CHAPTER 5

NOVEL APPROACHES FOR INTEGRATING WEB CACHING AND MART1 CLUSTERING BASED PRE-FETCHING TECHNIQUE

5.1 INTRODUCTION

As discussed in previous chapter (chapter 4), Web caching and Web pre-fetching are two well known techniques used for enhancing the performance of Web-based information retrieval system.

One of the ultimate research objectives of this thesis research work mentioned in chapter 1 is to identify and experiment Web caching replacement algorithms and to integrate Web caching and clustering-based Web pre-fetching using MART1 so as to optimize the Webpage retrieval speed and thus to reduce the latency in accessing a Webpage. Section 5.2 introduces the need and significance of Web caching and clustering-based Web pre-fetching. Section 5.3 elaborates the research progress in Web caching and pre-fetching. Section 5.4 describes the various Web caching policies and its comparative performances features. Section 5.5 portrays the proposed MLFU algorithm and section 5.6 portrays the proposed MPLFU algorithm. Section 5.7 illustrates and explains the results of experiments and the discussions based on the results obtained under various experiments.
The conventional replacement policies like LRU, LFU and FIFO are used in Web caching environments. Conventional policies are most suitable for memory caching since it involves fixed Web page size. But dealing with Web caching involves pages of varying size and hence there is a need for an efficient algorithm that works better in Web cache environment.

It is already noted in the previous chapter (chapter 4) that conventional replacement policies are not suitable for caching with clustering-based pre-fetching.

5.2 SIGNIFICANCE OF WEB CACHING AND CLUSTERING-BASED WEB PRE-FETCHING TECHNIQUE

Web pre-fetching is used to improve the performance of Web-based information retrieval system by pre-loading Web pages into the cache before the actual user request arrives. It involves following steps:

- Prediction system which anticipates user’s future request based on their past access pattern.

- Pre-fetching it from origin server and loading them into the cache.

Thus, Web pre-fetching also involves Web caching. A lot of research works have been focused and addressed on Web caching and Web pre-fetching done separately by many researchers earlier. Only few research works have been carried out on integration of Web caching and Web pre-fetching techniques. The following section gives an overview of all such techniques.
Most of the existing pre-fetching techniques employ single object pre-fetching technique, which can be handled by traditional cache replacement policies. However, in clustering-based pre-fetching technique, multiple objects are pre-fetched; hence traditional policies lead to inefficient Web caching. Thus, this chapter provides different heuristic techniques in cache replacement policies which will ultimately increase byte hit rate and save the bandwidth so as to improve the network performance.

5.3 RESEARCH ON WEB CACHING AND PRE-FETCHING

It is very much important to investigate the impact of these two techniques when combined together. Kroeger et al (1997) have discussed and combined Web caching and Web pre-fetching to improve the delay in accessing the Webpages. The result has shown that combination of Web caching and Web pre-fetching improved the Web access latency up to 60%, whereas Web caching alone improved the Web access latency up to 26%. In a similar work, the authors have proposed pre-fetching technique based on Web usage mining. They have used Web usage mining to obtain Web access patterns and have used these patterns to enhance GDSF cache replacement policy with Web pre-fetching policy (Yang el al 2001). Teng et al (2005) have proposed a cache replacement algorithm called Integrated Web Caching and Pre-fetching (IWCP) for integrating Web caching and Web pre-fetching.

Similar to Ibrahim and Xu (2004), the author, Acharjee (2006) have used Artificial Neural Network (ANN) in both Web pre-fetching and Web cache removal policy. In his approach, keywords presented in URL anchor text are used to predict the user’s future request. The main drawback is that it does not consider the factors such as recency and frequency in Web cache
removal policy. Also, the keywords extracted from Web documents were given as input to ANN and hence it causes extra overhead on the server side.

Jin et al (2007) have developed an algorithm for integrating Web caching and Web pre-fetching technique for wireless local area network. Similarly, Sulaiman et al (2009) have designed a framework for integrating Web caching and Web pre-fetching on mobile environment. Also, they have used only traditional policies to test their works.

All the works discussed above integrates Web pre-fetching with caching. However, these approaches are still found to be inefficient (Liu 2009).

Most of the works discussed above were single object pre-fetching technique which uses association rules for predicting user next page access. They are inaccurate and inefficient since these works predict a particular page depending on the user’s previous access patterns (Khalil et al 2009 and Xiao et al 2001).

Moreover, these approaches employ the traditional replacement policies that are inefficient in clustering based pre-fetching technique. Hence there is a need to identify a suitable replacement policy that would make use of clustering-based pre-fetching technique.

5.4 WEB CACHING POLICIES

Web caching policy decides which pages are to be eliminated from the cache, when there is no enough space for storing new page. The main goals of Web caching are:
It has to use the cache space efficiently;

- It has to optimize one or several Web cache performance metrics such as cache hit rate; byte hit rate, response time, etc.

- It has to improve CPU, memory utilization and network performance.

If the future requests are known, then Web cache system can be optimized further. But the problem is that knowing future requests is obviously impossible. Hence, Web cache policies must use some form of heuristics to decide which documents to replace and which to retain. Some of the heuristics includes areas that can be focused upon are:

- Time of last access
- Time of previous accesses
- Time at which the document was entered into the cache
- Number of past accesses to the document
- Document size
- Cost of retrieval
- Page popularity

5.4.1 Traditional Policies

LRU is the most commonly used caching policy and it is based on page access time that is when a page was referenced. If a page is not accessed for the longer time then that page is removed from the cache and thus emptying a room for the new page. This algorithm leads to a high hit rate
while causing little overhead. It does not consider the factors such as frequency, cost and popularity.

In LFU, each page has a priority value based on number of accesses to a page since it is in the cache. This algorithm makes it hard for new items to become as valuable as the older ones, thus the cache tends to be jammed with old objects that had once been popular. It also does not consider the factors such as recency, size and cost in removal policy.

In FIFO policy, objects are simply replaced in the same order as they arrive; i.e., when it is entered into the cache. Table 5.1 shows the heuristics that are utilized by different caching policies.

Similarly Random (RAND) policy randomly selects an object for replacement (Wessels and Duane 2001). Another simple policy called SIZE which replaces the larger objects from the cache. In Web-based applications, it performs better than LRU in terms of hit rate, but it performs poor in terms of byte hit rate.

LRU-k is the variation of LRU algorithm. LRU-k considered the $k^{th}$ last access to an object in the removal policy. For instance, in LRU-2, the value is based on the second most recent access. LRU-1 is equivalent to LRU (Tomkins 1997).

LRU-Threshold policy increases the cache hit rate but provides low byte hit rate. Log (Size) +LRU combines the SIZE and LRU strategies by evicting the least recently used of all documents whose log size is the highest (Pei and Irani 1997).
Table 5.1 Comparison of Replacement Policies

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Heuristics</th>
<th>Time of Last access</th>
<th>Time of $k-1^{th}$ accesses</th>
<th>Time of entry into the cache</th>
<th>Number of accesses to a Page</th>
<th>Page Size</th>
<th>Cost of Web Page Retrieval (Latency)</th>
<th>Page Popularity from log files</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU-$k$</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU-Threshold</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFU-DA</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heuristics</td>
<td>Algorithms</td>
<td>Time of Last access</td>
<td>Time of Previous accesses</td>
<td>Time of entry into the cache</td>
<td>Number of accesses to a Page</td>
<td>Page Size</td>
<td>Cost of Page Retrieval</td>
<td>Page Popularity</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
<td>---------------------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>-------------------------------</td>
<td>----------</td>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Log(Size)+ LRU</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRV</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDS</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDSF</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Proposed Algorithm (MLFU)</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Proposed Algorithm (MPLFU)</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LFU with Dynamic Aging (LFU-DA) combines LFU with age factor to neglect cache pollution of jamming the cache with one popular item. It is the combination of LFU and LRU. Recently requested documents get a bonus compared to the other ones. This bonus is called the cache age factor (Dilley et al 1999).

Greedy Dual-Size is another algorithm which is based on cost and size. It is mainly used in Web caching environment where Webpages are of different sizes. GDS assigns a priority value to each page which is stored in the cache. It is calculated as cost/size, where cost is the cost of fetching an object and size is the size of the Web object. When the cache is full and a new page has to be stored in the cache then GDS will choose the page which has lowest priority value for removal. When a cached page is requested again then its priority value is restored. Thus, it incorporates the time of last access, cost of retrieving a page and page size in the decision making process (Pei and Irani 1997).

This chapter provides the various heuristic techniques for clustering-based pre-fetching technique which ultimately reduces the number of objects pre-fetched and thereby saves the bandwidth. In this research work, the proposed heuristic techniques uses size, cost, frequency, recency and page popularity in decision making process.

The proposed algorithms MLFU and MPLFU are the variations of traditional Least Frequently Used cache replacement policy. Hence, it is appropriate to present the LFU cache replacement algorithm and the algorithms compared such as LRU and FIFO. MLFU provides higher hit rate than traditional LFU policy.
5.5 MODIFIED LEAST FREQUENTLY USED CACHE REPLACEMENT ALGORITHM (MLFU)

The first proposed heuristic technique uses Web usage mining as a tool to enhance Web caching system. The architecture of the proposed system is shown in Figure 5.1. The main component here is that the access pattern (chapter 3) is generated from the preprocessed Web log file. The access pattern is a matrix that represents access count of each unique user on the frequently visited Webpage. This access pattern gives the popularity of a Webpage from the past browsing behavior.

Whenever user requests for a Webpage, the page popularity is retrieved with reference to the access pattern in Figure 3.1 and it is appended to the current frequency. So its priority is updated as per Equation (5.1) below.

\[
Pr_i = L + \frac{(F_c + F_{pi})}{Size_i}
\]

where

- **L** - Is a recency or aging factor which is initialized to 0 and updated whenever a page replacement occurs (Priority of page getting replaced)
- **Pr\_i** - is the priority of a page\_i
- **F\_c** - is the current page frequency
- **F\_{pi}** - is the page popularity retrieved from Web log file
- **Size\_i** - is the current Web page size in bytes

When the cache is full and a new page arrives, the page with lowest priority is replaced. The algorithm is given in Figure 5.2.
Figure 5.1 Architecture of MLFU Pre-fetching System
//Algorithm: MLFU Cache system
//Input: $P_i$ - Requested Page, $F_{pi}$ - $i^{th}$ Page past access frequency (Page Popularity) and $Size_i$ - Requested page size
//Output: No. of Hits

Initialize

$L = 0$; // Recency Factor and updated to a Priority Value of Recently Replaced Page.

$F_i = 0$; // $F_i$ - $i^{th}$ Page Current frequency since it in Cache

$F_{pi} = 0$; // $F_{pi}$ - $i^{th}$ Page Past Access Frequency (Page Popularity)

For Each Requested Page

If page is already in cache then

- Update it frequency by $F_i = F_i + 1$;
- Update its priority value by $P_{ri} = L + (F_i + F_{pi})/Size_i$;
- Update Hits as $Hits = Hits + 1$;

Else

If cache has enough space to store incoming page then

- Load $P_i$ into the cache,
- Initialize it current frequency as $F_i = 1$ and
- Update its priority value by $P_{ri} = L + (F_i + F_{pi})/Size_i$;

Else

While there is no enough space then

- Let $L = \min(P_{ri})$, for all pages in cache
- Evict $P_j$ such that $P_{ri} = L$

Load $P_i$ into the cache
- Initialize current frequency as $F_i = 1$;
- Update its priority value by $P_{ri} = L + (F_i + F_{pi})/Size_i$;
- Decrement all others page priorities by $L$;

Display the Hits

Figure 5.2 MLFU Algorithm
5.6 MODIFIED PRE-FETCHING-BASED LEAST FREQUENTLY USED CACHE REPLACEMENT ALGORITHM (MPLFU)

The architecture of the second proposed method is presented in Figure 5.3. The second heuristic technique assigns the priority value based on whether it is the actual page request or pre-fetching page request. This is because there is a possibility that pre-fetched pages are replaced before it is accessed by the user. Since, in clustering-based pre-fetching environment, multiple objects are pre-fetched and so this cannot be handled by traditional replacement policies like LFU, LRU and FIFO. Hence in this work, MPLFU cache replacement policy has been proposed which optimizes the byte hit rate and saves the bandwidth.

The pre-fetch system will fetch all the pages in a user cluster by using MART1 and it will assign priority, based on whether it’s an actual page or pre-fetched page. If the cache memory has enough space to hold the current page then it inserts the page into the cache and sets the frequency to 1 if it is an actual request, otherwise finds the maximum frequency value among the pages already in cache and initializes the current frequency using the obtained value incrementing it by 1. The MPLFU algorithm is given in Figure 5.4.
Figure 5.3 Architecture of MPLFU Pre-fetching System
//Algorithm: Modified Cache Replacement Algorithm for Clustering-Based Pre-fetching Technique (MPLFU)
//input: User Actual Request and Pre-fetching Request and Page Size
//Output: No. of Hits and Byte Hit Rate

L=0    //Recency Factor
F_i=0;  // F_i - i^th Page current frequency since it is in cache
Hit=0;  // count if the actual requested page is in cache
Bytes=0 // Add the page size if it is served from the cache if it is cache hit.

For each user request

   If page is already in the cache then
      • Update current frequency F_i as F_i=F_i+1;
      • Update Hit by Hit=Hit+1;
      • Updates Bytes as Bytes=Bytes+Size_i;
      • Update its priority value by Pr_i=L+F_i;
   Else
      If cache memory has enough space to hold the current page then
         If (Actual Page) then
            • Set F_i=1;
            • Set its priority value by Pr_i=L+F_i;
         Else
            Set its priority value as Pr_i=max (Pr_i) +1;
         Else
            While there is no enough space then
               Let L as L=min (Pr_i) for all pages in cache then
               Evict P_j such that Pr_j= L
               If (Actual Page) then
                  Set F_i=1;
                  Set its priority value as Pr_i=L+F_i;
               Else
                  Set its priority value Pr_i=max (Pr_i) +1;
                  Decrement all others page priority by L.
      Display the number of Hits and Bytes.
5.7 RESULTS AND DISCUSSION

The proposed algorithms explained in the work is tested with real time datasets from IRCache, a NLANR (National Laboratory of Applied Network Research) project that encourages research on Web caching and provides data for researchers. These represent the traces for proxy server installation at Research Triangle Park, North Carolina.

Table 3.1 in chapter 3 shows the details of datasets used for testing the proposed method and its preprocessing information. In proxy-based Web caching system, each request from the client is assumed to be forwarded to the proxy server.

The proposed model is implemented as file level caching system that means only complete copy of the objects is cached (when a page is added into the cache then the whole page is added. Similarly when a page is removed from the cache also the entire page is removed).

The proposed model also ignores the issues of cache consistency (make sure that the cached copy is up-to-date). This is because revalidation takes place automatically by the proxy server. It sends IMS query to server and validates it. If it returns status code 200 then it is out-of-date and fresh copy is attached in response header; else if it sends 304 it means that cached copy is good and served from the cache.

Lastly, proposed system is tested by considering only static files like HTML and XML files.
5.7.1 Workload Traces

To test the proposed policies, Web proxy server logs are collected from NLANR and performed clustering using MART1 technique.

5.7.2 Experimental Design

There are two main factors used in the trace driven simulation of proposed work: i) Cache size and ii) Cache replacement policy. To evaluate the performance of the proposed work, metrics such as hit rate and byte hit rate are used.

5.7.3 Cache Size

The first and important factor in this study is cache size. The upper bound represents the total unique bytes in the trace (Bahn et al 1999). An infinite cache is one that is so large and once brought into the cache then it never gets evicted (Busari and Williamson 2001). In order to evaluate the performance, cache size has chosen smaller than testing datasets.

The minimum cache size of proxy server is 5 MB. However, based on the size of the actual request present in the testing dataset, the cache size considered here are 1MB, 2 MB, 3 MB, 4 MB and 5 MB.

5.7.4 Replacement Policies

The simulation result shows the performance of various algorithms. There are LFU, LRU, FIFO, LFU-DA, GDS, GDSF and the proposed works MLFU and MPLFU for hit rate and byte hit rate. For the GDS and GDSF algorithms considers the cost function as one. Cache size considered, starts from 1 MB to 5 MB, because the size of the testing dataset is smaller than 1.5MB. Pseudo code for various replacement policies are given in Appendix1.
5.7.5 Experimental Setup

All the methods presented in this part of the work were also developed using JAVA programming language and Java Development Kit 1.6 with hardware and software specifications mentioned in subsection 4.7.1.3 (chapter 4).

5.7.6 Experimental Results

The experiments were carried out for the proposed work; datasets used (along with its preprocessing details) are presented in Table 3.1 (chapter 3).

Performance of MLFU policy

The performance of various replacement algorithms including the proposed MLFU algorithm are carried out. Table 5.2 through Table 5.5 gives the information about the total hits under various cache policies under varying cache sizes. This is tested on four different datasets. From these tables representing the results, it is observed that the proposed work, MLFU produces higher number of hits compared with the other traditional algorithms such as FIFO, LRU, LFU, GDS, GDSF and LFU-DA.

Figure 5.5 through Figure 5.7 shows the performance of MLFU algorithm and other traditional replacement policies in terms of hit rate under varying cache size. It is inferred from these figures that MLFU performs much better than all other cache policies. Graphs plotted in these figures show the performance of proposed work, MLFU, in terms of hit rate with different cache size in dataset 1. Here the experiment considered 200 users’ requests for testing the proposed algorithm.
Table 5.2 Number of Hits in Dataset1

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>61</td>
<td>94</td>
<td>140</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>LRU</td>
<td>60</td>
<td>81</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>LFU</td>
<td>117</td>
<td>124</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>MLFU</td>
<td>135</td>
<td>138</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>GDS</td>
<td>131</td>
<td>139</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>GDSF</td>
<td>131</td>
<td>139</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
<tr>
<td>LFU-DA</td>
<td>93</td>
<td>119</td>
<td>142</td>
<td>142</td>
<td>142</td>
</tr>
</tbody>
</table>

Table 5.3 Number of Hits in Dataset2

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>111</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>LRU</td>
<td>117</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>LFU</td>
<td>130</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>MLFU</td>
<td>133</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>GDS</td>
<td>133</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>GDSF</td>
<td>133</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>LFU-DA</td>
<td>127</td>
<td>134</td>
<td>134</td>
<td>134</td>
<td>134</td>
</tr>
</tbody>
</table>
Table 5.4 Number of Hits in Dataset3

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>12</td>
<td>69</td>
<td>122</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>LRU</td>
<td>19</td>
<td>49</td>
<td>123</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>LFU</td>
<td>46</td>
<td>83</td>
<td>123</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>MLFU</td>
<td>87</td>
<td>107</td>
<td>120</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>GDS</td>
<td>81</td>
<td>106</td>
<td>119</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>GDSF</td>
<td>81</td>
<td>106</td>
<td>119</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>LFU-DA</td>
<td>64</td>
<td>97</td>
<td>123</td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

Table 5.5 Number of Hits in Dataset4

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>79</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>LRU</td>
<td>87</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>LFU</td>
<td>88</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>MLFU</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>GDS</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>GDSF</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>LFU-DA</td>
<td>88</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>
**Figure 5.5**  Comparison of Hit Rate by FIFO, LRU, LFU and MLFU on Dataset sv[1].sanitized-access.20070109

**Figure 5.6**  Comparison of Hit Rate by MLFU, GDS and GDSF on Dataset sv[1].sanitized-access.20070109
Performance of MPLFU policy

To increase the byte hit rate of LFU policy, an MPLFU policy has been proposed. Tables 5.6 through 5.9 gives the performance information about total hits produced by LRU, LFU and MPLFU policies in terms of total bytes served from the cache in KB, for all the four datasets. For this experiment, 200 users requests are considered for Web caching and pre-fetching.

It is observed from these tables that MPLFU algorithm performs better when the cache size gets increased. It is also observed that MPLFU policy provides better byte hit rate when compared with the LFU policy.
Table 5.6 Total Bytes Served From Cache by Dataset1 (in KB)

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>1224.103</td>
<td>1274.645</td>
<td>239627.27</td>
<td>239627.27</td>
<td>239627.27</td>
</tr>
<tr>
<td>LFU</td>
<td>29621.209</td>
<td>151348.95</td>
<td>239627.27</td>
<td>239627.27</td>
<td>239627.27</td>
</tr>
<tr>
<td>MPLFU</td>
<td>34296.439</td>
<td>166454.95</td>
<td>239627.27</td>
<td>239627.27</td>
<td>239627.27</td>
</tr>
</tbody>
</table>

Table 5.7 Total Bytes Served From Cache in Dataset2 (in KB)

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>18699.95</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
</tr>
<tr>
<td>LFU</td>
<td>20036.35</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
</tr>
<tr>
<td>MPLFU</td>
<td>22822.27</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
<td>31101.47</td>
</tr>
</tbody>
</table>

Table 5.8 Total Bytes Served From Cache in Dataset3 (in KB)

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>220.766</td>
<td>377.613</td>
<td>471807.21</td>
<td>471877.56</td>
<td>471877.56</td>
</tr>
<tr>
<td>LFU</td>
<td>128327.22</td>
<td>350639.29</td>
<td>471825.19</td>
<td>471877.56</td>
<td>471877.56</td>
</tr>
<tr>
<td>MPLFU</td>
<td>80927.836</td>
<td>338841.17</td>
<td>471855.01</td>
<td>471877.56</td>
<td>471877.56</td>
</tr>
</tbody>
</table>
Table 5.9 Total Bytes Served From Cache in Dataset4 (in KB)

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>1 MB</th>
<th>2 MB</th>
<th>3 MB</th>
<th>4 MB</th>
<th>5 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>33869.92</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
</tr>
<tr>
<td>LFU</td>
<td>33348.83</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
</tr>
<tr>
<td>MPLFU</td>
<td>33954.13</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
<td>33954.42</td>
</tr>
</tbody>
</table>

The corresponding graphs for Table 5.6 and Table 5.9 are presented in Figures 5.8 and 5.9 respectively. Total bytes of each dataset is given in Table 5.14.

Figure 5.8  Byte Hit Rate of LRU Vs LFU Vs MPLFU on bo2[1].sanitized-access.20070109
Graph plotted in Figure 5.10 shows the byte hit rate of MPLFU policy compared with LFU policy. However, while considering hit rate of MPLFU policy it provides only a moderate hit rate than LFU policies. Hence MPLFU policy can improve the network performance by yielding higher byte hit rate and better bandwidth utilization.

Graph given in Figure 5.11 gives the performance of MPLFU policy in terms of hit rate under varying cache size. From this graph, it is observed that MPLFU policy provides moderate hit rate but provides higher byte hit rate. Hence network performance gets improved.
Figure 5.10 Comparison of Hit Rate in MLFU and LFU on bo2[1].sanitized-access.20070109

Figure 5.11 Comparison of Hit Rate in MLFU and LFU on uc[1].sanitized-access.20070109
Figure 5.12 Comparison of Hit Rate in MLFU and MPLFU on uc[1].sanitized-access.20070109

Figure 5.13 Comparison of Byte Hit Rate in MLFU and MPLFU on uc[1].sanitized-access.20070109
Figures 5.12 and 5.13 shows the comparison of two proposed works including MLFU and MPLFU policies. These two policies are compared against in terms of hit rate and byte hit rate. It is inferred that MPLFU policy provides higher byte hit rate and low hit rate whereas MLFU policy provides mostly higher hit rate and low byte hit rate. Hence it saves the bandwidth.

5.8 SUMMARY

Experiments were performed as intended and specified in research objectives. Studies were made by making a comparison between the performances of different replacement strategies on various datasets.

The performances of the proposed policies show that they improve network performance by yielding higher hit rate and byte hit rate. Hence, by using the proposed system, information providers can provide information at much faster rate in a restricted environment. Thus, they can satisfy and retain their users. It also utilizes the available bandwidth efficiently.

Based on the experiments carried out on the four different datasets using the traditional policies and the proposed algorithms for cache replacement, it is inferred that the proposed algorithms (MLFU and MPLFU) outperform the existing algorithms in terms of hit rate and byte hit rate. Moreover, MPLFU policy provides higher byte hit rate and low hit rate while MLFU policy provides mostly higher hit rate and low byte hit rate. From these study reports and inferences drawn, conclusions, findings and future study recommendations are made and presented in the next chapter.