZERO
                if(counter == Configuration.ZERO)
                {
                        requestedItemBuffer = new StringBuffer();

                        requestedObject = extractRequestedObject(request.getRequestedURL());

                        //uniqueObjSet.add(requestedObject);
                        uniqueObjSet.put(requestedObject,request.getRequestedItemSize());

                        requestedItemBuffer.
                                                append(requestedObject);

                        counter += Configuration.ONE;
                }
                else
                {
                        requestedObject = extractRequestedObject(request.getRequestedURL());

                        //uniqueObjSet.add(requestedObject);
                        uniqueObjSet.put(requestedObject,request.getRequestedItemSize());

                        requestedItemBuffer.
                                                append("," + requestedObject);
                }

        }

        //Add items to each session to a SessionList
        sessionList.add(requestedItemBuffer);

        requestedItemBuffer = null;

        //Invoke the Garbage Collector
        invokeGC();


        /** If the number of user sessions is at least greater than 2 only then there is point in
         * creating a cluster of popular objects for that domain, otherwise the if number of sessions is
         * just 1 then there will be only one cluster with each object having probability as 1
         * Thus, this would not be an effective clustering pre-fetching scheme */

        //When the above loop is done run the code to find
        //clusters
        if(sessionList.size() >=2)
        {
                clusteredObjects = clusterProcessor.computeCluster(sessionList,uniqueObjSet);
        }

        System.out.println("Number of Unique Objects : " + uniqueObjSet.size());

        printStats();

        return clusteredObjects;
}

/**
```

```
 * Helper to extract Requested Object
 */
private String extractRequestedObject(String URL)
{
	int index = URL.indexOf("//");

	URL = URL.substring(index + Configuration.TWO);

	System.out.println(URL);

	index = URL.indexOf("/");

	if(index != -1)
		URL = URL.substring(index);

	return URL;
}

/**
 * Helper to process each line in the file
 *//*
private Request processLine(String logLine)
{
	Request request = null;

	String[] fields = logLine.split(" ");

	request = createRequestObject(fields);

	return request;
}*/


/**
 * Get the request time
 *
 * @param requestTimeStr
 * @return
 */
private Long processRequestTime(String requestTimeStr)
{
	Double d = new Double(requestTimeStr);

	Long l = new Long((long) (d * Configuration.THOUSAND));

	return l;
}

/**
 * Helper to extract the web object from the URL
 *
 * @param rawURL
 * @return
 */
private String extractWebObject(String rawURL)
{
	return null;
}

/**
 * Helper to process the URL
 */
```

```
private String processURL(String URL)
{
	//URL : http://www.gmail.com/skd/
	URL = URL.trim();

	int index = URL.indexOf("//");
	int index2 = -1;

	//www.gmail.com/skd/
	URL = URL.substring(index + 2);

	index2 = URL.indexOf("/");

	if(index2>-1){
	//www.gmail.com
		URL = URL.substring(0,index2);
	}else
	{
		URL = URL.substring(0);
	}

	if(URL.contains(":"))
	{
		int index3 = URL.indexOf(":");
		URL = URL.substring(0,index3);
	}

	if (uniqueDomainNames.containsKey(URL)) {

		Long count = uniqueDomainNames.get(URL);
		count+=1;
		uniqueDomainNames.put(URL, count);

	}else{

		Long count = new Long(1);
		uniqueDomainNames.put(URL, count);
	}

	return URL;
}

/**
 * Create a Request Object out of the Log File Line
 *
 * @param fields
 * @return
 */
private Request createRequestObject(String[] fields)
{
	Request request = new Request();

	//Long requestTime = new Long(fields[0].toString());
	String requestTimeString = fields[0].trim();

	Long requestTime = new Long(requestTimeString);

	Long requestProcessingTime = new Long(fields[1]);
	String clientIPAddress = fields[2].trim();
	String tcpCode = fields[3].trim();
	Long requestedItemSize = new Long(fields[4].toString());
	String requestedObject = fields[6].trim();
```

```
		//requestParam[7] --> '-'
		String timeOutRedirectionAttribute = fields[8].trim();
		String requestedObjectType = fields[9].trim();

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedURL(requestedObject);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);

		return request;
	}

	/**
	 * Helper to invoke Garbage Collection
	 */
	private void invokeGC()
	{
		try {
			GarbageCollectorHelper.invokeGC();
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}


	private void printStats() throws IOException
	{
		Set<String> uniqueDomain = uniqueDomainNames.keySet();

		/*File outputFile = new File(outputFileName);
		FileWriter fileWriter = null;
		fileWriter = new FileWriter(outputFile);*/

		for (String string : uniqueDomain) {

			System.out.println(string);
			/*fileWriter.write(string + ",Count:"
					+ uniqueDomainNames.get(string)+"\n");*/
		}

		//fileWriter.close();
		System.out.println("Unique Domain Count:" + uniqueDomain.size());

	}

}
package com.sjsu.edu.domainprocessor.helpers;

import java.util.List;
import java.util.Map;
import java.util.Set;


/**
 * Helper to process the Cluster
 *
 * DEPENDS ON :
```

```
 * 1) LCSAlgoImpl
 * 2) ProbablityComponent
 *
 *
 * @author Administrator
 *
 */
public class ClusterProcessor {

	private ProbabilityProcessor probabilityProcessor;
	private SequenceProcessor sequenceProcessor;

	public ClusterProcessor()
	{
		probabilityProcessor = new ProbabilityProcessor();
		sequenceProcessor = new SequenceProcessor();
	}

	/**
	 * Entry point for Cluster Processor
	 */
	//public Map<String,Integer> computeCluster(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)
	public Map<String,Long> computeCluster(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)
	{
		//findSequence(sessionList);

		return processProbablity(sessionList,uniqueObjSet);
	}

	/**
	 * Helper to invoke Sequence Processor
	 *
	 * @param sessionList
	 * @return
	 */
	private List<String> findSequence(List<StringBuffer> sessionList)
	{
		return sequenceProcessor.
			computeSequence(sessionList);
	}

	/**
	 * Helper to process probability
	 *
	 * @param sessionList
	 */
	/*private Map<String,Integer> processProbablity(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)*/
	private Map<String,Long> processProbablity(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)
	{
		return probabilityProcessor.
				computProbabilisticCluster(sessionList,uniqueObjSet);
	}

}
package com.sjsu.edu.domainprocessor.helpers;

import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
```

```
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;

/**
 *
 * Component to compute Probability driven Clusters
 *
 * @author Administrator
 *
 */
public class ProbabilityProcessor {

	//private Map<String,Double> uniqueObjectProbability;
	//private Map<String,Integer> uniqueObjectProbability;
	private Map<String,Long> uniqueObjectProbability;

	public ProbabilityProcessor()
	{
		//uniqueObjectProbability = new LinkedHashMap<String, Double>();
		//uniqueObjectProbability = new LinkedHashMap<String, Integer>();
		uniqueObjectProbability = new LinkedHashMap<String, Long>();
	}

	/**
	 * Entry point for processor for Probability
	 * driven Clustering
	 */
	/*public Map<String,Integer> computProbabilisticCluster
				(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)*/
	public Map<String,Long> computProbabilisticCluster
	(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet)
	{
		return computeCluster(sessionList,uniqueObjSet);
	}

	/**
	 * Helper to Compute the probabilistic Cluster
	 */
	/*private Map<String,Integer> computeCluster(List<StringBuffer> sessionList,
				Map<String,Long> uniqueObjMap)*/
	private Map<String,Long> computeCluster(List<StringBuffer> sessionList,
				Map<String,Long> uniqueObjMap)
	{
		Set<String> uniqueObjSet = uniqueObjMap.keySet();

		//Unique Objects Iterator
		Iterator<String> setIterator =
			uniqueObjSet.iterator();

		Integer uniqueObjectCounter = Configuration.ZERO;

		//Session List Iterator
		Iterator<StringBuffer> sessionListIterator
												= sessionList.iterator();

		Integer noOfSessions = sessionList.size();

		Double numOfSessions = new Double(noOfSessions.toString());

		StringBuffer stringBuffer = null;
```

```
String webObject = null;

String[] sessionObjects = null;

String sessionString = null;

Double theObjectCount = null;

Double probabilityOfObject = null;

int i =0;

//Iterate through the Unique Object List
while(setIterator.hasNext())
{
	webObject = setIterator.next();

	i = i + 1;

	if(i%100==0)
		System.out.println("Unique Object Number :" + i);

	sessionListIterator = sessionList.iterator();

	uniqueObjectCounter = Configuration.ZERO;

	//Try to see if the Unique Object Occurs in how many sessions
	while (sessionListIterator.hasNext())
	{
		//
		stringBuffer =
			(StringBuffer) sessionListIterator.next();

		//Convert the String Buffer Session String to a Session String
		sessionString = stringBuffer.toString();

		//Array of Session's Web Objects
		sessionObjects = sessionString.split(",");

		//Traverse through all of the Session's Objects
		for (String string : sessionObjects)
		{
			if(string.equals(webObject))
			{
				uniqueObjectCounter +=
					Configuration.ONE;
			}
		}
	}

	//The Web Object has been checked for in all sessions.
	//Now compute the probability
	theObjectCount
				= new Double(uniqueObjectCounter.toString());

	probabilityOfObject = theObjectCount/numOfSessions;

	if(probabilityOfObject > 0.30)
	{
		uniqueObjectProbability.put(webObject, uniqueObjMap.get(webObject));
	}
```

```
                        stringBuffer = null;
                        sessionString = null;
                        sessionObjects = null;
                        probabilityOfObject = null;
                        theObjectCount = null;

                        //Invoke the GC
                        invokeGC();
                }

                return uniqueObjectProbability;
        }

        private void invokeGC()
        {
                try {
                        GarbageCollectorHelper.invokeGC();
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

}
package com.sjsu.edu.edge;

import com.sjsu.edu.node.Node;

public class Edge {

        private Integer count;
        private Node startNode;
        private Node endNode;
        private Double confidenceMeasure;

        public Double getConfidenceMeasure() {
                return confidenceMeasure;
        }

        public void setConfidenceMeasure(Double confidenceMeasure) {
                this.confidenceMeasure = confidenceMeasure;
        }

        public Node getStartNode() {
                return startNode;
        }

        public void setStartNode(Node startNode) {
                this.startNode = startNode;
        }

        public Node getEndNode() {
                return endNode;
        }

        public void setEndNode(Node endNode) {
                this.endNode = endNode;
        }


        public Integer getCount() {
```

```
		return count;
	}

	public void setCount(Integer count) {
		this.count = count;
	}

	public void incrementEdgeCount()
	{
		this.count +=1;
	}


}

package com.sjsu.edu.edge;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.Request;

public class RequestProcessor {

	public static Request processRequest(String logLine)
	{

		Request request = null;

		String[] fields = logLine.split(" ");

		request = createRequestObject(fields);

		return request;

	}

	private static Request createRequestObject(String[] fields)
	{
		Request request = new Request();

		//Long requestTime = new Long(fields[0].toString());
		String requestTimeString = fields[0].trim();

		int indexOfDot = fields[0].indexOf(".");
		requestTimeString = requestTimeString.substring(0, indexOfDot).trim();

		Long requestTime = processRequestTime(requestTimeString);

		Long requestProcessingTime = new Long(fields[1]);
		String clientIPAddress = fields[2].trim();
		String tcpCode = fields[3].trim();
		Long requestedItemSize = new Long(fields[4].toString());
		String requestedObject = fields[6].trim();

		String URL = processURL(requestedObject);

		//requestParam[7] --> '-'
		String timeOutRedirectionAttribute = fields[8].trim();
		String requestedObjectType = fields[9].trim();

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
```

```
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedURL(URL);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);

		return request;
}

/**
 * Get the request time
 *
 * @param requestTimeStr
 * @return
 */
private static Long processRequestTime(String requestTimeStr)
{
		Double d = new Double(requestTimeStr);

		Long l = new Long((long) (d * Configuration.THOUSAND));

		return l;
}

/**
 * Helper to process the URL
 */
private static String processURL(String URL)
{
		//URL : http://www.gmail.com/skd/
		URL = URL.trim();

		int index = URL.indexOf("//");
		int index2 = -1;

		//www.gmail.com/skd/
		URL = URL.substring(index + Configuration.TWO);

		//return URL;

		index2 = URL.indexOf("/");

		if(index2>-1)
		{
			URL =
				URL.substring(Configuration.ZERO,index2);
		}
		else
		{
			URL =
				URL.substring(Configuration.ZERO);
		}

		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");

			URL =
				URL.substring(Configuration.ZERO,index3);
		}

/*		if (uniqueDomainNames.containsKey(URL)) {
```

```
                Long count = uniqueDomainNames.get(URL);
                count+=1;
                uniqueDomainNames.put(URL, count);

        }else{

                Long count = new Long(CacheConstants.ONE);
                uniqueDomainNames.put(URL, count);
        }*/



        return URL;
}

public static Request processRequest2(String logLine)
{

        Request request = null;

        String[] fields = logLine.split(" ");

        request = createRequestObject2(fields);

        return request;

}

private static Request createRequestObject2(String[] fields)
{
        Request request = new Request();

        //Long requestTime = new Long(fields[0].toString());
        String requestTimeString = fields[0].trim();

        Long requestTime = processRequestTime(requestTimeString);

        Long requestProcessingTime = new Long(fields[1]);
        String clientIPAddress = fields[2].trim();
        String tcpCode = fields[3].trim();
        Long requestedItemSize = new Long(fields[4].toString());
        String requestedObject = fields[6].trim();

        String URL = processURL2(requestedObject);


        //requestParam[7] --> '-'
        String timeOutRedirectionAttribute = fields[8].trim();
        String requestedObjectType = fields[9].trim();

        request.setRequestTime(requestTime);
        request.setRequestProcessingTime(requestProcessingTime);
        request.setClientIPAddress(clientIPAddress);
        request.setTcpCode(tcpCode);
        request.setRequestedItemSize(requestedItemSize);
        request.setRequestedURL(URL);
        request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
        request.setRequestedObjectType(requestedObjectType);

        return request;
}
```

```
	/**
	 * Helper to process the URL
	 */
	private static String processURL2(String URL)
	{
		//URL : http://www.gmail.com/skd/
		URL = URL.trim();

		int index = URL.indexOf("//");
		int index2 = -1;

		//www.gmail.com/skd/
		URL = URL.substring(index + Configuration.TWO);

		return URL;
	}
}
package com.sjsu.edu.filewriter;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.node.Node;

/**
 *
 * @author Administrator
 *
 */
public class FileProcessor
{
	String baseDir;
	FileWriter fstream;
	BufferedWriter out;

	public FileProcessor(String baseDirectory)
	{
		baseDir = baseDirectory;
	}

	/**
	 *
	 * Method writes Intersite Cluster to File
	 *
	 * @param clusters
	 * @param clusterNum
	 * @throws IOException
	 */
	public void writeInterSiteClusterToFile(Set<Node> clusters,int clusterNum,String clientIPAddress) throws
IOException
	{
		String topDirValue = baseDir + "\\" + clientIPAddress;

		String fileName = topDirValue + "\\cluster" + clusterNum + ".txt";

		File topDir = new File(topDirValue);
```

```
                if(!topDir.exists())
                        topDir.mkdir();

                File filename = new File(fileName);

                fstream = new FileWriter(filename);
        out = new BufferedWriter(fstream);

        writeIntrasiteNodes(clusters,clientIPAddress);

        }


        /**
         * Method writes Intrasite Cluster to File
         * @throws IOException
         */
        public void writeIntraSiteClusterToFile(String clientIP, Map<Node,Map<String,Long>> intraSiteCluster) throws
IOException
        {

                String currentURL = null;
                String intraSiteFileDir = null;
                File intraSiteDir = null;
                Long size = null;
                String completeURL = null;
                File intraSiteCurrentURLPath = null;
                Set<String> uRLSet = null;
                String completeIntraSiteFilePath = null;
                Map<String,Long> intraSiteNodes = null;
                String topDirValue = baseDir + "\\" + clientIP;
                Map<String,Long> intraSiteClusterForURL = null;

                File topDir = new File(topDirValue);

                if(!topDir.exists())
                        topDir.mkdir();

                Set<Node> interSiteNodes = intraSiteCluster.keySet();

                for (Node node : interSiteNodes)
                {
                        currentURL = node.getURL();

                        intraSiteFileDir = topDirValue + "\\" + currentURL;

                        intraSiteClusterForURL = intraSiteCluster.get(node);

                        intraSiteDir = new File(intraSiteFileDir);

                        if(!intraSiteDir.exists())
                                intraSiteDir.mkdir();

                        completeIntraSiteFilePath = intraSiteFileDir + "\\" + currentURL + ".txt";

                        intraSiteCurrentURLPath = new File(completeIntraSiteFilePath);

                        fstream = new FileWriter(intraSiteCurrentURLPath);
                out = new BufferedWriter(fstream);

                intraSiteNodes = intraSiteCluster.get(node);
```

```
                uRLSet = intraSiteNodes.keySet();

                for (String currentIntraSiteObject : uRLSet)
                {
                        size = intraSiteNodes.get(currentIntraSiteObject);

                        completeURL = currentURL + currentIntraSiteObject;

                        out.write(completeURL + "," + size + "\n");
                                }

                out.close();
                        }

        }

        private void writeIntrasiteNodes(Set<Node> clusters,String cip) throws IOException
        {

                for (Node node : clusters)
                {
                        if(cip.equals(Configuration.dp) && node.getURL().contains("au.download"))
                                continue;

                        out.write(node.getURL() + "\n");
                }

                out.close();
        }

}
package com.sjsu.edu.graphintersite;

import java.util.Collection;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.edge.Edge;
import com.sjsu.edu.node.Node;

import edu.uci.ics.jung.graph.DirectedGraph;
import edu.uci.ics.jung.graph.DirectedSparseGraph;
import edu.uci.ics.jung.graph.event.GraphEvent.Vertex;
import edu.uci.ics.jung.graph.util.EdgeType;

/**
 *
 * GRAPH API -
 *
 * 1) Create a Graph
 * 2) Edit Graph by Confidence value
 * 3) Edit Graph by Support value
 * 4) Compute the cluster for inter site clustering
 *
```

```
 * @author Administrator
 *
 */

public class InterSiteGraph
{
	/**
	 *
	 * A map to store the URL - Node in the graph mapping
	 *
	 */
	private Map<String,Node> nodeSet;

	/**
	 * Instance of a directed graph that actually holds the graph objects
	 */
	private DirectedGraph<Node, Edge> graphInterSite;
	private Node prevNode;

	public InterSiteGraph()
	{
		prevNode = null;
		nodeSet = new HashMap<String, Node>();
		graphInterSite = new DirectedSparseGraph<Node, Edge>();
	}

	/**
	 * Returns all the Edges in the Graph
	 *
	 * @return
	 */
	public Collection<Edge> getEdges()
	{
		Collection<Edge> edges = graphInterSite.getEdges();

		return edges;
	}

	/**
	 * Returns all the Nodes/Vertices in the Graph
	 *
	 * @return
	 */
	public Collection<Node> getNodes()
	{
		Collection<Node> nodes = graphInterSite.getVertices();

		return nodes;
	}

	/**
	 *
	 * Adds node to the graph
	 *
	 * @param nextNode
	 */
	public void addNode(Node nextNode)
	{
		Integer edgeCount = null;
		String URL = null;
		Edge edge = null;
		Edge e = null;
```

```
if(prevNode!=null) //Graph is Already created
{
	URL = nextNode.getURL().trim();

	if(nodeSet.containsKey(URL))
	{
		nextNode = nodeSet.get(URL);

		edge = graphInterSite.findEdge(prevNode,nextNode);

		//Edge Found
		if(edge!= null)
		{
			edgeCount = edge.getCount();
			edgeCount +=1;

			edge.setCount(edgeCount);

			e = graphInterSite.findEdge(prevNode, nextNode);

			//Update the next Node's Popularity
			updateNodePopularity(nextNode);

			prevNode = nextNode;
		}
		else // Add New Edge
		{
			e = new Edge();

			e.setCount((Integer) 1);

			//Set the Start Node & End Node
			e.setStartNode(prevNode);
			e.setEndNode(nextNode);

			graphInterSite.addEdge(e, prevNode, nextNode);

			//Update the next Node's Popularity
			updateNodePopularity(nextNode);

			//Set the Previous Node value to this Node
			prevNode = nextNode;
		}
	}
	else
	{
		//Add the nextNode to the Graph
		graphInterSite.addVertex(nextNode);

		//Add the Node to the Node Set
		nodeSet.put(nextNode.getURL(), nextNode);

		//Create a new Edge
		edge = new Edge();

		edge.setCount((Integer)1);

		//Set the Start Node and the End Node
		edge.setStartNode(prevNode);
		edge.setEndNode(nextNode);
```

```
                                graphInterSite.addEdge(edge, prevNode, nextNode);

                                //Update the next Node's Popularity
                                updateNodePopularity(nextNode);

                                //Set the prevNode as nextNode
                                prevNode = nextNode;
                        }
                }
                else //Start of the Graph
                {
                        //Add the Head Node to the Graph
                        nodeSet.put(nextNode.getURL(), nextNode);

                        //Set the Value of the previous Node to this node
                        prevNode = nextNode;

                        //Add the Node as a vertex to the Graph
                        graphInterSite.addVertex(nextNode);
                }
}


/**
 * Method to update the popularity of the node
 *
 * @param node
 */
private void updateNodePopularity(Node node)
{
        //Get the Node popularity
        Integer nodePopularity = node.getPopularity();

        if(nodePopularity == null)
        {
                node.setPopularity((Integer)1);
        }
        else //Increment it by one
        {
                nodePopularity +=1;
                node.setPopularity(nodePopularity);
        }
}

/**
 *
 * Procedure to cut with confidence
 *
 * @param confidenceThreshold
 */
public void cutWithConfidence(Double confidenceThreshold)
{
        //Set of Edges for the Graph
        Collection<Edge> edgeSet = graphInterSite.getEdges();

        //Directed Edge
        Edge edge = null;

        //Directed Edge Start Node
        Node edgeStartNode = null;

        //Directed Edge popularity
```

```
		Integer edgeCount = null;

		//Confidence Measure
		Double confidenceMeasure = null;

		//Start Node Popularity
		Integer edgeStartNodePopularity = null;

		//Edges to be discarded from the graph
		List<Edge> edgeToBeRemoved = new LinkedList<Edge>();

		for (Iterator<Edge> iterator = edgeSet.iterator(); iterator.hasNext();)
		{
			//Get the edge
			edge = (Edge) iterator.next();

			//Get the edge count for the edge
			edgeCount = edge.getCount();

			//Get the start node for the directed edge
			edgeStartNode = edge.getStartNode();

			Node endNode = edge.getEndNode();

			//Get the popularity of the start node
			edgeStartNodePopularity = edgeStartNode.getPopularity();

			if(edgeStartNodePopularity!=null) //Get the confidence, conditional probability f(A,B)/f(A)
			{
				confidenceMeasure = (new Double(edgeCount.toString())/ new
Double(edgeStartNodePopularity.toString()));
				edge.setConfidenceMeasure(confidenceMeasure);

				//Add the edge to list of edges to be removed if the edge confidence value
				//is less than confidence threshold value
				//if(confidenceMeasure >= confidenceThreshold)
				if(!((confidenceMeasure > confidenceThreshold) || (confidenceMeasure ==
confidenceThreshold)))
					edgeToBeRemoved.add(edge);
			}
			else
			{
				edgeToBeRemoved.add(edge);
			}
		}

		if(edgeToBeRemoved.size()>0)
		{
			//Remove all the edges that do not meet minimum confidence threshold
			for (Edge discardEdge : edgeToBeRemoved)
			{
				graphInterSite.removeEdge(discardEdge);
			}
		}
	}

	/**
	 *
	 * Procedure to cut with Support
	 *
	 * @param supportThreshold
	 */
```

```
public void cutWithSupport(Integer supportThreshold)
{
	Collection<Edge> edgeSet = graphInterSite.getEdges();

	//Edges to be Removed
	List<Edge> edgeToBeRemoved = new LinkedList<Edge>();

	//Individual Edge
	Edge edge = null;

	//Edge count
	Integer edgeCount = null;

	//EdgeSet iterator
	for (Iterator<Edge> iterator = edgeSet.iterator(); iterator.hasNext();)
	{
		//Get the edge
		edge = (Edge) iterator.next();

		//Get the edgeCount
		edgeCount = edge.getCount();

		//If the edgeCount is greater than the support threshold
		if(!((edgeCount > supportThreshold) || (edgeCount == supportThreshold)))
		{
			edgeToBeRemoved.add(edge);
		}
	}

	//Discard all the edges that do not meet support threshold
	if(edgeToBeRemoved.size()>0)
	{
		for (Edge discardEdge : edgeToBeRemoved)
		{
			graphInterSite.removeEdge(discardEdge);
		}
	}
}

/**
 * Does a breath first Traversal for the remaining of
 * the Graph
 *
 */
public List<Set<Node>> breadthFirstTraversal()
{
	Collection<Node> verticesOfGraph = graphInterSite.getVertices();

	LinkedList<Node> queue = new LinkedList<Node>();

	Set<Node> clusterOfNodes = new HashSet<Node>();
	List<Set<Node>> listOfClusters = null;

	boolean entryNode = true;

	/*System.out.println(" +++++ ***** **** +++++");
	System.out.println(" BFS ");
	System.out.println(" +++++ ***** **** +++++");*/

	Node n = getUnvisitedNode();

	Node neighbhorNode = null;
```

```
		if(n== null)
			return listOfClusters;

		n.setVisited(true);
		clusterOfNodes.add(n);
		queue.push(n);
		//System.out.println(n.getURL());

		if(verticesOfGraph!=null)
		{
			listOfClusters = new LinkedList<Set<Node>>();

			while(getUnvisitedNode()!=null)
			{
				if(entryNode)
					entryNode = false;
				else
				{
					Node e = getUnvisitedNode();
					queue.push(e);
				}

				while(!queue.isEmpty())
				{
					Node node = queue.remove();
					node.setVisited(true);
					Node child = null;

					clusterOfNodes.add(node);

					while((child = getUnvisitedChildNode(node))!= null)
					{
						child.setVisited(true);
						//boolean added = clusterOfNodes.add(child);
						clusterOfNodes.add(child);
						queue.add(child);
					}
				}

				listOfClusters.add(clusterOfNodes);
				clusterOfNodes = new HashSet<Node>();
			}
		}

		/*System.out.println(" +++++ ***** **** +++++");
		System.out.println(" BFS ENDS ");
		System.out.println(" +++++ ***** **** +++++");*/

		return listOfClusters;
}


/**
 * Helper to remove nodes from the graph who do not have
 * any neighbors or edge pointing from the node is pointing
 * to the node itself and there are no other nodes
 *
 */
public void preprocessGraph()
{
		Collection<Node> nodesOfGraph = graphInterSite.getVertices();
```

```
		Set<Node> nodesToBeRemoved = new HashSet<Node>();

		for (Node node : nodesOfGraph)
		{
			if(graphInterSite.getNeighbors(node).size()== 0)
			{
				nodesToBeRemoved.add(node);
			}
			else if(graphInterSite.getNeighborCount(node)== 1)
			{
				Collection<Node> neighbors = graphInterSite.getNeighbors(node);

				for (Node node2 : neighbors)
				{
					if(node2 == node)
					{
						nodesToBeRemoved.add(node2);
					}
				}
			}
			else
			{
				continue;
			}
		}

		if(nodesToBeRemoved.size() > 0)
		{
			for (Node node : nodesToBeRemoved)
			{
				graphInterSite.removeVertex(node);
			}
		}
}

/**
 *
 * Helper to get Unvisited Neighbor Node for the child Node
 *
 * @param childNode
 * @return
 */
private Node getUnvisitedChildNode(Node childNode)
{
		Collection<Node> neighbourNodes = graphInterSite.getNeighbors(childNode);

		for (Node node : neighbourNodes)
		{
			if(!node.getVisited())
			{
				//node.setVisited(true);
				return node;
			}
		}

		return null;
}


/**
 *
```

```
 * Process Graph with confidence threshold and support threshold and
 * do a graph traversal using BFS to obtain a cluster of nodes and
 * return the same
 *
 * @param confidenceThreshold
 * @param supportThreshold
 * @return
 */
public List<Set<Node>> processGraph(Double confidenceThreshold,Integer supportThreshold)
{
	List<Set<Node>> clusterOfNodes = null;

	System.out.println("--- ---- -- CUT WITH CONFIDENCE -- --- ---");

	cutWithConfidence(confidenceThreshold);

	System.out.println("--- ---- -- CUT WITH SUPPORT -- --- ---");

	cutWithSupport(supportThreshold);

	//Remove all vertices(Web page nodes) from the graph, that have no neighbors,
	// and edges out of the vertex(Web page Node) point to itself
	preprocessGraph();

	System.out.println("--- ---- -- BFS TRAVERSAL -- --- --- ");
	clusterOfNodes = breadthFirstTraversal();

	return clusterOfNodes;
}

/**
 *
 * Helper to get Unvisited Neighbor Node for the child Node
 *
 * @param childNode
 * @return
 */
private Node getUnvisitedNode(Node node)
{
	Collection<Node> neighbourNodes = graphInterSite.getNeighbors(node);

	for (Node node1 : neighbourNodes)
	{
		if(!node1.getVisited())
		{
			node1.setVisited(true);
			return node;
		}
	}

	return null;
}

private Node getUnvisitedNode()
{
	Collection<Node> neighbourNodes = graphInterSite.getVertices();

	for (Node node : neighbourNodes)
	{
		if(!node.getVisited())
		{
			//node.setVisited(true);
```

```
                                        return node;
                                }
                        }

                        return null;
                }

                public void processRequest(Request request)
                {
                        if(prevNode!=null)
                        {
                                if(!prevNode.getURL().equals(request.getRequestedURL()))
                                {
                                        Node node = new Node();

                                        node.setTimestamp(request.getRequestTime());
                                        node.setURL(request.getRequestedURL());
                                        node.setSizeOfObject(request.getRequestedItemSize());

                                        addNode(node);
                                }
                        }
                        else
                        {
                                Node node = new Node();

                                node.setTimestamp(request.getRequestTime());
                                node.setURL(request.getRequestedURL());
                                node.setSizeOfObject(request.getRequestedItemSize());

                                addNode(node);
                        }
                }

}

package com.sjsu.edu.graphlogfilereader;

import java.io.BufferedInputStream;


import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.edge.RequestProcessor;
import com.sjsu.edu.filewriter.FileProcessor;
import com.sjsu.edu.graphintersite.InterSiteGraph;
import com.sjsu.edu.intrasiteclustering.IntrasiteClusterHelper;
import com.sjsu.edu.node.Node;

public class GraphLogFileReader
{
        private String dp;
        private String baseDir;
        private Double interSiteConfidenceThreshold;
```

```
	private Double intraSiteConfidenceThreshold;
	private Integer interSiteSupportThreshold;
	private Integer intraSiteSupportThreshold;
	private String baseDirectoryForInterSite;
	private String baseDirectoryForIntraSite;


	/**
	 *
	 * @param baseDirectory
	 * @param interSiteConfidenceThreshold
	 * @param intraSiteConfidenceThreshold
	 * @param interSiteSupportThreshold
	 * @param intraSiteSupportThreshold
	 * @param baseDirectoryForInterSite
	 * @param baseDirectoryForIntraSite
	 */
	public GraphLogFileReader(String baseDirectory,Double interSiteConfidenceThreshold,
				Double intraSiteConfidenceThreshold,Integer interSiteSupportThreshold,
				Integer intraSiteSupportThreshold,String baseDirectoryForInterSite,String
baseDirectoryForIntraSite)
	{
		baseDir = baseDirectory;
		this.interSiteConfidenceThreshold = interSiteConfidenceThreshold;
		this.intraSiteConfidenceThreshold = intraSiteConfidenceThreshold;
		this.interSiteSupportThreshold = interSiteSupportThreshold;
		this.intraSiteSupportThreshold = intraSiteSupportThreshold;
		this.baseDirectoryForInterSite = baseDirectoryForInterSite;
		this.baseDirectoryForIntraSite = baseDirectoryForIntraSite;

	}

	public void processLogFile() throws IOException, InterruptedException
	{
		String[] filesInDir = null;
		InterSiteGraph clusterGraph = null;
		File f = new File(baseDir);
		dp = Configuration.dp;
		File clientIPFile = null;
		String fileDir = null;
		String filePath = null;
		String logLine = null;
		List<Set<Node>> interSiteNodes = null;
		boolean clientIPstored = true;
		String cipad = null;
		FileInputStream fileInputStream = null;
		BufferedInputStream bufferedInputStream = null;
		DataInputStream dataInputStream = null;
		Request request = null;
		Double confidenceThreshold = null;
		Integer supportThreshold = null;
		int clusterCounter = 0;
		FileProcessor fileWriter = null;
		String baseDirForInterSite = null;
		IntrasiteClusterHelper intraSiteClusterHelper = null;
		Map<Node,Map<String,Long>> intraSiteCluster = null;
		Map<String,Long> intraSiteMap = null;
		Set<String> intraSiteURLs = null;
		Set<Node> intraSiteNodes = null;

		baseDirForInterSite = "D:\\ClusteredWebObjects\\IntersiteClusters";
```

```
filesInDir = f.list();
clusterGraph = new InterSiteGraph();

//For each Client IP
for (String string1 : filesInDir)
{

	//Demo for Inter Site Clustering
	//String string = "D:\\ClientIPCustomizedLogFile\\98.241.74.53\\98.241.74.53.txt";
	//String string = "D:\\ClientIPCustomizedLogFile\\98.241.74.53\\98.241.743";

	//Demo for Intra Site Clustering
	//String string =
"D:\\CustomizedLogFile\\3.217.149.31\\www.chesshere.com\\www.chesshere.com.txt";

	fileDir = string1 + "\\" + string1 + ".txt";

	filePath = baseDir + "\\" + fileDir;

	clientIPFile = new File(filePath);
	//clientIPFile = new File(string);

	fileInputStream =
		new FileInputStream(clientIPFile);

	bufferedInputStream =
		new BufferedInputStream(fileInputStream);

	dataInputStream =
		new DataInputStream(bufferedInputStream);

	//while(dataInputStream.available()!= CacheConstants.ZERO)
	while(dataInputStream.available()!= Configuration.ZERO)
	{
		logLine = dataInputStream.readLine();

		request = RequestProcessor.processRequest(logLine);

		if(clientIPstored)
		{
			cipad = request.getcip();

			clientIPstored = false;

			checkcp(cipad);

		}

		clusterGraph.processRequest(request);
	}

  clientIPstored = true;

  System.out.println("PROCESSING FOR CLIENT IP: " + cipad);

  confidenceThreshold = interSiteConfidenceThreshold;
  supportThreshold = interSiteSupportThreshold;

  //Intersite Nodes for a particular Client IP
  interSiteNodes =   clusterGraph.processGraph(confidenceThreshold, supportThreshold);
  //interSiteNodes = clusterGraph.processGraph(0.2, 2);
```

```
if(interSiteNodes!=null && interSiteNodes.size() >0 )
{
	//Code for IntraSite Objects -- Ends
	System.out.println("Inter site Cluster Begins for Client IP " + cipad);
	System.out.println(" ------ Cluster Begins -------");


	for (Set<Node> set : interSiteNodes)
	{
		if(set.size() > 1)
		{
			intraSiteClusterHelper = new IntrasiteClusterHelper();

			intraSiteCluster =
				intraSiteClusterHelper.processIntraSiteNodes(set,
cipad,intraSiteConfidenceThreshold,intraSiteSupportThreshold);

			intraSiteNodes = intraSiteCluster.keySet();

			//try{
			FileProcessor fileWriterForCluster = new
FileProcessor(Configuration.cIPIntrasiteClusterDir);
			fileWriterForCluster.writeIntraSiteClusterToFile(cipad,intraSiteCluster);
			/*}
			catch(Exception ex)
			{
				System.out.println(ex.toString());
			}*/

			for (Node node : intraSiteNodes)
			{
				System.out.println("----------  URL  ----------");
				System.out.println(node.getURL());

				intraSiteMap = intraSiteCluster.get(node);

				System.out.println("---------- INTRASITE  URL  ----------");

				intraSiteURLs = intraSiteMap.keySet();

				for (String string : intraSiteURLs)
				{
					System.out.println("URL: " + string + ", Size :" +
intraSiteMap.get(string));
				}

				System.out.println("---------- INTRASITE  URL ENDS ----------
---");


				System.out.println("----------  URL ENDS ----------");
			}

			clusterCounter=clusterCounter+1;

			fileWriter = new FileProcessor(baseDirForInterSite);
			//fileWriter.writeInterSiteClusterToFile(set, clusterCounter,
currentClientIPAddress);
			fileWriter.writeInterSiteClusterToFile(intraSiteNodes, clusterCounter, cipad);
			System.out.println(" ------ Cluster ENDS -------");
		}
	}
```

```
                        clientIPstored = true;
                        clusterCounter =0;

                    }//
                    else
                    {
                        System.out.println("Intersite Nodes for Client IP :"  + cipad + " is null");
                    }

                    System.out.println();
                    Thread.sleep(50);
                    System.gc();
                    Thread.sleep(50);
                }

                System.out.println("------------------- COMPLETION OF THE CLUSTERING PROCEDURES ------------ ");
        }


        private void checkcp(String cipad)
        {
                if(cipad.equals(dp))
                {
                        interSiteSupportThreshold = Configuration.interSitSupportThreshold;
                }
                else
                {
                        interSiteSupportThreshold = Configuration.interSiteSupportThreshold;
                }
        }
}
package com.sjsu.edu.graphtesting;

import java.io.IOException;
import java.util.Collection;
import java.util.List;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.edge.Edge;
import com.sjsu.edu.graphintersite.InterSiteGraph;
import com.sjsu.edu.graphlogfilereader.GraphLogFileReader;
import com.sjsu.edu.node.Node;

public class GraphTester {

        public static void main(String[] args) throws IOException, InterruptedException
        {
                //GraphTester gTester = new GraphTester();

                //gTester.createAndProcessDefaultGraph(gTester);

                //String baseDirectory = "D:\\ClientIPCustomizedLogFile\\";

                Double interSiteConfidenceThreshold = Configuration.interSiteConfidenceThreshold;
                Double intraSiteConfidenceThreshold = Configuration.intraSiteConfidenceThreshold;
                Integer interSiteSupportThreshold = Configuration.interSiteSupportThreshold;
                Integer intraSiteSupportThreshold = Configuration.intraSiteSupportThreshold;
                String baseDirectory = Configuration.cIPLogFileBaseDir;
                String baseDirectoryForInterSite = Configuration.cIPIntersiteClusterDir;
                String baseDirectoryForIntraSite = Configuration.cIPIntrasiteClusterDir;
```

```
		InterSiteGraph g = new InterSiteGraph();

		GraphLogFileReader graphLogFileReader
		= new GraphLogFileReader(baseDirectory,interSiteConfidenceThreshold,
					intraSiteConfidenceThreshold,interSiteSupportThreshold,
					intraSiteSupportThreshold,baseDirectoryForInterSite,baseDirectoryForIntraSite);

		graphLogFileReader.processLogFile();

}

/**
 *
 * Creates and processes a default graph for processing
 * and obtains a cluster of objects from the graph
 *
 * @param gTester
 */
public void createAndProcessDefaultGraph(GraphTester gTester)
{

		InterSiteGraph g = new InterSiteGraph();

		//Create Default Graph
		gTester.createDefaultGraph(g);

		Collection<Edge> edges = g.getEdges();

		//Cut with Support
		g.cutWithSupport((Integer) 3);

		System.out.println(" +++++ ***** **** +++++");
		System.out.println("AFTER CUT WITH SUPPORT");
		System.out.println(" +++++ ***** **** +++++");

		for (Edge edge : edges)
		{
			System.out.println("Edge: " + edge.getStartNode().getURL() + "-->"
					+ edge.getEndNode().getURL() + ", Edge Count:" + edge.getCount());

			System.out.println();
		}

		g.cutWithConfidence((Double) 0.2);

		System.out.println(" +++++ ***** **** +++++");
		System.out.println("AFTER CUT WITH CONFIDENCE");
		System.out.println(" +++++ ***** **** +++++");

		edges = g.getEdges();

		for (Edge edge : edges) {

			System.out.println("Edge: " + edge.getStartNode().getURL() + "-->"
					+ edge.getEndNode().getURL() + ", Edge Count:" + edge.getCount()
					+ ",Confidence Measure: " + edge.getConfidenceMeasure());

			System.out.println();
		}

		g.preprocessGraph();
```

```
		/*Collection<Node> nodes = g.getNodes();

		for (Node node : nodes) {

			System.out.println("Node URL: " + node.getURL() + ", Node Popularity: " +
					node.getPopularity());

			System.out.println();

		}*/

		Collection<Node> nodes = g.getNodes();

		System.out.println(" +++++ ***** **** +++++");
		System.out.println("AFTER GRAPH PRE-PROCESSING");
		System.out.println(" +++++ ***** **** +++++");

		for (Node node : nodes)
		{
			System.out.println(node.getURL());
		}

		int i =1;

		List<Set<Node>> listOfCluster = g.breadthFirstTraversal();

		System.out.println();
		System.out.println("**** ++++ PRINTING CLUSTERS ++++ *****");
		System.out.println();

		for (Set<Node> set : listOfCluster) {

			System.out.println("++++ ***** Begin Cluster " + i + " **** ++++");

			for (Node node : set)
			{
				System.out.println(node.getURL());
			}

			System.out.println("++++ ***** End Cluster " + i + " **** ++++");
			System.out.println();
			i+=1;


		}
}

/**
 * Create a Default Graph for testing
 *
 * @param g
 */
public void createDefaultGraph(InterSiteGraph g)
{
		Node nextNode = new Node();
		nextNode.setURL("GOOGLE");
		//URL1/Page1
		//nextNode.setURL("www.google.com");
		g.addNode(nextNode);

		nextNode = new Node();
		nextNode.setURL("FACEBOOK");
```

```
//URL2/Page1
//nextNode.setURL("www.facebook.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("GOOGLE");
//URL1/Page2
//nextNode.setURL("www.google.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("FACEBOOK");
//URL2/Page2
//nextNode.setURL("www.facebook.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("GOOGLE");
//URL1//Page1
//nextNode.setURL("www.google.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("HOTMAIL");
//URL3/Page1
//nextNode.setURL("www.hotmail.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("NDTV");
//URL4/Page1
//nextNode.setURL("www.ndtv.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("IBNLIVE");
//URL5/Page1
//nextNode.setURL("www.ibnlive.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("TECRUNCH");
//URL6/Page1
//nextNode.setURL("www.ibnlive.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("IBNLIVE");
//URL5/Page2
//nextNode.setURL("www.ibnlive.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("TECRUNCH");
//URL6/Page2
//nextNode.setURL("www.ibnlive.com");
g.addNode(nextNode);

nextNode = new Node();
nextNode.setURL("IBNLIVE");
//URL5/Page2
//nextNode.setURL("www.ibnlive.com");
```

```
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("TECRUNCH");
                //URL6/Page1
                //nextNode.setURL("www.ibnlive.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("IBNLIVE");
                //URL5/Page2
                //nextNode.setURL("www.ibnlive.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("GOOGLE");
                //nextNode.setURL("www.google.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("FACEBOOK");
                //nextNode.setURL("www.facebook.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("HOTMAIL");
                //nextNode.setURL("www.hotmail.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("FACEBOOK");
                //nextNode.setURL("www.facebook.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("IBNLIVE");
                //nextNode.setURL("www.ibnlive.com");
                g.addNode(nextNode);

                nextNode = new Node();
                nextNode.setURL("FACEBOOK");
                //nextNode.setURL("www.facebook.com");
                g.addNode(nextNode);



        }

}
package com.sjsu.edu.intrasiteclustering;



import java.io.IOException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.domainprocesso.IntrasiteLogProcessor;
import com.sjsu.edu.node.Node;
```

```
public class IntrasiteClusterHelper {

	//private String baseDir;
	//private Set<Node> noIntraSiteClusterNodes;


	public IntrasiteClusterHelper()
	{
		//baseDir = dir; -- Commented Out (String dir) from Constructor
		//noIntraSiteClusterNodes = new HashSet<Node>();
	}

	/**
	 * Helper that takes in set of Nodes and Client IP as input and
	 * outputs intra site cluster for every unique URL(obtained) from Node
	 *
	 * @param intersiteNodes: InterSite Nodes for the Cluster
	 * @param clientIP: Client IP for which inter site nodes belong
	 * @return
	 * @throws IOException
	 */
	public Map<Node,Map<String,Long>> processIntraSiteNodes
		(Set<Node> intersiteNodes,String clientIP,Double confidenceThreshold, Integer supportThreshold) throws
IOException
	{
		//Intrasite Log processor
		IntrasiteLogProcessor intraSiteLogProccessor = null;

		//Current URL
		String currentURLDomain = null;

		//Intra site cluster for a particular URL
		Map<String,Long> intraSiteCluster = null;

		//A Map of a URL and intrasite URL's for the URL
		Map<Node,Map<String,Long>> intraSiteClusterOfObjects = null;

		Set<Node> ignoreNodes = new HashSet<Node>();

		//A Hash Map of URL, Intrasite Cluster of Objects for that URL
		intraSiteClusterOfObjects = new HashMap<Node, Map<String,Long>>();

		for (Node node : intersiteNodes)
		{
			//Initializes the Log File
			intraSiteLogProccessor = new IntrasiteLogProcessor();

			//Get the Current Intersite Node of the Intersite Cluseter
			currentURLDomain = node.getURL();

			//Sets the parameters of the Log file
			intraSiteLogProccessor.setParameters(clientIP,
currentURLDomain,confidenceThreshold,supportThreshold);

			System.out.println("Current URL Name: " + currentURLDomain);

			//Gets the intrasite Cluster of Objects
			intraSiteCluster = intraSiteLogProccessor.processURLLogFile();

			//If there are no objects in the intrasite Cluster
			if(intraSiteCluster == null || intraSiteCluster.size() < 1)
			{
```

```
                                        //noIntraSiteClusterNodes.add(currentURLDomain);
                                        //intraSiteClusterOfObjects.put(node, intraSiteCluster);
                                        ignoreNodes.add(node);
                                }
                                else
                                {
                                        intraSiteClusterOfObjects.put(node, intraSiteCluster);
                                }

                        }

                        return intraSiteClusterOfObjects;
                }
}
package com.sjsu.edu.node;

public class Node {

        private String webObject;
        private Integer popularity;
        private Long sizeOfObject;
        private Boolean visited;
        private String URL;
        private Long timestamp;

        public Long getTimestamp() {
                return timestamp;
        }


        public void setTimestamp(Long timestamp) {
                this.timestamp = timestamp;
        }


        public String getURL() {
                return URL;
        }


        public void setURL(String uRL) {
                URL = uRL;
        }
        public Node()
        {
                visited = new Boolean(false);
        }
        public Boolean getVisited() {
                return visited;
        }
        public void setVisited(Boolean visited) {
                this.visited = visited;
        }
        public Long getSizeOfObject()
        {
                return sizeOfObject;
        }

        public void setSizeOfObject(Long sizeOfObject)
        {
                this.sizeOfObject = sizeOfObject;
        }
```

```
	public Integer getPopularity()
	{
		return popularity;
	}
	public void setPopularity(Integer popularity)
	{
		this.popularity = popularity;
	}
	public String getWebObject()
	{
		return webObject;
	}
	public void setWebObject(String webObject)
	{
		this.webObject = webObject;
	}
}
package com.sjsu.edu.cachelfu;
import java.util.Collection;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.SortedMap;
import java.util.SortedSet;
import java.util.TreeMap;
import java.util.TreeSet;
import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;

public class CacheLFU {
	/**
	 * Map of Cached Item and Size of the Cached Item
	 */
	private Map<String,Long> cacheMap;

	/**
	 * Cached Item and		Number of Times it was requested
	 */
	private Map<String,Integer> lFUCacheMap;

	/**
	 * Cached Item and the Time at which it was requested
	 */
	private Map<String,Long> cacheTimeMap;

	/**
	 * Cache capacity in terms of the number of items it can store
	 */
	private Integer capacity;

	/**
	 * The Max Cache Size for the Cache
	 */
	//private Integer maxCacheSize;
	private Long maxCacheSize;
	public Long getMaxCacheSize()
	 {
		return maxCacheSize;
	}
```

```
public void setMaxCacheSize(Long maxCacheSize)
{
	this.maxCacheSize = maxCacheSize;
}

/**
 * The total Size of the requested Item from the cache : Parameter<br/>
 * used to compute BHR(Byte Hit Ratio)
 */
private Long totalRequestedItemSize;
/*private Integer totalRequestedItemSize;*/

/**
 * Total number of Items in the Cache
 */
private Long totalNumberOfItems;
/*private Integer totalNumberOfItems;*/

/**
 * Total Cache Hit Size in terms of the Number of Bytes:
 * <br/> Parameter used to compute the BHR(Byte Hit Ratio)
 */
private Long totalCacheHitByteSize;
//private Integer totalCacheHitByteSize;

/**
 * Total Cache Miss Size in terms of the Number of Bytes:
 * <br/> Parameter used to compute the BHR(Byte Hit Ratio)
 */
private Long totalCacheMissByteSize;
//private Integer totalCacheMissByteSize;

/**
 * Absolute Number of the Cache Hits
 * <br/> Parameter used to compute the HR(Hit Ratio)
 */
private Long cacheHits;
/*private Integer cacheHits;*/

/**
 * Absolute Number of the Cache Misses
 * <br/> Parameter used to compute the BHR(Hit Ratio)
 */
private Long cacheMisses;
//private Integer cacheMisses;

/**
 * Parameter used to keep track of the current cache size
 * <br/>in terms of the number of bytes.
 * <br/><b> Parameter Used to prevent Cache Overshooting the Size</b>
 */
private Long currentCacheSize;
//private Integer currentCacheSize;

public Long getCurrentCacheSize() {
	return currentCacheSize;
}

public void setCurrentCacheSize(Long currentCacheSize) {
	this.currentCacheSize = currentCacheSize;
}
```

```
//public CacheLFU(Integer cacheCap)
public CacheLFU(Long maxCacheCapacity)
{
	lFUCacheMap = new HashMap<String, Integer>();
	cacheTimeMap = new HashMap<String, Long>();
	cacheMap = new HashMap<String, Long>();
	totalRequestedItemSize = new Long(0);
	totalNumberOfItems = new Long(0);
	totalCacheHitByteSize = new Long(0);
	totalCacheMissByteSize = new Long(0);
	cacheHits = new Long(0);
	cacheMisses = new Long(0);
	currentCacheSize = new Long(0);
	maxCacheSize = maxCacheCapacity;
}

//public Integer getTotalRequestedItemSize()
public Long getTotalRequestedItemSize()
{
	return totalRequestedItemSize;
}

//public void setTotalRequestedItemSize(Integer totalRequestedItemSize)
public void setTotalRequestedItemSize(Long totalRequestedItemSize)
{
	this.totalRequestedItemSize = totalRequestedItemSize;
}

//public Integer getTotalNumberOfItems()
public Long getTotalNumberOfItems()
{
	return totalNumberOfItems;
}

//public void setTotalNumberOfItems(Integer totalNumberOfItems)
public void setTotalNumberOfItems(Long totalNumberOfItems)
{
	this.totalNumberOfItems = totalNumberOfItems;
}

//public Integer getTotalCacheHitByteSize()
public Long getTotalCacheHitByteSize()
{
	return totalCacheHitByteSize;
}

//public void setTotalCacheHitByteSize(Integer totalCacheHitByteSize)
public void setTotalCacheHitByteSize(Long totalCacheHitByteSize)
{
	this.totalCacheHitByteSize = totalCacheHitByteSize;
}

//public Integer getTotalCacheMissByteSize()
public Long getTotalCacheMissByteSize()
{
	return totalCacheMissByteSize;
}

//public void setTotalCacheMissByteSize(Integer totalCacheMissByteSize)
public void setTotalCacheMissByteSize(Long totalCacheMissByteSize)
{
	this.totalCacheMissByteSize = totalCacheMissByteSize;
```

```
}

//public Integer getCacheHits()
public Long getCacheHits()
{
	return cacheHits;
}

//public void setCacheHits(Integer cacheHits)
public void setCacheHits(Long cacheHits)
{
	this.cacheHits = cacheHits;
}

//public Integer getCacheMisses()
public Long getCacheMisses()
{
	return cacheMisses;
}

//public void setCacheMisses(Integer cacheMisses)
public void setCacheMisses(Long cacheMisses)
{
	this.cacheMisses = cacheMisses;
}

//public Integer getCacheMapSize()
public Integer getCacheMapSize()
{
	return cacheMap.size();
	//return capacity;
}

/*public void setCacheCapacity(Integer cacheCap)
{
	capacity = cacheCap;
}*/

/**
 * The interface to cache Objects
 */
public void cacheItem(Request theItemToBeCached)
{
	addItemToCache(theItemToBeCached);
}

private void addItemToCache(Request theItemToBeCached)
{

	//The actual Object to be cached
	String itemToBeCached = theItemToBeCached.getRequestedURL();

	//The item to be cached's size
	Long requestedItemSize = new Long(theItemToBeCached.getRequestedItemSize());

	//Use case to be executed
	int actionToBeExecuted = computeCase(theItemToBeCached);
	//int actionToBeExecuted = computeCase(itemToBeCached);

	Integer itemCount=null;

	Long currentTime=null;
```

```
switch(actionToBeExecuted)
{
	case 1:
		//Case 1: Cache contains item and cache is full
		//1) Get the existing Item say 'X' from the main queue and Update its count
		//2) If the item 'X' exists on the leastFrequentItemMap remove it
		//3) If the item 'X' exists on the lFUCacheEntryTime remove it

		/**
		 *	- cacheMap<String,Long> : Cached Item and Size

			- lFUCacheMap<String,Integer> : Cached Item and
Number of Times it was requested

			- cacheTimeMap<String,Long> : Cached Item and at
time it was requested
		 */

		//Get the item count used to increase the requested Item count
		itemCount = lFUCacheMap.get(itemToBeCached);

		//Increase the Item count by one
		itemCount+=Configuration.ONE;

		//Get the Current System Time
		currentTime = System.currentTimeMillis();

		//Number of time the item was requested
		lFUCacheMap.put(itemToBeCached, itemCount);

		//Time the item was requested
		cacheTimeMap.put(itemToBeCached, currentTime);

		//Requested Item
		//cacheMap.put(itemToBeCached, requestedItemSize);

		//Use case of Cache Hit : Increase the Cache Hit count by One
		cacheHits += Configuration.ONE;

		//Increase the parameter for cache hit size parameter
		totalCacheHitByteSize = totalCacheHitByteSize + requestedItemSize;

		//Edit the Total Cache Size
		/**
		currentCacheSize = currentCacheSize
+
theItemToBeCached.requestedItemSize;
*/

		break;

	case 2:
		//Case 2: Cache does not contain item, but cache is full
		//Actions:
		//1) Remove LFU item from LFUCache
		//2) Also remove LFU item from LFU replica & LFU time; if they exist on the cache
		//3) Place the new item on the LFUCache
		//4) Update the LFU Replica & LFU Time
```

```
                    //Remove the Least frequent Item
                    removeLeastFrequentItem(theItemToBeCached);

                    //Initialize the requested Item Size to One
                    itemCount = new Integer(Configuration.ONE);

                    //Get the current Time from the System
                    currentTime = new Long(System.currentTimeMillis());

                    //LFUCacheMap - Item to be cached and Item Count
                    lFUCacheMap.put(itemToBeCached, itemCount);

                    //Log the Time at which the item was pushed on to the cache
                    cacheTimeMap.put(itemToBeCached, currentTime);

                    //Record the Entry into the Cache Map
                    //cacheMap.put(itemToBeCached, new Long(theItemToBeCached.requestedItemSize));
                    cacheMap.put(itemToBeCached, requestedItemSize);

                    //Increment the Cache Miss by One
                    cacheMisses += Configuration.ONE;

                    //totalCacheMissByteSize += theItemToBeCached.requestedItemSize;
                    totalCacheMissByteSize = totalCacheMissByteSize + requestedItemSize;

                    //Update the Current Cache Size to reflect the current cache Size
                    currentCacheSize = currentCacheSize
                                    + requestedItemSize;
                    /*currentCacheSize = currentCacheSize
                                                    + theItemToBeCached.requestedItemSize;*/

                    break;

            case 4:
                    //Case 4: Cache does not contain item & Cache is not full
                    //Actions:
                    //1) Push the Item on the LFU Queue
                    //2) Register Item on the LFU Replica
                    //3) Register Item on the LFUCacheTime
                    //4) Also If there are any other LFU items in the leastFrequentMap, lFUCacheReplica
remove those items

                    //Count of the Item
                    itemCount = new Integer(Configuration.ONE);

                    //Get the current Time
                    currentTime = new Long(System.currentTimeMillis());

                    //LFUCacheMap - Item to be cached and Frequency
                    lFUCacheMap.put(itemToBeCached, itemCount);

                    //CacheTimeMap - Item to be cached and the Current Time at which the Cache was filled
                    cacheTimeMap.put(itemToBeCached, currentTime);

                    //Log the Item to be cached in the CacheMap - Item to be cached & Size of the Item
                    //cacheMap.put(itemToBeCached,new Long(theItemToBeCached.requestedItemSize));
                    cacheMap.put(itemToBeCached,requestedItemSize);

                    //Increment the Cache Misses
                    cacheMisses += Configuration.ONE;

                    //Increment the Cache Miss Size parameter
```

```
                                totalCacheMissByteSize = totalCacheMissByteSize + requestedItemSize;
                                //totalCacheMissByteSize += theItemToBeCached.requestedItemSize;

                                //Update the Current Cache Size
                                currentCacheSize = currentCacheSize + requestedItemSize;
                                /*currentCacheSize = currentCacheSize + theItemToBeCached.requestedItemSize;*/

                                break;

                                /*case 3:
                                //Case 3: Cache Already contains item & Cache is not full
                                //Actions:
                                //1) Get the existing Item say 'X' from the main queue and Update its count
                                //2) If the item 'X' exists on the leastFrequentItemMap remove it
                                //3) If the item 'X' exists on the lFUCacheEntryTime remove it

                                itemCount = lFUCacheMap.get(itemToBeCached);

                                itemCount +=CacheConstants.ONE;

                                lFUCacheMap.put(itemToBeCached, itemCount);

                                currentTime = new Long(System.currentTimeMillis());

                                cacheTimeMap.put(itemToBeCached, currentTime);

                                cacheHits += CacheConstants.ONE;

                                totalCacheHitByteSize += theItemToBeCached.requestedItemSize;

                                break;*/

                        default:

                                break;

                }

                /*totalRequestedItemSize = totalRequestedItemSize
                                                                                +
theItemToBeCached.requestedItemSize;*/
                totalRequestedItemSize = totalRequestedItemSize + requestedItemSize;

                totalNumberOfItems = totalNumberOfItems + Configuration.ONE;
        }

        /**
         * Computes which use case is to be executed
         *
         * @param itemToBeCached
         * @return
         */
        //private int computeCase(String itemToBeCached)
        private int computeCase(Request theItemToBeCached)
        {

                Long tempCacheSize = null;
                String itemToBeCached = null;

                //Object to be cached
                itemToBeCached = theItemToBeCached.getRequestedURL();
```

```
		Long itemToBeCachedSize = new Long(theItemToBeCached.getRequestedItemSize());

		//Get the temporary Cache Size
		tempCacheSize = currentCacheSize + itemToBeCachedSize;

		//if(lFUCacheMap.containsKey(itemToBeCached) && lFUCacheMap.size() >=capacity )
		//if(lFUCacheMap.containsKey(itemToBeCached) && tempCacheSize >= maxCacheSize)
		//if(lFUCacheMap.containsKey(itemToBeCached) && tempCacheSize > maxCacheSize)
		if(lFUCacheMap.containsKey(itemToBeCached))
		{//Case 1: Cache contains item and cache is full
			return Configuration.ONE;
		}
		//else if(!lFUCacheMap.containsKey(itemToBeCached)&& lFUCacheMap.size() >=capacity)
		//else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize >= maxCacheSize)
		else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize > maxCacheSize)
		{//Case 2: Cache does not contain item, but cache is full
			return Configuration.TWO;
		}
		//else if(lFUCacheMap.containsKey(itemToBeCached)&&lFUCacheMap.size() < capacity)
		//else if(lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize < maxCacheSize)
		/*else if(lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize <= maxCacheSize)
		{//Case 3: Cache Already contains item & Cache is not full
			return CacheConstants.THREE;
		}*/
		//else if(!lFUCacheMap.containsKey(itemToBeCached)&&lFUCacheMap.size()<capacity)
		//else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize < maxCacheSize)
		else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize <= maxCacheSize)
		{//Case 4: Cache does not contain item, but cache is not full
			return Configuration.FOUR;
		}

		return Configuration.ZERO;
}

/**
 * Helper function to remove the least frequent Item
 * <br/>
 * <b>
 *		<br/>	In case there are multiple items that are least frequently occurring
 *  <br/>  remove the item that was first inserted into the cache in terms of the
 *  <br/>  time it was entered in the cache
 * </b>
 *
 */
public void removeLeastFrequentItem(Request theRequestedItem)
{
		Integer leastCount=null;

		//Variable to get the first key or least count
		Long firstKey = null;

		//The least frequent item reference object
		String leastFrequentItem = null;

		Runtime r = null;

		Long entryTime = null;

		//Set of Least Frequently Used Item's values
		Collection<Integer> countValues = null;

		//Set of Least Frequently Used Item's values
```

```
Set<String> cachedItemSet = null;

SortedMap<Long, String> sortedMap = null;

TreeSet<Integer> countValuesSet = null;

// Stores the current Cache Size after adding
// current Cache size and size of item to be cached
Long tempCacheSize = null;

//Size of the Item to be removed
Long itemToBeRemovedSize = null;

Long theRequestedItemSize = new Long(theRequestedItem.getRequestedItemSize());

/*tempCacheSize = currentCacheSize
                                        + theRequestedItem.requestedItemSize;*/
tempCacheSize = currentCacheSize
                                                    + theRequestedItemSize;

/*****************************************************/
// Remove Item from the cache till the cache has enough space to
// accommodate the item to be cached
/*****************************************************/
//while(tempCacheSize >= maxCacheSize)
while(tempCacheSize > maxCacheSize)
{
        // ++++++++++++++++++++++++++++++++++++++++++
        // Traverse over the counts of items in
        // the Map to get the Least Count
        // ++++++++++++++++++++++++++++++++++++++++++

        //Set of Least Frequently Used Item's values
        countValues = lFUCacheMap.values();

        //Set of actual Cached Items items
        cachedItemSet = lFUCacheMap.keySet();

        //Sorted Map
        sortedMap = new TreeMap<Long, String>();

        //TreeSet - To Hold Least Frequent count for Items in sorted order
        countValuesSet = new TreeSet<Integer>();

        /*** ********************************** */
        // 1. -- Add the various count values
        //       for all items in the Cache  --
        /*** ********************************** */
        countValuesSet.addAll(countValues);

        /*** ********************************** */
        //Get the Least Count value
        /*** ********************************** */
        try
        {
                leastCount = countValuesSet.first();
        }
        catch(Exception ex)
        {
                System.out.println("Exception");
                System.out.println(this.totalNumberOfItems);
        }
```

```
/*** ********************************** */
// 2. --- Get all the Items who have least count ---
// 3. --- Create a Map of
//    (Entry Time for the Least Frequent Item, Least Frequent Item itself)
/*** ********************************** */
for (String string : cachedItemSet)
{
        // Get the items that has leastCount
        if(lFUCacheMap.get(string) == leastCount)
        {
                entryTime = cacheTimeMap.get(string);
                sortedMap.put(entryTime, string);
        }
}

/*** ********************************** */
// 3. --- Get all the Item who have least count
//                          & inserted first into the Cache ---
/*** ********************************** */
firstKey = sortedMap.firstKey();

if(firstKey==null)
{
        System.out.println("First Key Empty");
}

/*** ********************************** */
//4. --- Get the least frequent Item ---
/*** ********************************** */
leastFrequentItem = sortedMap.get(firstKey);

//System.out.println("Throwing out + " + leastFrequentItem);

/*System.out.println("Removing :: " + leastFrequentItem + " Current Cache Size :" +
this.currentCacheSize);*/

//Get the Item to be removed's size in bytes
itemToBeRemovedSize = cacheMap.get(leastFrequentItem);

//Remove the Item from the Cache Map
// - A map of item and size of the item
cacheMap.remove(leastFrequentItem);

//Remove the item from the lfu cache map
// - A map of item and number of times it was requested
lFUCacheMap.remove(leastFrequentItem);

//Remove the item from the cacheTimeMap
// - A map of item and the time it was inputed into the cache
cacheTimeMap.remove(leastFrequentItem);

//Edit the current Cache Size
currentCacheSize = currentCacheSize - itemToBeRemovedSize;

// ++++++++++++++++++++++++++++++
        /*****************************/
        // Initialize the local variables to null
        /*****************************/
// ++++++++++++++++++++++++++++++
cachedItemSet = null;
```

```
                        sortedMap = null;

                        firstKey = null;

                        itemToBeRemovedSize = null;

                        leastCount = null;

                        leastFrequentItem = null;

                        sortedMap = null;

                        countValuesSet = null;

                        countValues = null;
                        // ++++++++++++++++++++++++++++++

                        // Compute the tempCacheSize to make sure the size after inserting
                        // the item to be cached the cache size does not overshoot.
                        tempCacheSize = currentCacheSize + theRequestedItemSize;

                        //tempCacheSize = currentCacheSize + theRequestedItem.requestedItemSize;
                }
                /*******************************************************/
                //While Ends
                /*******************************************************/

                r = Runtime.getRuntime();

                // ++++++++++++++++++++++++++++++
                // Try to do a garbage collection
                // ++++++++++++++++++++++++++++++
                try
                {
                        // +++++++++++++++++++++++++++
                        // Invoke Garbage Collection
                        // +++++++++++++++++++++++++++
                        r.gc();

                        // ++++++++++++++++++++++++++++++++++++++++++++++
                        // Put the currently executing thread to sleep
                        // ++++++++++++++++++++++++++++++++++++++++++++++
                        Thread.sleep(100);

                        // +++++++++++++++++++++++++++
                        // Invoke Garbage Collection
                        // +++++++++++++++++++++++++++
                        r.gc();
                }
                catch (InterruptedException e)
                {
                        e.printStackTrace();
                }
        }
}
package com.sjsu.edu.cachelfu;

import java.io.BufferedInputStream;



import java.io.BufferedOutputStream;
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Date;

import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.printer.Printer;

public class CacheLFUReader {

	private CacheLFU theLFUCache;

	//public CacheLFUReader(Integer capacity) {
	public CacheLFUReader(Long capacity) {
		// theLFUCache = new CacheLFU(capacity);
		theLFUCache = new CacheLFU(capacity);
	}

	public void executeRequest(String fileName, String outputFileName,Long datasetSize)
	{
		readLogFile(fileName, outputFileName,datasetSize);
	}

	@SuppressWarnings({ "deprecation" })
	private void readLogFile(String inputFile, String outputFile,Long datasetSize)
	{
		// Create File from which to read
		File fileInput = new File(inputFile);

		// Create File to which to write
		File fileOutput = new File(new String(outputFile));

		// The FileOutputStream
		FileOutputStream fos = null;

		// The FileInputStream
		FileInputStream fis = null;

		// Number of Requests
		long noOfRequests = Configuration.ZERO;

		// Runtime Object
		Runtime r = Runtime.getRuntime();

		Long numberOfItems = new Long(0);

		try {

			// Make sure the File exists - To prevent File not found exception
			if (fileInput.exists()) {

				// FileInputStream
				fis = new FileInputStream(fileInput);

				// FileOutputStream
				fos = new FileOutputStream(fileOutput);
```

```
// BufferedOutputStream
BufferedOutputStream bos = new BufferedOutputStream(fos);

// BufferedInputStream
BufferedInputStream bis = new BufferedInputStream(fis);

// DataOutputStream
DataOutputStream dos = new DataOutputStream(bos);

// DataInputStream
DataInputStream dis = new DataInputStream(bis);

Long requestedObjectSize = null;

while (dis.available() != Configuration.ZERO)
// while(noOfRequests < 100)
{

        numberOfItems = numberOfItems + Configuration.ONE;

        // Read a line from the Web Log File
        String line = dis.readLine();

         if(numberOfItems < Configuration.seventyPercentDataSet1)
          {
                  continue;
          }

        // Pre-process the request
        //Request request = preProcessRequest(line);
        Request request = preProcessRequest(line);

        // Push the Object to the Cache
        // LRU Cache
        // theLRUCache.cacheItem(request);

        /*** ********************/
        // Make sure the Item to be Cached is no bigger than
        // the Cache itself
        /*** ********************/

        requestedObjectSize = request.getRequestedItemSize();

        //if (request.requestedItemSize < theLFUCache.getMaxCacheSize())
        if (!(requestedObjectSize > theLFUCache.getMaxCacheSize()))
        {
                // LFU Cache
                theLFUCache.cacheItem(request);

                // Once the item is pushed to the Cache set the handle to
                // null
                request = null;

                // Increment the Total Number of requests
                noOfRequests += 1;
                // System.out.println(noOfRequests);
        }

        // Every 25000 request clear out the JVM by trying to run
        // GARBAGE Collector
        if (noOfRequests % Configuration.TWENTYFIVETHOUSHAND ==
Configuration.ZERO)
```

```
                                        {

                                                try {
                                                        // Invoke GARBAGE Collector
                                                        //r.gc();
                                                        GarbageCollectorHelper.invokeGC();

                                                        Printer.printString("Putting the thread to sleep:" +
noOfRequests
                                                                        + ",Current Cache Size:" +
theLFUCache.getCurrentCacheSize() + ", CacheMapSize:" + theLFUCache.getCacheMapSize());

                                                        // Put the Thread to sleep
                                                        //Thread.sleep(Configuration.TWOTHOUSAND);

                                                        GarbageCollectorHelper.invokeGC();

                                                        // Invoke GARBAGE Collector
                                                        //r.gc();

                                                } catch (InterruptedException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }
                                        }
                                }

                                printLFUStats();
                                // Attempt a write to file
                                // dos.writeBytes("Total Hits : " + theLRUCache.getCacheHits());

                        }

                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } finally {
                        try {
                                fis.close();
                                fos.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }

                }
        }

        private void printLFUStats() {
                // Once the Complete Processing of all Web Log Entries is done - Print
                // out all the details

                System.out.println("Total Number of Items : "
                                + theLFUCache.getTotalNumberOfItems());
                System.out.println("Total Misses : " + theLFUCache.getCacheMisses());
                System.out.println("Total Hits : " + theLFUCache.getCacheHits());
                System.out.println("Total CacheSize : "
                                + theLFUCache.getTotalRequestedItemSize());
                System.out.println("Cache Map Size : " + theLFUCache.getCacheMapSize());
                System.out.println("Total Cache Hit Size : "
                                + theLFUCache.getTotalCacheHitByteSize());
                System.out.println("Total Cache Miss Size : "
```

```
                                + theLFUCache.getTotalCacheMissByteSize());
        Date currentDate = new Date(System.currentTimeMillis());
        System.out.println(" End TimeStamp " + currentDate.getMonth()
                                + currentDate.getDate() + "," + currentDate.getYear()
                                + currentDate.getHours() + ":" + currentDate.getMinutes() + ":"
                                + currentDate.getSeconds());

}

private Request preProcessRequest(String stringRequest)
{
        String[] requestParam = stringRequest.split(" ");
        //int i=0;
        Request request = new Request();

        /*Long requestTime = new Long(requestParam[0].toString());*/
        /*Long requestTime = new Long(requestParam[0].toString());*/
        String requestTimeString = requestParam[0].toString();

        int index = requestTimeString.indexOf(".");

        String reqString = requestTimeString.substring(Configuration.ZERO, index);

        Long requestTime = new Long(reqString);
        Long requestProcessingTime = new Long(requestParam[1]);
        String clientIPAddress = requestParam[2].trim();
        String tcpCode = requestParam[3].trim();
        Long requestedItemSize = new Long(requestParam[4].trim());
        String requestedObject = requestParam[6].trim();
        String timeOutRedirectionAttribute = requestParam[8].trim();
        String requestedObjectType = requestParam[9].trim();

        String requestedURL = null;

        requestedURL = processURL(requestedObject).trim();

        request.setRequestTime(requestTime);
        request.setRequestProcessingTime(requestProcessingTime);
        request.setClientIPAddress(clientIPAddress);
        request.setTcpCode(tcpCode);
        request.setRequestedItemSize(requestedItemSize);
        request.setRequestedObjectType(requestedObject);
        request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
        request.setRequestedObjectType(requestedObjectType);
        request.setRequestedURL(requestedURL);

        requestParam = null;

        return request;
}

private String processURL(String url)
{
        String URL = url;


        //URL : http://www.gmail.com/skd/
        URL = URL.trim();

        int index = URL.indexOf("//");
        int index2 = -1;
```

```
                    //www.gmail.com/skd/
                    URL = URL.substring(index + Configuration.TWO);

                    return URL;

/*                  index2 = URL.indexOf("/");

                    if(index2>-1)
                    {
                            URL =
                                    URL.substring(CacheConstants.ZERO,index2);
                    }
                    else
                    {
                            URL =
                                    URL.substring(CacheConstants.ZERO);
                    }

                    if(URL.contains(":"))
                    {
                            int index3 = URL.indexOf(":");

                            URL =
                                    URL.substring(CacheConstants.ZERO,index3);
                    }
*/
                    //return URL;
            }
}

package com.sjsu.edu.cachelfu;
import com.sjsu.edu.commonconstants.Configuration;
public class CacheLFUTest {

            public static void main(String[] args)
            {
                    //Main Testing Content - Error Throwing Data Set
                    String inputFile = Configuration.dataSet4Path;

                    CacheLFUReader theLRULogReader = new CacheLFUReader(Configuration.dataSet4CacheSize);

                    String outputFile = Configuration.resultsPath;

                    Long dataSetSize = Configuration.seventyPercentDataSet4;

                    theLRULogReader.executeRequest(inputFile,outputFile,dataSetSize);
            }

}

package com.sjsu.edu.printer;

public class Printer {

            public static void printString(String toBePrinted){
                    System.out.println(toBePrinted);
            }

            public static void printInteger(int integerToBePrinted){

                    Integer toBePrinted = new Integer(integerToBePrinted);
```

```
                System.out.println(toBePrinted.toString());
        }
}

package com.sjsu.edu.cachelfu;

import com.sjsu.edu.commonconstants.Configuration;

public class HybridCacheLFUTest {

        public static void main(String[] args)
        {
                //Main Testing Content - Error Throwing Data Set
                String inputFile = Configuration.dataSet4Path;

                HybridCacheLFUReader theLfULogReader = new
HybridCacheLFUReader(Configuration.dataSet4CacheSize);

                String outputFile = Configuration.resultsPath;

                Long datasetSize = Configuration.seventyPercentDataSet4;

                theLfULogReader.executeRequest(inputFile,outputFile,datasetSize);
        }

}

package com.sjsu.edu.cachelfu;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Date;

import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.printer.Printer;

public class HybridCacheLFUReader {

        private HybridCacheLFU theLFUCache;
        private Long datasize;

        public HybridCacheLFUReader()
        {
                datasize = new Long(0);
        }

        //public CacheLFUReader(Integer capacity) {
        public HybridCacheLFUReader(Long capacity) {
                // theLFUCache = new CacheLFU(capacity);
                theLFUCache = new HybridCacheLFU(capacity);
        }

        public void executeRequest(String fileName, String outputFileName,Long datasetSize)
```

```
{
	readLogFile(fileName, outputFileName,datasetSize);
}

@SuppressWarnings({ "deprecation" })
private void readLogFile(String inputFile, String outputFile,Long datasetSize)
{
	// Create File from which to read
	File fileInput = new File(inputFile);

	// Create File to which to write
	File fileOutput = new File(new String(outputFile));

	// The FileOutputStream
	FileOutputStream fos = null;

	// The FileInputStream
	FileInputStream fis = null;

	long numberOfItems = 0;

	// Number of Requests
	long noOfRequests = Configuration.ZERO;

	// Runtime Object
	Runtime r = Runtime.getRuntime();

	try {

		// Make sure the File exists - To prevent File not found exception
		if (fileInput.exists()) {

			// FileInputStream
			fis = new FileInputStream(fileInput);

			// FileOutputStream
			fos = new FileOutputStream(fileOutput);

			// BufferedOutputStream
			BufferedOutputStream bos = new BufferedOutputStream(fos);

			// BufferedInputStream
			BufferedInputStream bis = new BufferedInputStream(fis);

			// DataOutputStream
			DataOutputStream dos = new DataOutputStream(bos);

			// DataInputStream
			DataInputStream dis = new DataInputStream(bis);

			Long requestedObjectSize = null;

			while (dis.available() != Configuration.ZERO)
			// while(noOfRequests < 100)
			{
				numberOfItems = numberOfItems + 1;

				// Read a line from the Web Log File
				String line = dis.readLine();

				if(numberOfItems < datasetSize)
				{
```

```
                                        continue;
                                }

                                // Pre-process the request
                                //Request request = preProcessRequest(line);
                                Request request = preProcessRequest(line);

                                // Push the Object to the Cache
                                this.datasize = datasize;

                                // LRU Cache
                                // theLRUCache.cacheItem(request);

                                /*** ********************/
                                // Make sure the Item to be Cached is no bigger than
                                // the Cache itself
                                /*** ********************/

                                requestedObjectSize = request.getRequestedItemSize();

                                //if (request.requestedItemSize < theLFUCache.getMaxCacheSize())
                                if (!(requestedObjectSize > theLFUCache.getMaxCacheSize()))
                                {
                                        // LFU Cache
                                        theLFUCache.cacheItem(request);

                                        // Once the item is pushed to the Cache set the handle to
                                        // null
                                        request = null;

                                        // Increment the Total Number of requests
                                        noOfRequests += 1;
                                        // System.out.println(noOfRequests);
                                }

                                // Every 25000 request clear out the JVM by trying to run
                                // GARBAGE Collector
                                if (noOfRequests % Configuration.TWENTYFIVETHOUSHAND ==
Configuration.ZERO)
                                {

                                                invokegc();

                                                Printer.printString("Putting the thread to sleep:" +
noOfRequests
                                                                + ",Current Cache Size:" +
theLFUCache.getCurrentCacheSize() + ", CacheMapSize:" + theLFUCache.getCacheMapSize());

                                                invokegc();


                                        }
                                }

                                printLFUStats();
                                // Attempt a write to file
                                // dos.writeBytes("Total Hits : " + theLRUCache.getCacheHits());

                        }

                } catch (IOException e) {
                        // TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                } finally {
                    try {
                        fis.close();
                        fos.close();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }

                }
            }

            private void printLFUStats() {
                // Once the Complete Processing of all Web Log Entries is done - Print
                // out all the details
                invokegc();
                System.out.println("Total Number of Items : "
                        + theLFUCache.getTotalNumberOfItems());
                System.out.println("Total Misses : " + theLFUCache.getCacheMisses());
                System.out.println("Total Hits : " + theLFUCache.getCacheHits());
                System.out.println("Total CacheSize : "
                        + theLFUCache.getTotalRequestedItemSize());
                System.out.println("Cache Map Size : " + theLFUCache.getCacheMapSize());
                System.out.println("Total Cache Hit Size : "
                        + theLFUCache.getTotalCacheHitByteSize());
                System.out.println("Total Cache Miss Size : "
                        + theLFUCache.getTotalCacheMissByteSize());
                Date currentDate = new Date(System.currentTimeMillis());
                System.out.println(" End TimeStamp " + currentDate.getMonth()
                        + currentDate.getDate() + "," + currentDate.getYear()
                        + currentDate.getHours() + ":" + currentDate.getMinutes() + ":"
                        + currentDate.getSeconds());

            System.out.println("/*************************************************************************/");
                    System.out.println("The total Number Of items Prefetched:" +
theLFUCache.getNumberOfClusterPrefetched());
                    System.out.println("The total Number of items Prefetched Hits:" +
theLFUCache.getNumberOfClusterPrefetchHits());
                    System.out.println("The total Number of items Prefetched Items Removed:" +
theLFUCache.getNumberOfClusterPrefetchedRemoved());
                    System.out.println("The toal Number of items Prefetched Items Misses:" +
theLFUCache.getNumberOfClusterPrefetchMisses());
                    System.out.println("The total Size of Items Prefetched:" + theLFUCache.getSizeOfItemPrefetched());
                    System.out.println("The total Cache Hit Size of Items Prefetched:" +
theLFUCache.getSizeOfItemsPrefetchedHits());
                    System.out.println("The total Size of Cache Miss of Items Prefetched:" +
theLFUCache.getSizeOfItemsPrefetchedMiss());
                    System.out.println("The total Size of Prefetched Items Removed:
"+theLFUCache.getSizeOfItemsPrefetchedRemoved());
                    System.out.println("The toal number of Prefetch Cache Misses :" +
theLFUCache.getCacheMissList().size());

            System.out.println("/*************************************************************************/");


            }

            private Request preProcessRequest(String stringRequest)
            {
                String[] requestParam = stringRequest.split(" ");
                //int i=0;
```

```
		Request request = new Request();

		/*Long requestTime = new Long(requestParam[0].toString());*/
		String requestTimeString = requestParam[0].toString();

		/*Long requestTime = new Long(requestParam[0].toString());*/
		int index = requestTimeString.indexOf(".");

		String reqString = requestTimeString.substring(Configuration.ZERO, index);


		Long requestProcessingTime = new Long(requestParam[1]);
		String clientIPAddress = requestParam[2].trim();
		String tcpCode = requestParam[3].trim();
		Long requestedItemSize = new Long(requestParam[4].trim());
		String requestedObject = requestParam[6].trim();
		String timeOutRedirectionAttribute = requestParam[8].trim();
		String requestedObjectType = requestParam[9].trim();
		String domainName = null;
		String requestedURL = null;

		requestedURL = processURL(requestedObject).trim();
		domainName = processURLDomainName(requestedObject);

		Long requestTime = new Long(reqString);

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedObject(requestedURL);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);
		request.setDomainName(domainName);

		requestParam = null;

		return request;
	}

	private String processURL(String url)
	{
		String URL = url;


		//URL : http://www.gmail.com/skd/
		URL = URL.trim();

		int index = URL.indexOf("//");
		int index2 = -1;

		//www.gmail.com/skd/
		URL = URL.substring(index + Configuration.TWO);

		return URL;

/*		index2 = URL.indexOf("/");

		if(index2>-1)
		{
			URL =
```

```
                                        URL.substring(CacheConstants.ZERO,index2);
                        }
                        else
                        {
                                URL =
                                        URL.substring(CacheConstants.ZERO);
                        }

                        if(URL.contains(":"))
                        {
                                int index3 = URL.indexOf(":");

                                URL =
                                        URL.substring(CacheConstants.ZERO,index3);
                        }
*/
                        //return URL;
                }

                private String processURLDomainName(String url)
                {
                        String URL = url;


                        //URL : http://www.gmail.com/skd/
                        URL = URL.trim();

                        int index = URL.indexOf("//");
                        int index2 = -1;

                        //www.gmail.com/skd/
                        URL = URL.substring(index + Configuration.TWO);

                        index2 = URL.indexOf("/");

                        if(index2>-1)
                        {
                                URL =
                                        URL.substring(Configuration.ZERO,index2);
                        }
                        else
                        {
                                URL =
                                        URL.substring(Configuration.ZERO);
                        }

                        if(URL.contains(":"))
                        {
                                int index3 = URL.indexOf(":");

                                URL =
                                        URL.substring(Configuration.ZERO,index3);
                        }

                        return URL;
                }

                private void invokegc()
                {
                        try
                        {
                                GarbageCollectorHelper.invokeGC();
```

```
                        }
                        catch (InterruptedException e)
                        {
                                e.printStackTrace();

                        }
                }
}
package com.sjsu.edu.cachelfu;

import java.io.IOException;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import java.util.TreeSet;

import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.fileprocessor.FileProcessor;
import com.sjsu.edu.node.Node;

public class HybridCacheLFU {

        /**
         * Map of Cached Item and Size of the Cached Item
         */
        private Map<String,Long> cacheMap;

        /**
         * Cached Item and              Number of Times it was requested
         */
        private Map<String,Integer> lFUCacheMap;

        /**
         * Cached Item and the Time at which it was requested
         */
        private Map<String,Long> cacheTimeMap;

        /**
         * Cache capacity in terms of the number of items it can store
         */
        private Integer capacity;

        /**
         * The Max Cache Size for the Cache
         */
        //private Integer maxCacheSize;
        private Long maxCacheSize;

        /*** ------------------------ **/
        //For Prefetching Statistics
        /*** ------------------------ **/
        //Cluster Cache
        private Set<String> clusterCache;
        private Map<String,Boolean> clustCache;
```

```
public Map<String, Boolean> getClustCache()
{
	return clustCache;
}

public void setClustCache(Map<String, Boolean> clustCache) {
	this.clustCache = clustCache;
}

private List<String> cacheMissList;

public List<String> getCacheMissList() {
	return cacheMissList;
}

public void setCacheMissList(List<String> cacheMissList) {
	this.cacheMissList = cacheMissList;
}

//Number Of Cluster Prefetched
private Long numberOfClusterPrefetched;

public Long getNumberOfClusterPrefetched() {
	return numberOfClusterPrefetched;
}

public void setNumberOfClusterPrefetched(Long numberOfClusterPrefetched) {
	this.numberOfClusterPrefetched = numberOfClusterPrefetched;
}

//Number Of Cluster Prefetch Hits
private Long numberOfClusterPrefetchHits;

public Long getNumberOfClusterPrefetchHits() {
	return numberOfClusterPrefetchHits;
}

public void setNumberOfClusterPrefetchHits(Long numberOfClusterPrefetchHits) {
	this.numberOfClusterPrefetchHits = numberOfClusterPrefetchHits;
}

//Number Of Cluster Prefetch Misses
private Long numberOfClusterPrefetchMisses;

//Number Of Cluster Prefetched Removed
private Long numberOfClusterPrefetchedRemoved;

//Size of Prefetched Items Removed
private Long sizeOfItemsPrefetchedRemoved;

//Size of All items Prefetched
private Long sizeOfItemPrefetched;

//Size of All Item Hits
private Long sizeOfItemsPrefetchedHits;

//Size of All Items Prefetched Miss
private Long sizeOfItemsPrefetchedMiss;
/*** ---------------------- **/

/**
 * Clustered Web Object Cache
```

```
 */
private Map<String,Long> clusteredWebObjCache;

public Long getMaxCacheSize() {
		return maxCacheSize;
}

public void setMaxCacheSize(Long maxCacheSize)
{
		this.maxCacheSize = maxCacheSize;
}

/**
 * The total Size of the requested Item from the cache : Parameter<br/>
 * used to compute BHR(Byte Hit Ratio)
 */
private Long totalRequestedItemSize;
/*private Integer totalRequestedItemSize;*/

/**
 * Total number of Items in the Cache
 */
private Long totalNumberOfItems;
/*private Integer totalNumberOfItems;*/

/**
 * Total Cache Hit Size in terms of the Number of Bytes:
 * <br/> Parameter used to compute the BHR(Byte Hit Ratio)
 */
private Long totalCacheHitByteSize;
//private Integer totalCacheHitByteSize;

/**
 * Total Cache Miss Size in terms of the Number of Bytes:
 * <br/> Parameter used to compute the BHR(Byte Hit Ratio)
 */
private Long totalCacheMissByteSize;
//private Integer totalCacheMissByteSize;

/**
 * Absolute Number of the Cache Hits
 * <br/> Parameter used to compute the HR(Hit Ratio)
 */
private Long cacheHits;
/*private Integer cacheHits;*/

/**
 * Absolute Number of the Cache Misses
 * <br/> Parameter used to compute the BHR(Hit Ratio)
 */
private Long cacheMisses;
//private Integer cacheMisses;

/**
 * Parameter used to keep track of the current cache size
 * <br/>in terms of the number of bytes.
 * <br/><b> Parameter Used to prevent Cache Overshooting the Size</b>
 */
private Long currentCacheSize;
//private Integer currentCacheSize;

public Long getCurrentCacheSize()
```

```
{
	return currentCacheSize;
}

public void setCurrentCacheSize(Long currentCacheSize) {
	this.currentCacheSize = currentCacheSize;
}

//public CacheLFU(Integer cacheCap)
public HybridCacheLFU(Long maxCacheCapacity)
{

	/*** ------------------------ **/
	//For Prefetching Statistics
	/*** ------------------------ **/
	numberOfClusterPrefetchHits = new Long(0);
	numberOfClusterPrefetched = new Long(0);
	clusterCache = new HashSet<String>();
	numberOfClusterPrefetchMisses = new Long(0);
	numberOfClusterPrefetchedRemoved = new Long(0);
	sizeOfItemsPrefetchedRemoved = new Long(0);
	sizeOfItemPrefetched = new Long(0);
	sizeOfItemsPrefetchedHits = new Long(0);
	sizeOfItemsPrefetchedMiss = new Long(0);
	/*** ------------------------ **/
	//---- Ends
	/*** ------------------------ **/
	clustCache = new HashMap<String, Boolean>();
	cacheMissList = new LinkedList<String>();

	lFUCacheMap = new HashMap<String, Integer>();
	cacheTimeMap = new HashMap<String, Long>();
	cacheMap = new HashMap<String, Long>();
	totalRequestedItemSize = new Long(0);
	totalNumberOfItems = new Long(0);
	totalCacheHitByteSize = new Long(0);
	totalCacheMissByteSize = new Long(0);
	cacheHits = new Long(0);
	cacheMisses = new Long(0);
	currentCacheSize = new Long(0);
	maxCacheSize = maxCacheCapacity;
}

//public Integer getTotalRequestedItemSize()
public Long getTotalRequestedItemSize()
{
	return totalRequestedItemSize;
}

//public void setTotalRequestedItemSize(Integer totalRequestedItemSize)
public void setTotalRequestedItemSize(Long totalRequestedItemSize)
{
	this.totalRequestedItemSize = totalRequestedItemSize;
}

//public Integer getTotalNumberOfItems()
public Long getTotalNumberOfItems()
{
	return totalNumberOfItems;
}

//public void setTotalNumberOfItems(Integer totalNumberOfItems)
```

```
public void setTotalNumberOfItems(Long totalNumberOfItems)
{
	this.totalNumberOfItems = totalNumberOfItems;
}

//public Integer getTotalCacheHitByteSize()
public Long getTotalCacheHitByteSize()
{
	return totalCacheHitByteSize;
}

//public void setTotalCacheHitByteSize(Integer totalCacheHitByteSize)
public void setTotalCacheHitByteSize(Long totalCacheHitByteSize)
{
	this.totalCacheHitByteSize = totalCacheHitByteSize;
}

//public Integer getTotalCacheMissByteSize()
public Long getTotalCacheMissByteSize()
{
	return totalCacheMissByteSize;
}

//public void setTotalCacheMissByteSize(Integer totalCacheMissByteSize)
public void setTotalCacheMissByteSize(Long totalCacheMissByteSize)
{
	this.totalCacheMissByteSize = totalCacheMissByteSize;
}

//public Integer getCacheHits()
public Long getCacheHits()
{
	return cacheHits;
}

//public void setCacheHits(Integer cacheHits)
public void setCacheHits(Long cacheHits)
{
	this.cacheHits = cacheHits;
}

//public Integer getCacheMisses()
public Long getCacheMisses()
{
	return cacheMisses;
}

//public void setCacheMisses(Integer cacheMisses)
public void setCacheMisses(Long cacheMisses)
{
	this.cacheMisses = cacheMisses;
}

//public Integer getCacheMapSize()
public Integer getCacheMapSize()
{
	return cacheMap.size();
	//return capacity;
}

/*public void setCacheCapacity(Integer cacheCap)
{
```

```
		capacity = cacheCap;
}*/

/**
 * The interface to cache Objects
 * @throws IOException
 */
public void cacheItem(Request theItemToBeCached) throws IOException
{
		//addItemToCache(theItemToBeCached);
		addItem(theItemToBeCached);
}

private void addItem(Request theRequestedItem) throws IOException
{
		Map<Node,Map<String,Long>> consolidateCluster = null;

		if(lFUCacheMap.containsKey(theRequestedItem.getRequestedObject()))
		{
			addItemToCache(theRequestedItem);
		}
		else
		{
			consolidateCluster = processRequest(theRequestedItem);

			if(consolidateCluster == null)
			{
				addItemToCache(theRequestedItem);
			}
			else
			{
				loadClusterIntoCache(consolidateCluster, theRequestedItem);
				addItemToCache(theRequestedItem);
			}
		}
}

/**
 *
 * Helper checks if the object exists in any of the clusters
 *
 * @param theRequestedItem
 * @return
 * @throws IOException
 */
private Map<Node,Map<String,Long>> processRequest(Request theRequestedItem) throws IOException
{
		FileProcessor fileProcessor = new FileProcessor();

		Map<Node,Map<String,Long>> consolidateCluster =
				fileProcessor.processRequest(theRequestedItem);

		return consolidateCluster;
}

/**
 *
 * Helper to Load all items in the  Cluster into the Cache
 *
 */
private void loadClusterIntoCache
(Map<Node,Map<String,Long>> consolidatedCluster,Request theRequestedItem)
```

```
	{
		Long size = null;
		Long sizeOfItemsToBeRemoved = new Long(0);
		Long tempSize = null;
		String currentURL = null;

		Set<Node> nodes = consolidatedCluster.keySet();
		Set<String> nodesAlreadyInCache = new HashSet<String>();

		for (Node node : nodes)
		{
			currentURL = node.getURL();

			clusteredWebObjCache = consolidatedCluster.get(node);

			Set<String> clusteredObject = clusteredWebObjCache.keySet();

			//Calculate size of all Items to be added to the cache
			for (String webObjectToBeInserted : clusteredObject)
			{
				if(cacheMap.containsKey(webObjectToBeInserted))
				{
					cacheTimeMap.put(webObjectToBeInserted, System.currentTimeMillis());

					nodesAlreadyInCache.add(webObjectToBeInserted);

					/** ------------------------------------------------------ */
					/** ------------ Cluster Prefetch Statistics ------------ */
					/** ------------------------------------------------------ */
					numberOfClusterPrefetched = numberOfClusterPrefetched + 1;
					sizeOfItemPrefetched = sizeOfItemPrefetched +
cacheMap.get(webObjectToBeInserted);
					clustCache.put(webObjectToBeInserted, true);
					/** ------------------------------------------------------ */
					/** ------------ Cluster Prefetch Statistics ------------ */
					/** ------------------------------------------------------ */

				}
				else
				{
					size = clusteredWebObjCache.get(webObjectToBeInserted);

					sizeOfItemsToBeRemoved = sizeOfItemsToBeRemoved + size;
				}
			}

			for (String intraSiteObjectToBeIgrnored : nodesAlreadyInCache)
			{
				clusteredWebObjCache.remove(intraSiteObjectToBeIgrnored);
			}
		}

		tempSize = currentCacheSize + sizeOfItemsToBeRemoved;

		if(tempSize > maxCacheSize)
		{

			createSpaceInCache(tempSize,sizeOfItemsToBeRemoved,theRequestedItem);
			loadObjectsIntoCache(consolidatedCluster);
		}
		else
		{
```

```
                        loadObjectsIntoCache(consolidatedCluster);
                }
}

public Long getSizeOfItemsPrefetchedRemoved() {
        return sizeOfItemsPrefetchedRemoved;
}

public void setSizeOfItemsPrefetchedRemoved(Long sizeOfItemsPrefetchedRemoved) {
        this.sizeOfItemsPrefetchedRemoved = sizeOfItemsPrefetchedRemoved;
}

public Long getSizeOfItemPrefetched() {
        return sizeOfItemPrefetched;
}

public void setSizeOfItemPrefetched(Long sizeOfItemPrefetched) {
        this.sizeOfItemPrefetched = sizeOfItemPrefetched;
}

public Long getSizeOfItemsPrefetchedHits() {
        return sizeOfItemsPrefetchedHits;
}

public void setSizeOfItemsPrefetchedHits(Long sizeOfItemsPrefetchedHits) {
        this.sizeOfItemsPrefetchedHits = sizeOfItemsPrefetchedHits;
}

public Long getSizeOfItemsPrefetchedMiss() {
        return sizeOfItemsPrefetchedMiss;
}

public void setSizeOfItemsPrefetchedMiss(Long sizeOfItemsPrefetchedMiss) {
        this.sizeOfItemsPrefetchedMiss = sizeOfItemsPrefetchedMiss;
}

/**
 * Removes Item from the Cache to make space into the Cache
 */
private void createSpaceInCache(Long tempSize,Long sizeOfItemsToBeRemoved,Request theRequestedItem)
{
        Collection<Integer> lFUCounts = null;
        TreeSet<Integer> sortedLeastFrequentSet = new TreeSet<Integer>();
        Integer firstItem = null;
        Integer leastCount=null;
        Runtime r = null;
        SortedMap<Long, String> sortedMap = null;
        TreeSet<Integer> countValuesSet = null;
        Long entryTime = null;

        //Variable to get the first key or least count
        Long firstKey = null;

        //The least frequent item reference object
        String leastFrequentItem = null;

        //Set of Least Frequently Used Item's values
        Collection<Integer> countValues = null;

        //Set of Least Frequently Used Item's values
        Set<String> cachedItemSet = null;
```

```
// Stores the current Cache Size after adding
// current Cache size and size of item to be cached
Long tempCacheSize = null;

//Size of the Item to be removed
Long itemToBeRemovedSize = null;

Long theRequestedItemSize = theRequestedItem.getRequestedItemSize();

lFUCounts = lFUCacheMap.values();
sortedLeastFrequentSet.addAll(lFUCounts);

while(tempSize > maxCacheSize)
{
	// ++++++++++++++++++++++++++++++++++++++++++++
	// Traverse over the counts of items in
	// the Map to get the Least Count
	// ++++++++++++++++++++++++++++++++++++++++++++
	//Set of Least Frequently Used Item's values
	countValues = lFUCacheMap.values();

	//Set of actual Cached Items items
	cachedItemSet = lFUCacheMap.keySet();

	//Sorted Map
	sortedMap = new TreeMap<Long, String>();

	//TreeSet - To Hold Least Frequent count for Items in sorted order
	countValuesSet = new TreeSet<Integer>();

	/*** ********************************** */
	// 1. -- Add the various count values
	//	for all items in the Cache  --
	/*** ********************************** */
	countValuesSet.addAll(countValues);

	/*** ********************************** */
	//Get the Least Count value
	/*** ********************************** */
	try
	{
		leastCount = countValuesSet.first();
	}
	catch(Exception ex)
	{
		System.out.println("Exception");
		System.out.println(this.totalNumberOfItems);
		//return;
	}

	/*** ********************************** */
	// 2. --- Get all the Items who have least count ---
	// 3. --- Create a Map of
	//    (Entry Time for the Least Frequent Item, Least Frequent Item itself)
	/*** ********************************** */
	for (String string : cachedItemSet)
	{
		// Get the items that has leastCount
		if(lFUCacheMap.get(string) == leastCount)
		{
			entryTime = cacheTimeMap.get(string);
			sortedMap.put(entryTime, string);
```

```
                }
        }

        /*** ********************************** */
        // 3. --- Get all the Item who have least count
        //                          & inserted first into the Cache ---
        /*** ********************************** */
        firstKey = sortedMap.firstKey();

        if(firstKey==null)
        {
                System.out.println("First Key Empty");
        }

        /*** ********************************** */
        //4. --- Get the least frequent Item ---
        /*** ********************************** */
        leastFrequentItem = sortedMap.get(firstKey);

        //Get the Item to be removed's size in bytes
        itemToBeRemovedSize = cacheMap.get(leastFrequentItem);

        //Remove the Item from the Cache Map
        // - A map of item and size of the item
        cacheMap.remove(leastFrequentItem);

        //Remove the item from the lfu cache map
        // - A map of item and number of times it was requested
        lFUCacheMap.remove(leastFrequentItem);

        //Remove the item from the cacheTimeMap
        // - A map of item and the time it was inputed into the cache
        cacheTimeMap.remove(leastFrequentItem);

        /** ------------------------------------------------------ */
        /** ------------ Cluster Prefetch Statistics ------------ */
        /** ------------------------------------------------------ */
        if(clusterCache.contains(leastFrequentItem))
        {
                numberOfClusterPrefetchedRemoved = numberOfClusterPrefetchedRemoved + 1;
                clusterCache.remove(leastFrequentItem);

                Boolean b = clustCache.get(leastFrequentItem);

                if(!b)
                {
                        cacheMissList.add(leastFrequentItem);
                }

                clustCache.remove(leastFrequentItem);
                sizeOfItemsPrefetchedRemoved = sizeOfItemsPrefetchedRemoved +
itemToBeRemovedSize;
        }
        /** ------------------------------------------------------ */
        /** ------------ Cluster Prefetch Statistics ------------ */
        /** ------------------------------------------------------ */


        //Edit the current Cache Size
        currentCacheSize = currentCacheSize - itemToBeRemovedSize;

        tempSize = tempSize - itemToBeRemovedSize;
```

```
                    // ++++++++++++++++++++++++++++++
                            /*******************************/
                            // Initialize the local variables to null
                            /*******************************/
                    // ++++++++++++++++++++++++++++++
                    cachedItemSet = null;

                    sortedMap = null;

                    firstKey = null;

                    itemToBeRemovedSize = null;

                    leastCount = null;

                    leastFrequentItem = null;

                    sortedMap = null;

                    countValuesSet = null;

                    countValues = null;
                    // ++++++++++++++++++++++++++++++

                    // Compute the tempCacheSize to make sure the size after inserting
                    // the item to be cached the cache size does not overshoot.
                    tempCacheSize = currentCacheSize + theRequestedItemSize;
            }
}

/**
 * Helper that loads all the objects Into the Cache
 */
private void loadObjectsIntoCache(Map<Node,Map<String,Long>> consolidateCluster)
{
            Set<Node> nodes = consolidateCluster.keySet();

            Map<String,Long> intraSiteCluster = null;

            Set<String> intraSiteWebObjects = null;

            Long size = null;

            Long currentTime = null;

            for (Node node : nodes)
            {
                    intraSiteCluster = consolidateCluster.get(node);

                    intraSiteWebObjects = intraSiteCluster.keySet();

                    for (String webObjectToBeInserted : intraSiteWebObjects)
                    {
                            size = new Long(intraSiteCluster.get(webObjectToBeInserted));

                            currentTime = System.currentTimeMillis();

                            lFUCacheMap.put(webObjectToBeInserted, (Integer)1);

                            cacheTimeMap.put(webObjectToBeInserted, currentTime);
```

```
                                cacheMap.put(webObjectToBeInserted, size);

                                currentCacheSize = currentCacheSize + size;

                                /*** ------------------------ **/
                                //For Prefetching Statistics
                                /*** ------------------------ **/
                                clusterCache.add(webObjectToBeInserted);
                                clustCache.put(webObjectToBeInserted, false);
                                numberOfClusterPrefetched = numberOfClusterPrefetched + 1;
                                sizeOfItemPrefetched = sizeOfItemPrefetched + size;
                                /*** ------------------------ **/
                                //---- Ends
                                /*** ------------------------ **/

                        }
                }
}
private void addItemToCache(Request theItemToBeCached)
{

        //The actual Object to be cached
        String itemToBeCached = theItemToBeCached.getRequestedObject();

        //The item to be cached's size
        Long requestedItemSize = new Long(theItemToBeCached.getRequestedItemSize());

        //Use case to be executed
        int actionToBeExecuted = computeCase(theItemToBeCached);
        //int actionToBeExecuted = computeCase(itemToBeCached);

        Integer itemCount=null;

        Long currentTime=null;

        switch(actionToBeExecuted)
        {
                case 1:
                        //Case 1: Cache contains item and cache is full
                        //1) Get the existing Item say 'X' from the main queue and Update its count
                        //2) If the item 'X' exists on the leastFrequentItemMap remove it
                        //3) If the item 'X' exists on the lFUCacheEntryTime remove it

                        /**
                         *      - cacheMap<String,Long> : Cached Item and Size

                                - lFUCacheMap<String,Integer> : Cached Item and
Number of Times it was requested

                                - cacheTimeMap<String,Long> : Cached Item and at
time it was requested
                         */

                        //Get the item count used to increase the requested Item count
                        itemCount = lFUCacheMap.get(itemToBeCached);

                        //Increase the Item count by one
                        itemCount+=Configuration.ONE;

                        //Get the Current System Time
```

```
currentTime = System.currentTimeMillis();

//Number of time the item was requested
lFUCacheMap.put(itemToBeCached, itemCount);

//Time the item was requested
cacheTimeMap.put(itemToBeCached, currentTime);

//Requested Item
//cacheMap.put(itemToBeCached, requestedItemSize);

//Use case of Cache Hit : Increase the Cache Hit count by One
cacheHits += Configuration.ONE;

//Increase the parameter for cache hit size parameter
totalCacheHitByteSize = totalCacheHitByteSize + requestedItemSize;

/** ------------------------------------------------------ */
/** ------------ Cluster Prefetch Statistics ------------ */
/** ------------------------------------------------------ */
if(clusterCache.contains(itemToBeCached))
{
        numberOfClusterPrefetchHits = numberOfClusterPrefetchHits + 1;
        sizeOfItemsPrefetchedHits = sizeOfItemsPrefetchedHits +
theItemToBeCached.getRequestedItemSize();

        clustCache.put(itemToBeCached, true);
}
/** ------------------------------------------------------ */
/** ------------ Cluster Prefetch Statistics ------------ */
/** ------------------------------------------------------ */


break;

    case 2:
        //Case 2: Cache does not contain item, but cache is full
        //Actions:
        //1) Remove LFU item from LFUCache
        //2) Also remove LFU item from LFU replica & LFU time; if they exist on the cache
        //3) Place the new item on the LFUCache
        //4) Update the LFU Replica & LFU Time

        //Remove the Least frequent Item
        removeLeastFrequentItem(theItemToBeCached);

        //Initialize the requested Item Size to One
        itemCount = new Integer(Configuration.ONE);

        //Get the current Time from the System
        currentTime = new Long(System.currentTimeMillis());

        //LFUCacheMap - Item to be cached and Item Count
        lFUCacheMap.put(itemToBeCached, itemCount);

        //Log the Time at which the item was pushed on to the cache
        cacheTimeMap.put(itemToBeCached, currentTime);

        //Record the Entry into the Cache Map
        //cacheMap.put(itemToBeCached, new Long(theItemToBeCached.requestedItemSize));
        cacheMap.put(itemToBeCached, requestedItemSize);
```

```
                                        //Increment the Cache Miss by One
                                        cacheMisses += Configuration.ONE;

                                        //totalCacheMissByteSize += theItemToBeCached.requestedItemSize;
                                        totalCacheMissByteSize = totalCacheMissByteSize + requestedItemSize;

                                        //Update the Current Cache Size to reflect the current cache Size
                                        currentCacheSize = currentCacheSize
                                                        + requestedItemSize;
                                        /*currentCacheSize = currentCacheSize
                                                                        + theItemToBeCached.requestedItemSize;*/

                                        /** ------------------------------------------------------ */
                                        /** ------------ Cluster Prefetch Statistics ------------ */
                                        /** ------------------------------------------------------ */
                                        if(clusterCache.contains(itemToBeCached))
                                        {
                                                //clusterCache.remove(itemToBeCached);
                                                //numberOfClusterPrefetchedRemoved = numberOfClusterPrefetchedRemoved
+ 1;
                                                numberOfClusterPrefetchMisses = numberOfClusterPrefetchMisses + 1;
                                                sizeOfItemsPrefetchedMiss = sizeOfItemsPrefetchedMiss +
theItemToBeCached.getRequestedItemSize();
                                                sizeOfItemsPrefetchedRemoved = sizeOfItemsPrefetchedRemoved +
theItemToBeCached.getRequestedItemSize();
                                        }
                                        /** ------------------------------------------------------ */
                                        /** ------------ Cluster Prefetch Statistics ------------ */
                                        /** ------------------------------------------------------ */

                                        break;

                                case 4:
                                        //Case 4: Cache does not contain item & Cache is not full
                                        //Actions:
                                        //1) Push the Item on the LFU Queue
                                        //2) Register Item on the LFU Replica
                                        //3) Register Item on the LFUCacheTime
                                        //4) Also If there are any other LFU items in the leastFrequentMap, lFUCacheReplica
remove those items

                                        //Count of the Item
                                        itemCount = new Integer(Configuration.ONE);

                                        //Get the current Time
                                        currentTime = new Long(System.currentTimeMillis());

                                        //LFUCacheMap - Item to be cached and Frequency
                                        lFUCacheMap.put(itemToBeCached, itemCount);

                                        //CacheTimeMap - Item to be cached and the Current Time at which the Cache was filled
                                        cacheTimeMap.put(itemToBeCached, currentTime);

                                        //Log the Item to be cached in the CacheMap - Item to be cached & Size of the Item
                                        //cacheMap.put(itemToBeCached,new Long(theItemToBeCached.requestedItemSize));
                                        cacheMap.put(itemToBeCached,requestedItemSize);

                                        //Increment the Cache Misses
                                        cacheMisses += Configuration.ONE;

                                        //Increment the Cache Miss Size parameter
                                        totalCacheMissByteSize = totalCacheMissByteSize + requestedItemSize;
```

```
                                        //totalCacheMissByteSize += theItemToBeCached.requestedItemSize;

                                        //Update the Current Cache Size
                                        currentCacheSize = currentCacheSize + requestedItemSize;
                                        /*currentCacheSize = currentCacheSize + theItemToBeCached.requestedItemSize;*/

                                        /** ------------------------------------------------------ */
                                        /** ------------ Cluster Prefetch Statistics ------------ */
                                        /** ------------------------------------------------------ */
                                        if(clusterCache.contains(itemToBeCached))
                                        {
                                                //clusterCache.remove(itemToBeCached);
                                                //numberOfClusterPrefetchedRemoved = numberOfClusterPrefetchedRemoved
+ 1;
                                                numberOfClusterPrefetchMisses = numberOfClusterPrefetchMisses + 1;
                                                sizeOfItemsPrefetchedMiss = sizeOfItemsPrefetchedMiss +
theItemToBeCached.getRequestedItemSize();
                                        }
                                        /** ------------------------------------------------------ */
                                        /** ------------ Cluster Prefetch Statistics ------------ */
                                        /** ------------------------------------------------------ */


                                        break;

                                        /*case 3:
                                        //Case 3: Cache Already contains item & Cache is not full
                                        //Actions:
                                        //1) Get the existing Item say 'X' from the main queue and Update its count
                                        //2) If the item 'X' exists on the leastFrequentItemMap remove it
                                        //3) If the item 'X' exists on the lFUCacheEntryTime remove it

                                        itemCount = lFUCacheMap.get(itemToBeCached);

                                        itemCount +=CacheConstants.ONE;

                                        lFUCacheMap.put(itemToBeCached, itemCount);

                                        currentTime = new Long(System.currentTimeMillis());

                                        cacheTimeMap.put(itemToBeCached, currentTime);

                                        cacheHits += CacheConstants.ONE;

                                        totalCacheHitByteSize += theItemToBeCached.requestedItemSize;

                                        break;*/

                                default:

                                        break;

                        }

                        totalRequestedItemSize = totalRequestedItemSize + requestedItemSize;
                        totalNumberOfItems = totalNumberOfItems + Configuration.ONE;
        }

        public Long getNumberOfClusterPrefetchMisses() {
                return numberOfClusterPrefetchMisses;
        }
```

```
public void setNumberOfClusterPrefetchMisses(Long numberOfClusterPrefetchMisses) {
	this.numberOfClusterPrefetchMisses = numberOfClusterPrefetchMisses;
}

public Long getNumberOfClusterPrefetchedRemoved() {
	return numberOfClusterPrefetchedRemoved;
}

public void setNumberOfClusterPrefetchedRemoved(
		Long numberOfClusterPrefetchedRemoved) {
	this.numberOfClusterPrefetchedRemoved = numberOfClusterPrefetchedRemoved;
}

/**
 * Computes which use case is to be executed
 *
 * @param itemToBeCached
 * @return
 */
private int computeCase(Request theItemToBeCached)
{

	Long tempCacheSize = null;
	String itemToBeCached = null;

	//Object to be cached
	itemToBeCached = theItemToBeCached.getRequestedObject();

	Long itemToBeCachedSize = new Long(theItemToBeCached.getRequestedItemSize());

	//Get the temporary Cache Size
	tempCacheSize = currentCacheSize + itemToBeCachedSize;

	//if(lFUCacheMap.containsKey(itemToBeCached) && lFUCacheMap.size() >=capacity )
	//if(lFUCacheMap.containsKey(itemToBeCached) && tempCacheSize >= maxCacheSize)
	//if(lFUCacheMap.containsKey(itemToBeCached) && tempCacheSize > maxCacheSize)
	if(lFUCacheMap.containsKey(itemToBeCached))
	{//Case 1: Cache contains item and cache is full
		return Configuration.ONE;
	}
	//else if(!lFUCacheMap.containsKey(itemToBeCached)&& lFUCacheMap.size() >=capacity)
	//else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize >= maxCacheSize)
	else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize > maxCacheSize)
	{//Case 2: Cache does not contain item, but cache is full
		return Configuration.TWO;
	}
	//else if(lFUCacheMap.containsKey(itemToBeCached)&&lFUCacheMap.size() < capacity)
	//else if(lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize < maxCacheSize)
	/*else if(lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize <= maxCacheSize)
	{//Case 3: Cache Already contains item & Cache is not full
		return CacheConstants.THREE;
	}*/
	//else if(!lFUCacheMap.containsKey(itemToBeCached)&&lFUCacheMap.size()<capacity)
	//else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize < maxCacheSize)
	else if(!lFUCacheMap.containsKey(itemToBeCached)&& tempCacheSize <= maxCacheSize)
	{//Case 4: Cache does not contain item, but cache is not full
		return Configuration.FOUR;
	}

	return Configuration.ZERO;
}
```

```
/**
 * Helper function to remove the least frequent Item
 * <br/>
 * <b>
 *          <br/>       In case there are multiple items that are least frequently occurring
 *  <br/>   remove the item that was first inserted into the cache in terms of the
 *  <br/>   time it was entered in the cache
 * </b>
 *
 */
public void removeLeastFrequentItem(Request theRequestedItem)
{
            Integer leastCount=null;

            //Variable to get the first key or least count
            Long firstKey = null;

            //The least frequent item reference object
            String leastFrequentItem = null;

            Runtime r = null;

            Long entryTime = null;

            //Set of Least Frequently Used Item's values
            Collection<Integer> countValues = null;

            //Set of Least Frequently Used Item's values
            Set<String> cachedItemSet = null;

            SortedMap<Long, String> sortedMap = null;

            TreeSet<Integer> countValuesSet = null;

            // Stores the current Cache Size after adding
            // current Cache size and size of item to be cached
            Long tempCacheSize = null;

            //Size of the Item to be removed
            Long itemToBeRemovedSize = null;

            Long theRequestedItemSize = new Long(theRequestedItem.getRequestedItemSize());

            /*tempCacheSize = currentCacheSize
                                                + theRequestedItem.requestedItemSize;*/
            tempCacheSize = currentCacheSize
                                                                    + theRequestedItemSize;

            /*****************************************************/
            // Remove Item from the cache till the cache has enough space to
            // accommodate the item to be cached
            /*****************************************************/
            //while(tempCacheSize >= maxCacheSize)
            while(tempCacheSize > maxCacheSize)
            {
                        // ++++++++++++++++++++++++++++++++++++++++++
                        // Traverse over the counts of items in
                        // the Map to get the Least Count
                        // ++++++++++++++++++++++++++++++++++++++++++

                        //Set of Least Frequently Used Item's values
                        countValues = lFUCacheMap.values();
```

```
//Set of actual Cached Items items
cachedItemSet = lFUCacheMap.keySet();

//Sorted Map
sortedMap = new TreeMap<Long, String>();

//TreeSet - To Hold Least Frequent count for Items in sorted order
countValuesSet = new TreeSet<Integer>();

/*** ********************************** */
// 1. -- Add the various count values
//        for all items in the Cache  --
/*** ********************************** */
countValuesSet.addAll(countValues);

/*** ********************************** */
//Get the Least Count value
/*** ********************************** */
try
{
        leastCount = countValuesSet.first();
}
catch(Exception ex)
{
        System.out.println("Exception");
        System.out.println(this.totalNumberOfItems);
}

/*** ********************************** */
// 2. --- Get all the Items who have least count ---
// 3. --- Create a Map of
//    (Entry Time for the Least Frequent Item, Least Frequent Item itself)
/*** ********************************** */
for (String string : cachedItemSet)
{
        // Get the items that has leastCount
        if(lFUCacheMap.get(string) == leastCount)
        {
                entryTime = cacheTimeMap.get(string);
                sortedMap.put(entryTime, string);
        }
}

/*** ********************************** */
// 3. --- Get all the Item who have least count
//                          & inserted first into the Cache ---
/*** ********************************** */
firstKey = sortedMap.firstKey();

if(firstKey==null)
{
        System.out.println("First Key Empty");
}

/*** ********************************** */
//4. --- Get the least frequent Item ---
/*** ********************************** */
leastFrequentItem = sortedMap.get(firstKey);

//System.out.println("Throwing out + " + leastFrequentItem);
```

```
                        /*System.out.println("Removing :: " + leastFrequentItem + " Current Cache Size :" +
this.currentCacheSize);*/

                        //Get the Item to be removed's size in bytes
                        itemToBeRemovedSize = cacheMap.get(leastFrequentItem);

                        //Remove the Item from the Cache Map
                        // - A map of item and size of the item
                        cacheMap.remove(leastFrequentItem);

                        //Remove the item from the lfu cache map
                        // - A map of item and number of times it was requested
                        lFUCacheMap.remove(leastFrequentItem);

                        //Remove the item from the cacheTimeMap
                        // - A map of item and the time it was inputed into the cache
                        cacheTimeMap.remove(leastFrequentItem);

                        //Edit the current Cache Size
                        currentCacheSize = currentCacheSize - itemToBeRemovedSize;

                        /** ------------------------------------------------------ */
                        /** ------------ Cluster Prefetch Statistics ------------ */
                        /** ------------------------------------------------------ */
                        if(clusterCache.contains(leastFrequentItem))
                        {
                                clusterCache.remove(leastFrequentItem);
                                numberOfClusterPrefetchedRemoved = numberOfClusterPrefetchedRemoved + 1;
                                sizeOfItemsPrefetchedRemoved = sizeOfItemsPrefetchedRemoved +
itemToBeRemovedSize;

                                Boolean b = clustCache.get(leastFrequentItem);

                                if(!b)
                                {
                                        cacheMissList.add(leastFrequentItem);
                                }

                                clustCache.remove(leastFrequentItem);
                        }
                        /** ------------------------------------------------------ */
                        /** ------------ Cluster Prefetch Statistics ------------ */
                        /** ------------------------------------------------------ */


                        // ++++++++++++++++++++++++++++++
                                /*****************************/
                                // Initialize the local variables to null
                                /*****************************/
                        // ++++++++++++++++++++++++++++++
                        cachedItemSet = null;

                        sortedMap = null;

                        firstKey = null;

                        itemToBeRemovedSize = null;

                        leastCount = null;

                        leastFrequentItem = null;
```

```
                        sortedMap = null;

                        countValuesSet = null;

                        countValues = null;
                        // ++++++++++++++++++++++++++++++

                        // Compute the tempCacheSize to make sure the size after inserting
                        // the item to be cached the cache size does not overshoot.
                        tempCacheSize = currentCacheSize + theRequestedItemSize;

                        //tempCacheSize = currentCacheSize + theRequestedItem.requestedItemSize;
                }
                /*******************************************************/
                //While Ends
                /*******************************************************/

                r = Runtime.getRuntime();

                // ++++++++++++++++++++++++++++++
                // Try to do a garbage collection
                // ++++++++++++++++++++++++++++++
                try
                {
                        // +++++++++++++++++++++++++
                        // Invoke Garbage Collection
                        // +++++++++++++++++++++++++
                        r.gc();

                        // ++++++++++++++++++++++++++++++++++++++++++++
                        // Put the currently executing thread to sleep
                        // ++++++++++++++++++++++++++++++++++++++++++++
                        Thread.sleep(100);

                        // +++++++++++++++++++++++++
                        // Invoke Garbage Collection
                        // +++++++++++++++++++++++++
                        r.gc();
                }
                catch (InterruptedException e)
                {
                        e.printStackTrace();
                }
        }


}
package com.sjsu.edu.fileprocessor;

import java.io.BufferedReader;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URLConnection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
```

```
import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.node.Node;

public class FileProcessor
{
	private String fileName;
	private String clientIP;
	private String urlDomainName;

	/**
	 * 1. Get the Client IP Address
	 * 2. Get the URL base name or domain name
	 * 3. Check if the the URL occurs in any of the Clusters for the Client IP Address
	 * 4. If the URL Occurs in the any of the Clusters, get the Intrasite Cluster for the URL
	 * 5. If the Requested Item occurs in the Intrasite Cluster for that URL, load this intrastie
	 *		cluster into the Cache. Load other objects for other URLs also into the Cache
	 * 6. If the requested URL item does not occur in the cluster, discard the intrasite cluster
	 *		and also the intersite cluster.
	 *
	 * @throws IOException
	 *
	 */

	public Map<Node,Map<String,Long>> processRequest(Request request) throws IOException
	{
		//Map<String,Long> intraSite = processInterSite(request);

		Map<Node,Map<String,Long>> consolidatedCluster = processInterSite(request);

	/*		if(intraSite!=null)
				loadAllInterSiteClusters(intraSite,request);*/

		return consolidatedCluster;


	}

	/**
	 *
	 * @param request
	 * @return
	 * @throws IOException
	 */
	private Map<Node,Map<String,Long>> processInterSite(Request request) throws IOException
	{

		Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

		Set<String> currentDomainURLs = null;

		//
		String currentClusterFilePath = null;

		//
		File currentFilePath = null;

		Set<String> currentDomainURL = null;

		//Client IP Address
		String clientIPAddress = request.getClientIPAddress();
```

```
		//Domain Name
		String URLDomainName = request.getDomainName();

		//Client IP Intersite Directory
		String clientIPInterSiteDir = Configuration.cIPIntersiteClusterDir + "\\" + clientIPAddress;

		File directoryPath = new File(clientIPInterSiteDir);

		if(directoryPath.exists())
		{
			String[] fileDirectories = directoryPath.list();

			currentDomainURLs = new HashSet<String>();

			for (String file : fileDirectories)
			{
				currentClusterFilePath = clientIPInterSiteDir + "\\" + file;

				FileInputStream fstream = new FileInputStream(currentClusterFilePath);

				// Get the object of DataInputStream
			   DataInputStream in = new DataInputStream(fstream);

			   BufferedReader br = new BufferedReader(new InputStreamReader(in));

			   String strLine;

			   //Read File Line By Line
			   while ((strLine = br.readLine()) != null)
			   {
			     currentDomainURLs.add(strLine.trim());
			   }
			   //Close the input stream
			   in.close();

			   if(currentDomainURLs.contains(URLDomainName))
			   {
				mapOfInterSiteNodes = processIntraSite(currentDomainURLs,request);
				break;
			   }
			   else
			   {
				mapOfInterSiteNodes = null;
				currentDomainURLs = null;
				currentDomainURLs = new HashSet<String>();
				continue;
			   }
			}
		}

		return mapOfInterSiteNodes;
	}

	/**
	 * Process Intra site Cluster
	 *
	 * @param currentDomainURLs
	 * @param request
	 * @throws IOException
	 */
```

```
	private Map<Node,Map<String,Long>> processIntraSite(Set<String> currentDomainURLs,Request request) throws
IOException
	{
		Map<String,Long> intraSiteMap = null;

		Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

		String currentURLDomain = request.getDomainName();

		String clientIPAddress = request.getClientIPAddress();

		String currentClusterFilePath = Configuration.cIPIntrasiteClusterDir
                                                                + "\\" + clientIPAddress + "\\" +
currentURLDomain + "\\" + currentURLDomain + ".txt";

		File file = new File(currentClusterFilePath);

		if(file.exists())
		{
			FileInputStream fstream = new FileInputStream(currentClusterFilePath);

			// Get the object of DataInputStream
		   DataInputStream in = new DataInputStream(fstream);

		   BufferedReader br = new BufferedReader(new InputStreamReader(in));

		   String strLine;

		   String requestedItem = null;

		   intraSiteMap = new HashMap<String, Long>();

		   //Read File Line By Line
		   while ((strLine = br.readLine()) != null)
		   {
		    String[] elements = strLine.split(",");

		    Long size = new Long(elements[1].trim());

		    intraSiteMap.put(elements[0].trim(), size);

		   }

		   //Close the input stream
		   in.close();

		   requestedItem = request.getRequestedObject();

		   if(!intraSiteMap.containsKey(requestedItem))
		   {
			intraSiteMap = null;
		   }
		   else
		   {
			mapOfInterSiteNodes = loadAllInterSiteClusters(intraSiteMap, request,currentDomainURLs);
		   }
		}

	   return mapOfInterSiteNodes;

	}
```

```
private Map<Node,Map<String,Long>> loadAllInterSiteClusters(Map<String,Long> intraSiteCluster,
                        Request request,Set<String> currentDomainURLs) throws IOException
{
        Set<String> currDomainURLs = new HashSet<String>();

        Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

        Node interSiteNode = null;

        Map<String,Long> webOjects = null;

        String currentClusterFilePath = null;

        String[] elements = null;

        Long size = null;

        //Client IP Address
        String clientIPAddress = request.getClientIPAddress().trim();

        currDomainURLs.addAll(currentDomainURLs);

        currDomainURLs.remove(request.getDomainName().trim());

        //Load the first Intersite Node
        interSiteNode = new Node();
        interSiteNode.setURL(request.getDomainName().trim());

        //Intialize map of Intersite Nodes
        mapOfInterSiteNodes = new HashMap<Node, Map<String,Long>>();
        mapOfInterSiteNodes.put(interSiteNode, intraSiteCluster);

        for (String currentURL : currDomainURLs)
        {
                currentClusterFilePath = Configuration.cIPIntrasiteClusterDir + "\\" + clientIPAddress
                        + "\\" + currentURL + "\\" + currentURL + ".txt";

                File f = new File(currentClusterFilePath);

                if(f.exists())
                {
                        FileInputStream fstream = new FileInputStream(currentClusterFilePath);

                        // Get the object of DataInputStream
                   DataInputStream in = new DataInputStream(fstream);

                   BufferedReader br = new BufferedReader(new InputStreamReader(in));

                   String strLine;

                   webOjects = new HashMap<String, Long>();

                   interSiteNode = new Node();

                   interSiteNode.setURL(currentURL);

                   //Read File Line By Line
                   while ((strLine = br.readLine()) != null)
                   {
                     elements = strLine.trim().split(",");
                     size = new Long(elements[1].trim());
                     webOjects.put(elements[0].trim(), size);
```

```
                                }
                                //Close the input stream
                                in.close();

                                mapOfInterSiteNodes.put(interSiteNode, webOjects);
                            }
                        }

                        return mapOfInterSiteNodes;

                }

}
package com.sjsu.cache.lru;

import java.util.HashMap;


import java.util.HashSet;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;

/**
 * This is implementation of the LRU Cache<br/>
 * @author <b>Akshay Shenoy</b><br/>
 *
 * @copyrights <b>reserved</b> by Akshay Shenoy
 *
 */
public class LRUCache {

        // Main Cache Map that holds the Cache of items
        //private Map<String,String> cacheMap;

        // Main Cache Map that holds the Cache of items
        private Map<String,Long> cacheMap;

        // Holds a map of what item was requested how many times*/
        /*private Map<K, Long> cacheRequestMap;*/

        // Counter for the Number of Cache Hits
        private Long cacheHits;

        // Counter for the Number of Cache Misses
        private Long cacheMiss;

        // Total Number of Items that were requested from the Cache
        private Long totalNumberOfItems;

        // Tracks the Size of all the items that were requested for the Cache
        private Long totalRequestedItemSize;

        // Tracks the Size of all the items that were requested from Cache and available in the Cache
        private Long totalCacheHitSize;

        // Tracks the Size of all Cache items that were requested Cache did not have
        private Long totalCacheMissSize;
```

```
private LinkedList<String> requestedItemQueue;
private int capacity;
private Long currentCacheSize;
private Long maxCacheSize;

public LRUCache(Long size)
{
	requestedItemQueue = new LinkedList<String>();
	//cacheMap = new HashMap<String, Long>(capacity);
	cacheMap = new HashMap<String, Long>();
	cacheHits = new Long(Configuration.ZERO);
	cacheMiss= new Long(Configuration.ZERO);
	totalNumberOfItems = new Long(Configuration.ZERO);
	totalRequestedItemSize = new Long(Configuration.ZERO);
	totalCacheHitSize = new Long(Configuration.ZERO);
	totalCacheMissSize = new Long(Configuration.ZERO);
	maxCacheSize = size;
	currentCacheSize = new Long(0);
}

public Long getMaxCacheSize() {
	return maxCacheSize;
}

public Long getCurrentCacheSize() {
	return currentCacheSize;
}

public void cacheItem(Request theRequestedItem)
{
	addItem(theRequestedItem);
}

/**
 *
 * @param requestedItem : This is the item that is requested from the Cache
 */
private void addItem(Request theRequestedItem)
{
	//String requestedItem = theRequestedItem.getRequestedObject();
	String requestedItem = theRequestedItem.getRequestedURL();

	int caseId = computeCase(theRequestedItem);

	switch (caseId)
	{
		case 2: // Case 2 : Cache is Full && Item to be cached is not duplicated ---> Cache Miss

			//1) Remove the LRU item from the LRUQueue
			//itemToBeRemoved = requestedItemQueue.remove();

			//2) Remove the LRU item from the cacheMap
			//cacheMap.remove(itemToBeRemoved);


			//Edit the Cache
			editCache(theRequestedItem);

			//3) Add the new Item to be cached at end of LRUQueue
			requestedItemQueue.addLast(requestedItem);
```

```
                        //4) Cache the new Item in the LRUQueue
                        cacheMap.put(requestedItem, new Long(theRequestedItem.getRequestedItemSize()));

                        //Update the parameter for the Cache Miss Size
                        totalCacheMissSize = totalCacheMissSize
                                                                +
theRequestedItem.getRequestedItemSize();

                        //Update the parameter for the number of Cache Misses
                        cacheMiss+=Configuration.ONE;

                        //Update the current Cache Size reflect the item added to the Cache
                        currentCacheSize = currentCacheSize
                                                                +
theRequestedItem.getRequestedItemSize();

                        break;

                case 1:  // Case 1: Cache is Full && Item to be cached is duplicated ---> Cache Hit

                        //1)Remove the item from requested LRU Queue
                        requestedItemQueue.remove(requestedItem);

                        //2)Put the item at the end for LRU Queue
                        requestedItemQueue.addLast(requestedItem);

                        //Update the Parameter for the total Cache Hit Size
                        totalCacheHitSize = totalCacheHitSize
                                                        + theRequestedItem.getRequestedItemSize();

                        //Update the parameter for the number of Cache Hits
                        cacheHits+=Configuration.ONE;


                        /*//Update the current Cache Size reflect the item added to the Cache
                        currentCacheSize = currentCacheSize
                                                        + theRequestedItem.requestedItemSize;*/

                        break;

                /*case 3: //Case 3: Cache is not Full && Item to be cached is duplicated ---> Cache Hit

                        //1) Remove the requestedItem from the LRUQueue
                        requestedItemQueue.remove(requestedItem);

                        //2) Add the item to end of LRUQUeue
                        requestedItemQueue.addLast(requestedItem);

                        //Update the parameter that keeps track of Total Cache Hit Size
                        totalCacheHitSize = totalCacheHitSize
                                                                + theRequestedItem.requestedItemSize;

                        //Update the parameter that tracks the number of Cache Hits
                        cacheHits+=CacheConstants.ONE;

                        //Since the item to be Cached is already in cache we do not need to update
                        // the current Cache Size

                        break;*/

                case 4: //Case 4: Cache is not Full && Item to be cached is not duplicated ---> Cache Miss
```

```
                                        //1) Add the new Item of end of LRUQueue
                                        requestedItemQueue.addLast(requestedItem);

                                        //2) cache the new Item in the cacheMap
                                        //cacheMap.put(requestedItem, requestedItem);

                                        //2) cache the new Item in the cacheMap
                                        cacheMap.put(requestedItem, new Long(theRequestedItem.getRequestedItemSize()));


                                        //Update the Parameter that keeps track of the Cache Miss Size
                                        totalCacheMissSize = totalCacheMissSize
                                                                + theRequestedItem.getRequestedItemSize();

                                        //Update the parameter that keeps track of the number
                                        //of Cache Misses
                                        cacheMiss+=Configuration.ONE;

                                        //Update the current Cache Size reflect the item added to the Cache
                                        currentCacheSize = currentCacheSize
                                                                + theRequestedItem.getRequestedItemSize();

                                        break;

                                case 0: // default case
                                        System.out.println("Case 0 : LRUCache 98 -> addItem()");
                                        break;

                                default: //default case
                                        System.out.println("Defualt case: LRUCache 102-> addItem()");
                                        break;
                        }

                        totalRequestedItemSize = totalRequestedItemSize + theRequestedItem.getRequestedItemSize();
                        totalNumberOfItems +=Configuration.ONE;
                }

                /**
                 * Computes the Use Case to Be Executed
                 *
                 * @param theRequestedItem
                 * @return
                 */
                //private int computeCase(String logEntry)
                private int computeCase(Request theRequestedItem)
                {
                        String logEntry = theRequestedItem.getRequestedURL();

                        Long tempSize = theRequestedItem.getRequestedItemSize()
                                                                                                        +
currentCacheSize;

                        //if((cacheMap.size() >= maxCacheSize) && (requestedItemQueue.contains(logEntry)))
                        //if((tempSize >= maxCacheSize) && (requestedItemQueue.contains(logEntry)))
                        //if((tempSize > maxCacheSize) && (requestedItemQueue.contains(logEntry)))
                        if((requestedItemQueue.contains(logEntry)))
                        {       // Case 1: Cache is Full && Item to be cached is duplicated
                                return Configuration.ONE;
                        }
                        //else if((cacheMap.size() >= capacity) && (!requestedItemQueue.contains(logEntry)))
                        //else if((tempSize >= maxCacheSize) && (!requestedItemQueue.contains(logEntry)))
                        else if((tempSize > maxCacheSize) && (!requestedItemQueue.contains(logEntry)))
```

```
		{		// Case 2 : Cache is Full && Item to be cached is not duplicated
			return Configuration.TWO;
		}
		//else if((cacheMap.size() < capacity) && (requestedItemQueue.contains(logEntry)))
		//else if((tempSize < maxCacheSize) && (requestedItemQueue.contains(logEntry)))
		/*else if((tempSize <= maxCacheSize) && (requestedItemQueue.contains(logEntry)))
		{		//Case 3: Cache is not Full && Item to be cached is duplicated
			return CacheConstants.THREE;
		}*/
		//else if((cacheMap.size()< capacity) && (!rwequestedItemQueue.contains(logEntry)))
		//else if((tempSize < maxCacheSize) && (!requestedItemQueue.contains(logEntry)))
		else if((tempSize <= maxCacheSize) && (!requestedItemQueue.contains(logEntry)))
		{		//Case 4: Cache is not Full && Item to be cached is not duplicated
			return Configuration.FOUR;
		}

		return Configuration.ZERO;
}

/**
 *
 * Edits the Cache to reduce the size to include the new item
 * to be cached
 *
 */
private void editCache(Request theRequestedItem)
{
		Long tempSize = null;
		String theFirstItem = null;
		Long currentItemSize = null;

		tempSize = currentCacheSize + theRequestedItem.getRequestedItemSize();

		while(tempSize > maxCacheSize)
		{
			if(requestedItemQueue.size() > 0)
			{
				theFirstItem = requestedItemQueue.remove();

				currentItemSize = cacheMap.get(theFirstItem);

				cacheMap.remove(theFirstItem);

				currentCacheSize = currentCacheSize - currentItemSize;

				tempSize = currentCacheSize + theRequestedItem.getRequestedItemSize();

			}
			else
			{
				break;
			}
		}

}

public Long getCacheHits() {
		return cacheHits;
}

public Long getCacheMiss() {
		return cacheMiss;
```

```
	}

	public Long getTotalNumberOfItems() {
		return totalNumberOfItems;
	}

	public Long getTotalRequestedItemSize() {
		return totalRequestedItemSize;
	}

	public Long getTotalCacheHitSize() {
		return totalCacheHitSize;
	}

	public Long getTotalCacheMissSize() {
		return totalCacheMissSize;
	}

	public int getCacheMapSize()
	{
		return cacheMap.size();
	}

	/**
	 *
	 * 1)Check if the Item already exists in the cache.<br/>
	 * 2)If exists then, remove the item from the Queue of Cache that is handle<br/>
	 *		of cache entries in temporal order and add the item at end of the queue<br/>
	 *		to keep the LRU item queue updated
	 *
	 * @param logEntry :
	 * @return
	 *//*
	protected boolean checkItemExistsInCache(String logEntry)
	{
		if(requestedItemQueue.contains(logEntry) && cacheMap.containsKey(logEntry))
		{
			requestedItemQueue.remove(logEntry);
			requestedItemQueue.add(logEntry);
			return true;
		}
		else
			return false;
	}*/
}
package com.sjsu.cache.lru.test;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.lrulogreader.LRULogReader;


public class LRUTest {

	/**
	 * @param args
	 */
	public static void main(String[] args)
	{
		//Main Testing Content - That would throw error on LFU
		String inputFile = Configuration.dataSet4Path;

		LRULogReader theLRULogReader = new LRULogReader(Configuration.dataSet4CacheSize);
```

```
                String outputFile = Configuration.resultsPath;

                theLRULogReader.executeRequest(inputFile,outputFile);

        }

}
package com.sjsu.edu.lrulogreader;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import com.sjsu.cache.helper.Request;
import com.sjsu.cache.lru.LRUCache;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.printer.Printer;

public class LRULogReader {

        private Integer maxItemSize;
        private     String reqItem;
        private Long requestNumber;


        //private LRUCache<String, String> theLRUCache;
        private LRUCache theLRUCache;

        //public LRULogReader(Integer capacity)
        public LRULogReader(Long capacity)
        {
                //theLRUCache = new LRUCache<String, String>(capacity);
                theLRUCache = new LRUCache(capacity);
                maxItemSize = new Integer(0);
                reqItem = null;
                requestNumber = new Long(0);
        }

        public Integer getMaxItemSize() {
                return maxItemSize;
        }

        public void executeRequest(String fileName,String outputFileName)
        {
                readLogFile(fileName,outputFileName);
        }

        @SuppressWarnings({ "deprecation"})
        private void readLogFile(String inputFile,String outputFile)
        {

                 Long dataViolatingSize = new Long(0);

                 Integer numberOfItemsViolatingSize = new Integer(0);
```

```
//Create File from which to read
File fileInput = new File(inputFile);

//Create File to which to write
File fileOutput = new File(new String(outputFile));

//The FileOutputStream
FileOutputStream fos = null;

//The FileInputStream
FileInputStream fis = null;

long numberOfItems = 0;

//Number of Requests
long noOfRequests = Configuration.ZERO;

//Runtime Object
Runtime r = Runtime.getRuntime();

try{

	//Make sure the File exists - To prevent File not found exception
	if(fileInput.exists()){

		//FileInputStream
		fis = new FileInputStream(fileInput);

		//FileOutputStream
		fos = new FileOutputStream(fileOutput);

		//BufferedOutputStream
		BufferedOutputStream bos = new BufferedOutputStream(fos);

		//BufferedInputStream
		BufferedInputStream bis = new BufferedInputStream(fis);

		//DataOutputStream
		DataOutputStream dos = new DataOutputStream(bos);

		//DataInputStream
		DataInputStream dis = new DataInputStream(bis);

		//while(dis.available()!=CacheConstants.ZERO)
		//while(noOfRequests < 366380 && dis.available()!=CacheConstants.ZERO)
		while(dis.available()!=Configuration.ZERO)
		{
			numberOfItems = numberOfItems + 1;

			//Read a line from the Web Log File
			String line = dis.readLine();

			if(numberOfItems < Configuration.seventyPercentDataSet4)
			{
				continue;
			}


		//Pre-process the request
		Request request = preProcessRequest(line);
```

```
                                        if(maxItemSize < request.getRequestedItemSize())
                                        {
                                                maxItemSize = request.getRequestedItemSize();
                                                reqItem = request.getRequestedURL();
                                                requestNumber = noOfRequests + 1;
                                        }

                                        if(!(request.getRequestedItemSize() > theLRUCache.getMaxCacheSize()))
                                        {
                                                //LRU Cache
                                                theLRUCache.cacheItem(request);

                                                //Increment the Total Number of requests
                                                noOfRequests +=Configuration.ONE;
                                        }
                                        else
                                        {
                                                dataViolatingSize = dataViolatingSize +  new
Long(request.getRequestedItemSize().toString());
                                                numberOfItemsViolatingSize = numberOfItemsViolatingSize + 1;
                                        }

                                        //Once the item is pushed to the Cache set the handle to null
                                        request = null;

                                        /*//Increment the Total Number of requests
                                        noOfRequests +=CacheConstants.ONE;*/

                                        //Every 25000 request clear out the JVM by trying to run GARBAGE Collector
                                        //if(noOfRequests%CacheConstants.TWENTYFIVETHOUSHAND
                                        if(noOfRequests%Configuration.TENTHOUSAND
                                                        ==Configuration.ZERO)
                                        {
                                                try
                                                {
                                                        //Invoke GARBAGE Collector
                                                        GarbageCollectorHelper.invokeGC();


                                                        Printer.printString("Analyzing Requests,Total Number of
Items Scanned :" + theLRUCache.getTotalNumberOfItems() + ",Size of All items in Cache :"
                                                                        + theLRUCache.getCurrentCacheSize()
+ " bytes, Number of Items in the Cache :" + theLRUCache.getCacheMapSize()
                                                                        );

                                                        //Put the Thread to sleep
                                                        //Thread.sleep(CacheConstants.TWOTHOUSAND);

                                                        //Invoke GARBAGE Collector
                                                        GarbageCollectorHelper.invokeGC();

                                                }
                                                catch (Exception e)
                                                {
                                                        e.printStackTrace();
                                                }
                                        }

                                }

                                printLRUStats();
```

```
                                        //System.out.println("Number of Items violating Size :" +
numberOfItemsViolatingSize);
                                        //System.out.println("Size of Items violationg Size : " + dataViolatingSize );
                                        //Attempt a write to file
                                        //dos.writeBytes("Total Hits : " + theLRUCache.getCacheHits());
                                }

                        }
                        catch (IOException e)
                        {
                                e.printStackTrace();
                        }
                        finally
                        {
                                try
                                {
                                        fis.close();
                                        fos.close();
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                }
                        }

                        /*  System.out.println("Max Item Size :" + maxItemSize +
                                        ",Request Number:" + this.requestNumber + ",Requested Item:" + this.reqItem);*/
        }

        private void printLRUStats()
        {
                //Once the Complete Processing of all Web Log Entries is done - Print out all the details
                System.out.println("Total Number of Items : "  + theLRUCache.getTotalNumberOfItems());
                System.out.println("Total Misses : " + theLRUCache.getCacheMiss());
                System.out.println("Total Hits : " + theLRUCache.getCacheHits());
                System.out.println("Total Size of All Items Analyzed : " + theLRUCache.getTotalRequestedItemSize() + "
bytes");
                //System.out.println("Cache Map Size : "  + theLRUCache.getCacheMapSize());
                System.out.println("Total Cache Hit Size : " + theLRUCache.getTotalCacheHitSize());
                System.out.println("Total Cache Miss Size : " + theLRUCache.getTotalCacheMissSize());

        }

        private Request preProcessRequest(String stringRequest)
        {
                String[] requestParam = stringRequest.split(" ");
                //int i=0;
                Request request = new Request();

                /*Long requestTime = new Long(requestParam[0].toString());*/
                String requestTimeString = requestParam[0].toString();

                int index = requestTimeString.indexOf(".");

                String reqString = requestTimeString.substring(Configuration.ZERO, index);

                Long requestTime = new Long(reqString);
                Long requestProcessingTime = new Long(requestParam[1]);
                String clientIPAddress = requestParam[2].trim();
                String tcpCode = requestParam[3].trim();
                Integer requestedItemSize = new Integer(requestParam[4]);
                String requestedObject = requestParam[6].trim();
```

```
		String timeOutRedirectionAttribute = requestParam[8].trim();
		String requestedObjectType = requestParam[9].trim();

		String requestedURL = null;

		requestedURL = processURL(requestedObject).trim();

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
		request.setRequestedObjectType(requestedObject);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);
		request.setRequestedURL(requestedURL);

		requestParam = null;

		return request;
	}


	private String processURL(String url)
	{
		String URL = url;


		//URL : http://www.gmail.com/skd/
		URL = URL.trim();

		int index = URL.indexOf("//");
		int index2 = -1;

		//www.gmail.com/skd/
		URL = URL.substring(index + Configuration.TWO);

		return URL;

/*		index2 = URL.indexOf("/");

		if(index2>-1)
		{
			URL =
				URL.substring(CacheConstants.ZERO,index2);
		}
		else
		{
			URL =
				URL.substring(CacheConstants.ZERO);
		}

		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");

			URL =
				URL.substring(CacheConstants.ZERO,index3);
		}
*/
		//return URL;
	}
```

```
}
package com.sjsu.cache.lru.test;

import com.sjsu.cache.helper.HybridLRULogReader;
import com.sjsu.edu.commonconstants.Configuration;


public class HybridLRUTest {

	/**
	 * @param args
	 */
	public static void main(String[] args) {

		//Main Testing Content - That would throw error on LFU String
		 String inputFile = Configuration.dataSet4Path;

		 HybridLRULogReader theLRULogReader = new
HybridLRULogReader(Configuration.dataSet4CacheSize);

		/* LRULogReader theLRULogReader = new LRULogReader(new Long(77875413)); */

		/* String inputFile = "D:\\CustomizedLogFile\\2jz.org\\2jz.org.txt"; */

		String outputFile = new String("E:\\CentralServerLogDump\\results2.txt");

		// LRULogReader theLRULogReader = new
		// LRULogReader(CacheConstants.Seven916);

		theLRULogReader.executeRequest(inputFile, outputFile);

		// System.out.println("Max Item Size: " +
		// theLRULogReader.getMaxItemSize().toString());

	}

}
package com.sjsu.cache.helper;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import com.sjsu.cache.lru.HybridLRUCache;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.printer.Printer;

public class HybridLRULogReader {

	private Integer maxItemSize;
	private	String reqItem;
	private Long requestNumber;
```

```
//private LRUCache<String, String> theLRUCache;
private HybridLRUCache theLRUCache;

//public LRULogReader(Integer capacity)
public HybridLRULogReader(Long capacity)
{
	//theLRUCache = new LRUCache<String, String>(capacity);
	theLRUCache = new HybridLRUCache(capacity);
	maxItemSize = new Integer(0);
	reqItem = null;
	requestNumber = new Long(0);
}

public Integer getMaxItemSize()
{
	return maxItemSize;
}

public void executeRequest(String fileName,String outputFileName)
{
	readLogFile(fileName,outputFileName);
}

@SuppressWarnings({ "deprecation"})
private void readLogFile(String inputFile,String outputFile)
{

	 Long dataViolatingSize = new Long(0);

	 Integer numberOfItemsViolatingSize = new Integer(0);

	 //Create File from which to read
	 File fileInput = new File(inputFile);

	 //Create File to which to write
	 File fileOutput = new File(new String(outputFile));

	 //The FileOutputStream
	 FileOutputStream fos = null;

	 //The FileInputStream
	 FileInputStream fis = null;

	 long numberOfItems = 0;

	 //Number of Requests
	 long noOfRequests = Configuration.ZERO;

	 //Runtime Object
	 Runtime r = Runtime.getRuntime();

	 try{

		//Make sure the File exists - To prevent File not found exception
		if(fileInput.exists()){

			//FileInputStream
			fis = new FileInputStream(fileInput);

			//FileOutputStream
			fos = new FileOutputStream(fileOutput);
```

```
//BufferedOutputStream
BufferedOutputStream bos = new BufferedOutputStream(fos);

//BufferedInputStream
BufferedInputStream bis = new BufferedInputStream(fis);

//DataOutputStream
DataOutputStream dos = new DataOutputStream(bos);

//DataInputStream
DataInputStream dis = new DataInputStream(bis);

while(dis.available()!=Configuration.ZERO)
// while(noOfRequests < 100)
{

        //Read a line from the Web Log File
        String line = dis.readLine();


         numberOfItems = numberOfItems + 1;

        if(numberOfItems < Configuration.seventyPercentDataSet4)
         {
                        continue;
         }


        //Pre-process the request
        Request request = preProcessRequest(line);

        //if(maxItemSize <= request.requestedItemSize)
        if(maxItemSize <= request.getRequestedItemSize())
        {
                maxItemSize = request.getRequestedItemSize();
                reqItem = request.getRequestedObject();
                requestNumber = noOfRequests + 1;
        }

        //if(!(request.requestedItemSize > theLRUCache.getMaxCacheSize()))
        if(!(request.getRequestedItemSize() > theLRUCache.getMaxCacheSize()))
        {
                //LRU Cache

                theLRUCache.cacheItem(request);
                //Increment the Total Number of requests
                noOfRequests +=Configuration.ONE;
        }
        else
        {
                //dataViolatingSize = dataViolatingSize +  new
Long(request.requestedItemSize.toString());
                dataViolatingSize = dataViolatingSize +
                                new Long(request.getRequestedItemSize());

                numberOfItemsViolatingSize = numberOfItemsViolatingSize + 1;
        }

        //Once the item is pushed to the Cache set the handle to null
        request = null;

        /*//Increment the Total Number of requests
```

```
                                        noOfRequests +=CacheConstants.ONE;*/

                                        //Every 25000 request clear out the JVM by trying to run GARBAGE Collector
                                        //if(noOfRequests%CacheConstants.TWENTYFIVETHOUSHAND
                                        if(noOfRequests%Configuration.TENTHOUSAND
                                                        ==Configuration.ZERO)
                                        {
                                                try
                                                {
                                                        //Invoke GARBAGE Collector
                                                        //r.gc();
                                                        GarbageCollectorHelper.invokeGC();

                                                        Printer.printString("Analyzing Requests,Total Number of
Items Scanned :" + theLRUCache.getTotalNumberOfItems() + ",Size of All items in Cache :"
                                                                        + theLRUCache.getCurrentCacheSize()
+ " bytes, Number of Items in the Cache :" + theLRUCache.getCacheMapSize()
                                                                        );
                                                        //Put the Thread to sleep
                                                        //Thread.sleep(Configuration.TWOTHOUSAND);

                                                        //Invoke GARBAGE Collector
                                                        //r.gc();
                                                        GarbageCollectorHelper.invokeGC();

                                                }
                                                catch (InterruptedException e)
                                                {
                                                        e.printStackTrace();
                                                }
                                        }

                                }

                                printLRUStats();

                                //System.out.println("Number of Items violating Size :" +
numberOfItemsViolatingSize);
                                //System.out.println("Size of Items violationg Size : " + dataViolatingSize );
                                //Attempt a write to file
                                //dos.writeBytes("Total Hits : " + theLRUCache.getCacheHits());
                        }

                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
                finally
                {
                        try
                        {
                                fis.close();
                                fos.close();
                        }
                        catch (IOException e)
                        {
                                e.printStackTrace();
                        }
                }

                // System.out.println("Max Item Size :" + maxItemSize +
```

```
                        //              ",Request Number:" + this.requestNumber + ",Requested Item:" + this.reqItem);
        }

        private void printLRUStats()
        {
                //Once the Complete Processing of all Web Log Entries is done - Print out all the details
                System.out.println("Total Number of Items : "  + theLRUCache.getTotalNumberOfItems());
                System.out.println("Total Misses : " + theLRUCache.getCacheMiss());
                System.out.println("Total Hits : " + theLRUCache.getCacheHits());
                System.out.println("Total CacheSize : " + theLRUCache.getTotalRequestedItemSize());
                System.out.println("Cache Map Size : "  + theLRUCache.getCacheMapSize());
                System.out.println("Total Cache Hit Size : " + theLRUCache.getTotalCacheHitSize());
                System.out.println("Total Cache Miss Size : " + theLRUCache.getTotalCacheMissSize());
                System.out.println("Current Cache Size:" + theLRUCache.getCurrentCacheSize());

                System.out.println("/******************************************** /");
                System.out.println("Total Number Of Items Prefetched:" + theLRUCache.getNumberOfClusterPrefetched());
                System.out.println("Total Number Of Items Prefetched Hits:" +
theLRUCache.getNumberOfClusterPrefetchHits());
                System.out.println("Total Number Of Items Prefetched Misses:" +
theLRUCache.getNumberOfClusterPrefetchMisses());
                System.out.println("Total Number Of Items Prefetched Removed:" +
theLRUCache.getNumberOfClusterPrefetchedRemoved());
                System.out.println("Total Size of Items Prefetched:" + theLRUCache.getSizeOfItemsPrefetched());
                System.out.println("Total Size of Items Prefetched Hits:" + theLRUCache.getSizeOfItemsPrefetchedHits());
                System.out.println("Total Size of Items Prefetched Misses:" +
theLRUCache.getSizeOfItemsPrefetchedMiss());
                System.out.println("Total Size of Items Prefeteched Removed:" +
theLRUCache.getSizeOfItemsPrefetchedRemoved());
                System.out.println("Total Number of Prefetch Misses:" + theLRUCache.getCacheMissList().size());
                System.out.println("/******************************************** /");
        }

        private Request preProcessRequest(String webLogEntry)
        {
                String[] requestParam = webLogEntry.split(" ");
                //int i=0;
                Request request = new Request();

                String requestTimeString = requestParam[0].toString();

                /*Long requestTime = new Long(requestParam[0].toString());*/
                int index = requestTimeString.indexOf(".");

                String reqString = requestTimeString.substring(Configuration.ZERO, index);

                Long requestTime = new Long(reqString);
                Long requestProcessingTime = new Long(requestParam[1]);
                String clientIPAddress = requestParam[2].trim();
                String tcpCode = requestParam[3].trim();
                Integer requestedItemSize = new Integer(requestParam[4]);
                String requestedObject = requestParam[6].trim();
                String timeOutRedirectionAttribute = requestParam[8].trim();
                String requestedObjectType = requestParam[9].trim();

                //Set the Request Time
                request.setRequestTime(requestTime);

                //Set the Processing Time
                request.setRequestProcessingTime(requestProcessingTime);

                //Set the Client IP Address
```

```
                request.setClientIPAddress(clientIPAddress);

                //Set the TCP Code
                request.setTcpCode(tcpCode);

                //Set the Requested Item Size
                request.setRequestedItemSize(requestedItemSize);

                //Set the TimeOutRedirection Attribute
                request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);

                //Set the Requested Object Type
                request.setRequestedObjectType(requestedObjectType);

                //Set the Domain for the Request
                int index1 = requestedObject.indexOf("/");

                String editedWebObject = requestedObject.substring(index1 + 2);

                request.setRequestedObject(editedWebObject);

                int index2 = editedWebObject.indexOf("/");

                String domainName = null;

                if(index2 != -1)
                        domainName = editedWebObject.substring(0, index2);
                else
                        domainName = editedWebObject.substring(0);

                request.setDomainName(domainName);

                requestParam = null;

                return request;
        }

}
package com.sjsu.edu.fileprocessor;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URLConnection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.sjsu.cache.helper.Request;
import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.node.Node;

public class FileProcessor
{
        private String fileName;
        private String clientIP;
        private String urlDomainName;
```

```
/**
 * 1. Get the Client IP Address
 * 2. Get the URL base name or domain name
 * 3. Check if the the URL occurs in any of the Clusters for the Client IP Address
 * 4. If the URL Occurs in the any of the Clusters, get the Intrasite Cluster for the URL
 * 5. If the Requested Item occurs in the Intrasite Cluster for that URL, load this intrastie
 *            cluster into the Cache. Load other objects for other URLs also into the Cache
 * 6. If the requested URL item does not occur in the cluster, discard the intrasite cluster
 *            and also the intersite cluster.
 *
 * @throws IOException
 *
 */

public Map<Node,Map<String,Long>> processRequest(Request request) throws IOException
{
            //Map<String,Long> intraSite = processInterSite(request);

            Map<Node,Map<String,Long>> consolidatedCluster = processInterSite(request);

/*          if(intraSite!=null)
                        loadAllInterSiteClusters(intraSite,request);*/

            return consolidatedCluster;


}

/**
 *
 * @param request
 * @return
 * @throws IOException
 */
private Map<Node,Map<String,Long>> processInterSite(Request request) throws IOException
{

            Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

            Set<String> currentDomainURLs = null;

            //
            String currentClusterFilePath = null;

            //
            File currentFilePath = null;

            Set<String> currentDomainURL = null;

            //Client IP Address
            String clientIPAddress = request.getClientIPAddress();

            //Domain Name
            String URLDomainName = request.getDomainName();

            //Client IP Intersite Directory
            String clientIPInterSiteDir = Configuration.cIPIntersiteClusterDir + "\\" + clientIPAddress;

            File directoryPath = new File(clientIPInterSiteDir);

            if(directoryPath.exists())
```

```
		{
			String[] fileDirectories = directoryPath.list();

			currentDomainURLs = new HashSet<String>();

			for (String file : fileDirectories)
			{
				currentClusterFilePath = clientIPInterSiteDir + "\\" + file;

				FileInputStream fstream = new FileInputStream(currentClusterFilePath);

				// Get the object of DataInputStream
			   DataInputStream in = new DataInputStream(fstream);

			   BufferedReader br = new BufferedReader(new InputStreamReader(in));

			   String strLine;

			   //Read File Line By Line
			   while ((strLine = br.readLine()) != null)
			   {
			     currentDomainURLs.add(strLine.trim());
			   }
			   //Close the input stream
			   in.close();

			   if(currentDomainURLs.contains(URLDomainName))
			   {
				mapOfInterSiteNodes = processIntraSite(currentDomainURLs,request);
				break;
			   }
			   else
			   {
				mapOfInterSiteNodes = null;
				currentDomainURLs = null;
				currentDomainURLs = new HashSet<String>();
				continue;
			   }
			}
		}

		return mapOfInterSiteNodes;
	}

	/**
	 * Process Intra site Cluster
	 *
	 * @param currentDomainURLs
	 * @param request
	 * @throws IOException
	 */
	private Map<Node,Map<String,Long>> processIntraSite(Set<String> currentDomainURLs,Request request) throws
IOException
	{
		Map<String,Long> intraSiteMap = null;

		Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

		String currentURLDomain = request.getDomainName();

		String clientIPAddress = request.getClientIPAddress();
```

```
                String currentClusterFilePath = Configuration.cIPIntrasiteClusterDir
                                                                        + "\\" + clientIPAddress + "\\" +
currentURLDomain + "\\" + currentURLDomain + ".txt";

                File file = new File(currentClusterFilePath);

                if(file.exists())
                {
                        FileInputStream fstream = new FileInputStream(currentClusterFilePath);

                        // Get the object of DataInputStream
                   DataInputStream in = new DataInputStream(fstream);

                   BufferedReader br = new BufferedReader(new InputStreamReader(in));

                   String strLine;

                   String requestedItem = null;

                   intraSiteMap = new HashMap<String, Long>();

                   //Read File Line By Line
                   while ((strLine = br.readLine()) != null)
                   {
                    String[] elements = strLine.split(",");

                    Long size = new Long(elements[1].trim());

                    intraSiteMap.put(elements[0].trim(), size);

                   }

                   //Close the input stream
                   in.close();

                   requestedItem = request.getRequestedObject();

                   if(!intraSiteMap.containsKey(requestedItem))
                   {
                        intraSiteMap = null;
                   }
                   else
                   {
                        mapOfInterSiteNodes = loadAllInterSiteClusters(intraSiteMap, request,currentDomainURLs);
                   }
                }

      return mapOfInterSiteNodes;

   }

   private Map<Node,Map<String,Long>> loadAllInterSiteClusters(Map<String,Long> intraSiteCluster,
                        Request request,Set<String> currentDomainURLs) throws IOException
   {
                Set<String> currDomainURLs = new HashSet<String>();

                Map<Node,Map<String,Long>> mapOfInterSiteNodes = null;

                Node interSiteNode = null;

                Map<String,Long> webOjects = null;
```

```
String currentClusterFilePath = null;

String[] elements = null;

Long size = null;

//Client IP Address
String clientIPAddress = request.getClientIPAddress().trim();

currDomainURLs.addAll(currentDomainURLs);

currDomainURLs.remove(request.getDomainName().trim());

//Load the first Intersite Node
interSiteNode = new Node();
interSiteNode.setURL(request.getDomainName().trim());

//Intialize map of Intersite Nodes
mapOfInterSiteNodes = new HashMap<Node, Map<String,Long>>();
mapOfInterSiteNodes.put(interSiteNode, intraSiteCluster);

for (String currentURL : currDomainURLs)
{
        currentClusterFilePath = Configuration.cIPIntrasiteClusterDir + "\\" + clientIPAddress
                + "\\" + currentURL + "\\" + currentURL + ".txt";

        File f = new File(currentClusterFilePath);

        if(f.exists())
        {
                FileInputStream fstream = new FileInputStream(currentClusterFilePath);

                // Get the object of DataInputStream
           DataInputStream in = new DataInputStream(fstream);

           BufferedReader br = new BufferedReader(new InputStreamReader(in));

           String strLine;

           webOjects = new HashMap<String, Long>();

           interSiteNode = new Node();

           interSiteNode.setURL(currentURL);

           //Read File Line By Line
           while ((strLine = br.readLine()) != null)
           {
             elements = strLine.trim().split(",");
             size = new Long(elements[1].trim());
             webOjects.put(elements[0].trim(), size);
           }
           //Close the input stream
           in.close();

           mapOfInterSiteNodes.put(interSiteNode, webOjects);
        }
}

return mapOfInterSiteNodes;

}
```

```
}
package com.sjsu.edu.domainprocesso;

import java.io.BufferedWriter;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;

public class LogProcessorTest {

	public static void main(String[] args)
	{
		String baseDir = Configuration.cIPCustomFileBaseDir;

		File baseDirectory = new File(baseDir);

		String listOfDir[] = baseDirectory.list();

		for (String string : listOfDir) {

			File subDirFiles = new File(baseDir + "\\" + string);

			String[] listOfSubDir = subDirFiles.list();

			if(listOfSubDir!=null)
			{
				for (String string2 : listOfSubDir)
				{

					String pathToFile = subDirFiles + "\\" + string2;
					LogProcessorTest.processCluster(pathToFile);
					System.out.println(pathToFile);

				}

			}
		}


	}

	public static void processCluster(String inputWebLogFile)
	{

		String inputFile =
			new String(inputWebLogFile);

		String baseDir = "D:\\CustomizedLogFile";
		Map<String,Long> clusteredObjects = null;
		Set<String> clusteredObjectSet = null;
		IntrasiteLogProcessor logFileReader = null;
		FileWriter fstream = null;
		BufferedWriter out = null;

		int index = inputFile.lastIndexOf("\\") + 1;
		int lastIndexOfDot = inputFile.lastIndexOf(".");
```

```
                String dmnString = inputFile.substring(index, lastIndexOfDot);
                String domainName = "http://" + dmnString ;
                String outputFile = "D:\\ClusteredWebObjects" + "\\" + dmnString + ".txt";

                //logFileReader = new IntrasiteLogProcessor(inputFile, outputFile, baseDir);
                logFileReader = new IntrasiteLogProcessor();

                try {

                        //clusteredObjects = logFileReader.readFile();
                        clusteredObjects = logFileReader.processURLLogFile();

                        if(clusteredObjects!=null && clusteredObjects.size() > 0)
                        {
                                clusteredObjectSet = clusteredObjects.keySet();
                                fstream = new FileWriter(outputFile);
                           out = new BufferedWriter(fstream);

                           for (String string : clusteredObjectSet)
                           {
                                        out.write(domainName + string + "," + clusteredObjects.get(string) +"\n");
                                }

                           //Close the output stream
                           out.close();
                        }
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }

        }

}
package com.sjsu.edu.domainprocessor;

import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;
import com.sjsu.edu.commonhelpers.IntrasiteRequestProcessor;
import com.sjsu.edu.commonhelpers.Request;
import com.sjsu.edu.domainprocessor.helpers.IntraSiteClusterer;
/**
 *
 * The Log processor is the main component that processes the
 * unique individual log files
 *
 * @author Administrator
 *
 */
```

```
public class IntrasiteLogProcessor
{
	private String currentURLDomain;
	private String baseDir;
	private String clientIP;
	private Map<String,Long> uniqueDomainNames;
	private IntraSiteClusterer clusterProcessor;
	private Integer supportThreshold;
	private Double confidenceThreshold;

	public IntrasiteLogProcessor()
	{
		baseDir = Configuration.cIPCustomFileBaseDir;
		uniqueDomainNames = new HashMap<String, Long>();
		clusterProcessor = new IntraSiteClusterer();
		currentURLDomain = null;
		clientIP = null;
	}

	/**
	 * Sets the parameters for processing of the URL based Log file
	 *
	 * @param clientIP
	 * @param currentURLDomain
	 */
	public void setParameters(String clientIP,String currentURLDomain,Double confidenceThreshold,Integer
supportThreshold)
	{
		this.currentURLDomain = currentURLDomain;
		this.clientIP = clientIP;
		this.supportThreshold = supportThreshold;
		this.confidenceThreshold = confidenceThreshold;
	}

	/**
	 * Interface Helper function to begin processing of the Log file
	 * @throws IOException
	 *
	 *  @return: returns a Map of Intrasite Cluster Objects and their sizes
	 */
	public Map<String,Long> processURLLogFile() throws IOException
	{
		return readFile();
	}

	/**
	 * Helper that initiates the Web Log File Reading Process
	 * @throws IOException
	 */
	//public Map<String,Integer> readFile() throws IOException
	private Map<String,Long> readFile() throws IOException
	{
		String fileName = baseDir + "\\" + clientIP + "\\" + currentURLDomain + "\\" + currentURLDomain + ".txt";

		File f = new File(fileName);

		FileInputStream fileInputStream =
			new FileInputStream(f);

		BufferedInputStream bufferedInputStream =
			new BufferedInputStream(fileInputStream);
```

```
		DataInputStream dataInputStream =
				new DataInputStream(bufferedInputStream);

		return readLines(dataInputStream);

}

//private Map<String,Integer> readLines(DataInputStream dataInputStream) throws IOException
private Map<String,Long> readLines(DataInputStream dataInputStream) throws IOException
{
		StringBuffer requestedItemBuffer = null;
		List<StringBuffer> sessionList = new ArrayList<StringBuffer>();
		//Set<String> uniqueObjSet = new HashSet<String>();
		Map<String,Long> uniqueObjSet = new HashMap<String,Long>();
		Request request = null;
		String logLine = null;
		Long currentRequestTime = null;
		Long requestTimePlus = null;
		String requestedObject = null;
		//Set<String> clusteredObjects = null;
		//Map<String,Double> clusteredObjects = null;
		Map<String,Long> clusteredObjects = null;
		Long prevRequestTime = null;
		Long timeDifference = null;
		Long maxTimeInterval = new Long(Configuration.THIRTY *
					Configuration.SIXTY *
					Configuration.THOUSAND);

		int counter = Configuration.ZERO;

		while(dataInputStream.available()!=Configuration.ZERO)
		{

		 logLine = dataInputStream.readLine();

		 request = IntrasiteRequestProcessor.processRequest(logLine);

		 if(prevRequestTime == null)
			prevRequestTime = request.getRequestTime();

		 //Get the request Time for the first Line
		 currentRequestTime = request.getRequestTime();

		 timeDifference = currentRequestTime - prevRequestTime;

		 if(timeDifference > maxTimeInterval
					&& counter != Configuration.ZERO)
		 {
			sessionList.add(requestedItemBuffer);

			requestedItemBuffer = null;

			counter = Configuration.ZERO;
		 }

		 //Set the RequestTimePlus; Only if Counter is set to ZERO
		 if(counter == Configuration.ZERO)
		 {
			requestedItemBuffer = new StringBuffer();

			requestedObject = extractRequestedObject(request.getRequestedURL());
```

```
                        //uniqueObjSet.add(requestedObject);
                        uniqueObjSet.put(requestedObject,request.getRequestedItemSize());

                        requestedItemBuffer.
                                                append(requestedObject);

                        counter += Configuration.ONE;
                }
                else
                {
                        requestedObject = extractRequestedObject(request.getRequestedURL());

                        //uniqueObjSet.add(requestedObject);
                        uniqueObjSet.put(requestedObject,request.getRequestedItemSize());

                        requestedItemBuffer.
                                                append("," + requestedObject);
                }

                prevRequestTime = currentRequestTime;

            }

            //Add items to each session to a SessionList
            sessionList.add(requestedItemBuffer);

            requestedItemBuffer = null;

            //Invoke the Garbage Collector
            invokeGC();

            clusteredObjects = computeCluster(sessionList, uniqueObjSet,
                                supportThreshold,confidenceThreshold);

            //printStats();

            return clusteredObjects;
}

/** If the number of user sessions is at least greater than 2 only then there is point in
 * creating a cluster of popular objects for that domain, otherwise the if number of sessions is
 * just 1 then there will be only one cluster with each object having probability as 1
 * Thus, this would not be an effective clustering pre-fetching scheme */
private Map<String,Long> computeCluster(List<StringBuffer> sessionList,
                        Map<String,Long> uniqueObjSet,Integer supportThreshold, Double confidenceThreshold)
{
            Map<String,Long> clusteredObjects = null;

            clusteredObjects = clusterProcessor.computeCluster(sessionList,uniqueObjSet,supportThreshold
                                ,confidenceThreshold);

            return clusteredObjects;
}

/**
 * Helper to extract Requested Object
 */
private String extractRequestedObject(String URL)
{
            int index = URL.indexOf("//");

            URL = URL.substring(index + Configuration.ONE);
```

```
		index = URL.indexOf("/");

		if(index != -1)
			URL = URL.substring(index);

		return URL;
}

/**
 * Helper to process each line in the file
 *//*
private Request processLine(String logLine)
{
		Request request = null;

		String[] fields = logLine.split(" ");

		request = createRequestObject(fields);

		return request;
}*/


/**
 * Get the request time
 *
 * @param requestTimeStr
 * @return
 */
private Long processRequestTime(String requestTimeStr)
{
		Double d = new Double(requestTimeStr);

		Long l = new Long((long) (d * Configuration.THOUSAND));

		return l;
}

/**
 * Helper to extract the web object from the URL
 *
 * @param rawURL
 * @return
 */
private String extractWebObject(String rawURL)
{
		return null;
}

/**
 * Helper to process the URL
 */
private String processURL(String URL)
{
		//URL : http://www.gmail.com/skd/
		URL = URL.trim();

		int index = URL.indexOf("//");
		int index2 = -1;

		//www.gmail.com/skd/
```

```
		URL = URL.substring(index + 2);

		index2 = URL.indexOf("/");

		if(index2>-1){
		//www.gmail.com
			URL = URL.substring(0,index2);
		}else
		{
			URL = URL.substring(0);
		}

		if(URL.contains(":"))
		{
			int index3 = URL.indexOf(":");
			URL = URL.substring(0,index3);
		}

		if (uniqueDomainNames.containsKey(URL)) {

			Long count = uniqueDomainNames.get(URL);
			count+=1;
			uniqueDomainNames.put(URL, count);

		}else{

			Long count = new Long(1);
			uniqueDomainNames.put(URL, count);
		}

		return URL;
}

/**
 * Create a Request Object out of the Log File Line
 *
 * @param fields
 * @return
 */
private Request createRequestObject(String[] fields)
{
		Request request = new Request();

		//Long requestTime = new Long(fields[0].toString());
		String requestTimeString = fields[0].trim();

		Long requestTime = new Long(requestTimeString);

		Long requestProcessingTime = new Long(fields[1]);
		String clientIPAddress = fields[2].trim();
		String tcpCode = fields[3].trim();
		Long requestedItemSize = new Long(fields[4].toString());
		String requestedObject = fields[6].trim();
		//requestParam[7] --> '-'
		String timeOutRedirectionAttribute = fields[8].trim();
		String requestedObjectType = fields[9].trim();

		request.setRequestTime(requestTime);
		request.setRequestProcessingTime(requestProcessingTime);
		request.setClientIPAddress(clientIPAddress);
		request.setTcpCode(tcpCode);
		request.setRequestedItemSize(requestedItemSize);
```

```
		request.setRequestedURL(requestedObject);
		request.setTimeOutRedirectionAttribute(timeOutRedirectionAttribute);
		request.setRequestedObjectType(requestedObjectType);

		return request;
	}

	/**
	 * Helper to invoke Garbage Collection
	 */
	private void invokeGC()
	{
		try
		{
			GarbageCollectorHelper.invokeGC();
		}
		catch (InterruptedException e)
		{
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		setConfig();
	}


	private void printStats() throws IOException
	{
		Set<String> uniqueDomain = uniqueDomainNames.keySet();

		/*File outputFile = new File(outputFileName);
		FileWriter fileWriter = null;
		fileWriter = new FileWriter(outputFile);*/

		for (String string : uniqueDomain) {

			System.out.println(string);
			/*fileWriter.write(string + ",Count:"
					+ uniqueDomainNames.get(string)+"\n");*/
		}

		//fileWriter.close();
		System.out.println("Unique Domain Count:" + uniqueDomain.size());

	}

	private void setConfig()
	{

		this.confidenceThreshold = Configuration.intraSiteConfidenceThreshold;

		if(clientIP.equals(Configuration.dp))
			supportThreshold = Configuration.intraSiteSupport1Threshold;
		else
			supportThreshold = Configuration.intraSiteSupportThreshold;


	}
}
package com.sjsu.edu.domainprocessor.helpers;

import java.util.List;
```

```
import java.util.Map;
import java.util.Set;


/**
 * Helper to process the Cluster
 *
 * DEPENDS ON :
 * 1) ProbablityComponent
 *
 *
 * @author Administrator
 *
 */
public class IntraSiteClusterer {

	private SessionProcessor probabilityProcessor;

	public IntraSiteClusterer()
	{
		probabilityProcessor = new SessionProcessor();
	}

	/**
	 * Entry point for Cluster Processor
	 */
	public Map<String,Long> computeCluster(List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet,
				Integer supportThreshold,Double confidenceThreshold)
	{
		return probabilityProcessor.

	computProbabilisticCluster(sessionList,uniqueObjSet,supportThreshold,confidenceThreshold);
	}

}
package com.sjsu.edu.domainprocessor.helpers;


import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.commonhelpers.GarbageCollectorHelper;

/**
 *
 * Component to compute Probability driven Clusters
 *
 * @author Administrator
 *
 */
public class SessionProcessor
{

	private Map<String,Long> uniqueObjectProbability;

	public SessionProcessor()
	{
		uniqueObjectProbability = new LinkedHashMap<String, Long>();
```

```
}

/**
 * Entry point for processor for Probability
 * driven Clustering
 */
public Map<String,Long> computProbabilisticCluster
        (List<StringBuffer> sessionList,Map<String,Long> uniqueObjSet,
                        Integer supportThreshold, Double confidenceThreshold)
{
        if(sessionList.size() >= 2)
        {
                return computeCluster(sessionList,uniqueObjSet,supportThreshold,confidenceThreshold);
        }
        else
        {
                return computePopularCluster(sessionList,uniqueObjSet,supportThreshold);
        }
}

/**
 * Helper to Compute the probabilistic Cluster
 */
private Map<String,Long> computeCluster(List<StringBuffer> sessionList,
                Map<String,Long> uniqueObjMap, Integer supportThreshold, Double confidenceThreshold)
{
        Set<String> uniqueObjSet = uniqueObjMap.keySet();

        //Unique Objects Iterator
        Iterator<String> setIterator =
                uniqueObjSet.iterator();

        Integer uniqueObjectCounter = Configuration.ZERO;

        //Session List Iterator
        Iterator<StringBuffer> sessionListIterator
                                                        = sessionList.iterator();

        Integer noOfSessions = sessionList.size();

        Double numOfSessions = new Double(noOfSessions.toString());

        StringBuffer stringBuffer = null;

        String webObject = null;

        String[] sessionObjects = null;

        String sessionString = null;

        Double theObjectCount = null;

        Double probabilityOfObject = null;

        int i =0;

        //Iterate through the Unique Object List
        while(setIterator.hasNext())
        {
                webObject = setIterator.next();

                i = i + 1;
```

```
                        sessionListIterator = sessionList.iterator();

                        uniqueObjectCounter = Configuration.ZERO;

                        //Try to see if the Unique Object Occurs in how many sessions
                        while (sessionListIterator.hasNext())
                        {
                                stringBuffer =
                                        (StringBuffer) sessionListIterator.next();

                                //Convert the String Buffer Session String to a Session String
                                sessionString = stringBuffer.toString();

                                //Array of Session's Web Objects
                                sessionObjects = sessionString.split(",");

                                //Traverse through all of the Session's Objects
                                for (String string : sessionObjects)
                                {
                                        if(string.equals(webObject))
                                        {
                                                uniqueObjectCounter += Configuration.ONE;
                                        }
                                }
                        }

                        //The Web Object has been checked for in all sessions.
                        //Now compute the probability
                        theObjectCount
                                                = new Double(uniqueObjectCounter.toString());

                        probabilityOfObject = theObjectCount/numOfSessions;

                        if(probabilityOfObject > confidenceThreshold && theObjectCount >= new
Double(supportThreshold.toString()))
                        {
                                uniqueObjectProbability.put(webObject, uniqueObjMap.get(webObject));
                        }

                        stringBuffer = null;
                        sessionString = null;
                        sessionObjects = null;
                        probabilityOfObject = null;
                        theObjectCount = null;

                        //Invoke the GC
                        invokeGC();
                }

                return uniqueObjectProbability;
        }

        private Map<String,Long> computePopularCluster(List<StringBuffer> sessionList,
                        Map<String,Long> uniqueObjMap,Integer supportTheshold)
        {
                Set<String> uniqueObjSet = uniqueObjMap.keySet();

                //Unique Objects Iterator
                Iterator<String> setIterator =
                        uniqueObjSet.iterator();
```

```
Integer uniqueObjectCounter = Configuration.ZERO;

//Session List Iterator
Iterator<StringBuffer> sessionListIterator
                                                        = sessionList.iterator();

StringBuffer stringBuffer = null;

String webObject = null;

String[] sessionObjects = null;

String sessionString = null;

Double theObjectCount = null;

int i =0;

//Iterate through the Unique Object List
while(setIterator.hasNext())
{
        webObject = setIterator.next();

        i = i + 1;

        sessionListIterator = sessionList.iterator();

        uniqueObjectCounter = Configuration.ZERO;

        //Try to see if the Unique Object Occurs in how many sessions
        while (sessionListIterator.hasNext())
        {
                stringBuffer =
                        (StringBuffer) sessionListIterator.next();

                //Convert the String Buffer Session String to a Session String
                sessionString = stringBuffer.toString();

                //Array of Session's Web Objects
                sessionObjects = sessionString.split(",");

                //Traverse through all of the Session's Objects
                for (String string : sessionObjects)
                {
                        if(string.equals(webObject))
                        {
                                uniqueObjectCounter += Configuration.ONE;
                        }
                }
        }

        //The Web Object has been checked for in all sessions.
        //Now compute the probability
        theObjectCount
                                = new Double(uniqueObjectCounter.toString());

        if(theObjectCount >= new Double(supportTheshold.toString()))
        {
                uniqueObjectProbability.put(webObject, uniqueObjMap.get(webObject));
        }

        stringBuffer = null;
```

```
                        sessionString = null;
                        sessionObjects = null;
                        theObjectCount = null;

                        //Invoke the GC
                        invokeGC();
                }

                return uniqueObjectProbability;
        }

        private void invokeGC()
        {
                try {
                        GarbageCollectorHelper.invokeGC();
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

}
package com.sjsu.edu.clientipsplit;

import java.io.IOException;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.helpers.logfilereader.clientIPURLbasedSplitter.WebLogClientIPURLSplitter;

public class ClientIPURLSplitterTest {

        public static void main(String[] args)
        {
                String inputFile = Configuration.dataSet3Path;
                String outputFile = Configuration.resultsPath;
                String baseDir = Configuration.cIPCustomFileBaseDir;
                Long seventyPercentRequests = Configuration.seventyPercentDataSet3;

                WebLogClientIPURLSplitter logFileReader = new WebLogClientIPURLSplitter();
                logFileReader.initializeFileName(inputFile,outputFile,baseDir,seventyPercentRequests);

                try
                {
                        logFileReader.readFile();
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
        }

}
package com.sjsu.edu.clientipsplit;

import java.io.IOException;


import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.helpers.logfilereader.clientIPsplitter.WebLogClientIPSplitter;

public class ClientIPSplitterTest {
```

```
	public static void main(String[] args)
	{
		/**
		 * Configurations For the Dataset
		 */
		String inputFile = Configuration.dataSet3Path;
		String outputFile = Configuration.resultsPath;
		String baseDir = Configuration.cIPLogFileBaseDir;
		Long seventyPercentRequests = Configuration.seventyPercentDataSet3;

		WebLogClientIPSplitter logFileReader = new WebLogClientIPSplitter();
		logFileReader.initializeFileName(inputFile,outputFile,baseDir,seventyPercentRequests);

		try
		{
			logFileReader.readFile();
		}
		catch (IOException e)
		{
			e.printStackTrace();
		}
	}

}
package com.sjsu.edu.logpreprocessor;

import java.io.IOException;

import com.sjsu.edu.commonconstants.Configuration;
import com.sjsu.edu.weblogpreprocessor.WebLogPreprocessor;

public class LogFileReaderTest {

	public static void main(String[] args)
	{

		String inputFile =
			/*new String("E:\\CentralServerLogDump\\sv.sanitized-access.20070109\\" +
					//"abcd.txt");
			"sv.sanitized-access.20070109");*/
		"E:\\CentralServerLogDump\\rtp.sanitized-access.20070109\\rtp.sanitized-access.20070109";

		String outputFile =
			Configuration.cleanDataSet3Path;

			/*"E:\\CentralServerLogDump\\sv.sanitized-access.20070109\\" +
				"processed.sv.sanitized-access.20070109.txt";*/

		String baseDir = "D:\\CustomizedLogFile";
		String datasetDetailsFileName = "E:\\CentralServerLogDump\\sv.sanitized-access.20070109\\" +
		"datasetDetailsFile.txt";

		WebLogPreprocessor logFileReader = new WebLogPreprocessor();
		logFileReader.initializeFileName(inputFile,outputFile,baseDir,datasetDetailsFileName);

		try
		{
			logFileReader.readFile();
		}
		catch (IOException e)
		{
			e.printStackTrace();
```

```
                }
            }
        }
```