CHAPTER 5
CACHE REPLACEMENT SCHEME TO ENHANCE WEB PREFETCHING

5.1 INTRODUCTION

Web caching and prefetching techniques provide effective solution to enhance the response time of end users. The web objects are stored at locations closer to end users for serving their requests with minimal delay. Web caching exploits the temporal locality and prefetching exploits the spatial locality that is inherent in the user access patterns of web objects. Web caches are categorized into: client cache, proxy cache and server cache depending on the location where they are deployed in the web architecture (Zeng et al 2004). Server cache also referred to as reverse or inverse cache handles web documents of a single web server and reduces its workload. Proxy caches that are often located near network gateways allow several users to share the resources and reduce bandwidth required over expensive dedicated internet connections. Client cache also referred to as browser cache are located close to the web users and provide short response time if the requested object is available in cache. It enhances the web access performance and is economical to manage due to its close proximity to end users.
Web prefetching decides *when* and *what* web objects to be fetched from web server during its operation. Two approaches for prefetching the objects from server are: a) online approach - It fetches web objects during short pauses that occur when user reads displayed page on screen b) offline approach – It fetches web objects during off-peak periods or when user remains idle for certain time period. When aggressive prefetching is employed it can create *cache pollution* by replacing useful data with prefetched data in the cache. Similarly, if web objects stored in the cache as part of web caching are not accessed frequently, then it creates *cache pollution* that negatively affects system performance. For effectively utilizing the limited cache capacity and to avoid *cache pollution*, replacement algorithms are designed to manage the contents in cache by effectively selecting the objects to be evicted from cache for storing new objects. Cache replacement schemes need to implement algorithms that don’t use complicated data structures to provide effective performance.

Several research work in recent years have applied intelligent techniques such as back-propagation neural network (Cobb and ElAarag 2008), fuzzy systems (Ali and Shamsuddin 2009) and evolutionary algorithms (Sulaiman et al 2008, Ali et al 2011) to implement cache replacement schemes in web caching and prefetching environment. The techniques reported in these works indicate that the replacement activity based on intelligent approaches are more efficient and adaptive to web caching environment compared to classical replacement approaches (e.g. LRU, LFU).
This chapter discusses an efficient cache replacement scheme for managing the client-side cache that is partitioned into regular and prefetch cache for handling web caching and prefetching. Regular cache stores web objects received from the following sources: a) objects that are demand requested by the users and b) frequently accessed objects in prefetch cache that are transferred to regular cache. Prefetch cache stores web objects downloaded based on the predictions generated as part of web prefetching. The contents of regular cache are managed using the replacement algorithm based on Fuzzy Inference System (FIS). LRU algorithm is used to manage the contents of prefetch cache. The proposed scheme is designed such that it retains the useful web objects for longer time duration and removes the unwanted objects from cache for efficient performance. Integration of prefetching in to the client cache system improves the hit ratio because the prefetched objects are stored in prefetching cache maintained independently from regular cache.

5.2 CACHE REPLACEMENT - OVERVIEW

Cache replacement algorithms are designed to effectively decide the web objects to be evicted from cache for satisfying the following aspects:

- Effective utilization of available cache space
- Improving hit ratio
- Reducing network traffic
- Minimizing the load on origin web server
Replacement algorithm will compute priority of web objects stored in the cache to select web objects to be evicted from cache. Factors considered for computing the priority of web objects are: popularity (frequency), recency, object size, popularity consistency, access latency (delay) and object type (html/text, image/video, application). Web access latency (delay) represents the time interval between sending the user request and receiving the last byte of requested content as response. Recency represents the time when object was last referenced and it reflects the temporal locality that exists in user access patterns. Web objects are selected for eviction from the cache such that it has the lowest access demand in the near future. Replacement policy is applied whenever cache reaches its maximum limit or to evict objects that are not used for long duration.

Combining several factors to influence the replacement process in deciding the web objects to be removed from cache is not an easy task as each factor has its own significance in different situations. Locality of reference characterizes the ability to predict future accesses to web objects based on the past accesses to objects. Two main types of locality are: Temporal and Spatial. Temporal locality indicates that recently accessed objects are likely to be accessed again in the future. Spatial locality indicates that accesses to certain objects can be used as a reference to predict future accesses to other objects.

Each web object is identified using different characteristics and among them URL is the unique characteristic to identify the object. Most replacement strategies use a combination of these characteristics to make their decisions.
Important characteristics of web objects are (Podlipnig et al 2003):

- **Recency** - Time when object was last requested
- **Frequency** - Number of requests to the object
- **Size** - Size of web object in bytes
- **Cost** - Cost involved in fetching object from origin server
- **Request value** - Benefit gained from storing the object in cache
- **Expiration time** - Time to Live (TTL) of the object

Factors such as object size, object type and access latency are static and they are determined only once when the object is initially requested by the user. Factors such as frequency, recency and popularity consistency are dynamic and they are computed frequently till the object resides in cache.

Podlipnig et al (2003) categorized cache replacement algorithms as:

- **Frequency based**
  Frequency of objects based on its popularity was analyzed and used as the deciding factor for future actions on the web objects.

- **Recency based**
  It exploits the temporal locality seen in web requests patterns and recency was used as the main deciding factor in selecting the objects to be removed from cache.
• *Frequency / Recency based*

It combines both recency and frequency factors in making decisions on the web objects stored in cache.

• *Function based*

It uses a general function to compute the value of an object based on which the decisions are taken.

• *Randomized*

The objects are randomly selected for removal from the cache.

Temporal locality and document popularity influence the web request sequences. Object size and cost of fetching an object from server, along with temporal locality and long term popularity plays significant role in performance of cache replacement schemes.

### 5.3 FUZZY INFERENCE SYSTEM

Fuzzy Inference System (FIS) shown in Figure 5.1 is a popular computing framework based on the concept of fuzzy set theory, fuzzy if-then rules and fuzzy reasoning. Fuzzy Inference is the process of formulating the mapping from a given input to an output using fuzzy logic. The mapping provides a basis from which decisions can be made or patterns discerned. FIS has good function approximation capability that is reflected in various problems such as control, modeling and classification. Fuzzification transforms the crisp input into degree of match with linguistic values. Knowledge base comprises of two components: Rule base and Database. The rule base contains various fuzzy if-
then rules and database defines the membership functions of fuzzy sets used in fuzzy rules. Inference engine is responsible for making decision operation on rules. Defuzzification transforms the fuzzy results into crisp output.

![Figure 5.1 Framework of Fuzzy Inference System](image.png)

The time complexity of FIS system depends on the number of rules it considers to make decision. Fewer rules in the database will result in better system performance.

### 5.3.1 Membership Function

It provides a measure of the degree of similarity of an element to a fuzzy set. It can be chosen either arbitrarily by the user based on his experience or designed using machine learning methods. Different shapes of membership function that exists are: triangular, trapezoidal, piecewise-linear, Gaussian and bell-shaped.
- **Gaussian Membership Function**

\[ f(x; \sigma, c) = e^{-\frac{(x-c)^2}{2\sigma^2}} \]

It depends on two parameters \( \sigma \) and \( c \).

- **Trapezoidal Membership Function**

\[
f(x; a, b, c, d) = \begin{cases} 
0, & x \leq a \\
\frac{x-a}{b-a}, & a \leq x \leq b \\
1, & b \leq x \leq c \\
\frac{d-x}{d-c}, & c \leq x \leq d \\
0, & d \leq x 
\end{cases}
\]

It depends on four scalar parameters: \( a, b, c \) and \( d \).

- **Bell Function**

\[
f(x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}}
\]

It depends on three parameters: \( a, b \) and \( c \). The parameter ‘\( b \)’ is usually positive. Parameter ‘\( c \)’ is used to locate the center of curve.
• Triangular Function

\[ f(x; a, b, c) = \begin{cases} 
0, & x \leq a \\
\frac{x-a}{b-a}, & a \leq x \leq b \\
\frac{c-x}{c-b}, & b \leq x \leq c \\
0, & c \leq x 
\end{cases} \]

It is a function of vector ‘x’ and depends on three parameters: a, b, c.

Each membership function is assigned a linguistic term and it will map the input parameters to the membership value in the range 0 to 1. The input space is sometimes referred to as Universe of Discourse (Z).

Figure 5.2 Membership Functions for Recency
Consider the system takes four input parameters: Recency, Frequency, Delay Time and Object Size. Each input parameter is associated with three membership functions: {low, medium and high} that will map the input values to the associated fuzzy sets in the degree of 0 to 1.

Figure 5.2 represents the **Recency** input being mapped to its membership functions {low, medium and high} and it is illustrated using a bell curve. Figure 5.3 represents the **Frequency** input being mapped to its membership functions {low, medium and high}.

**Figure 5.3 Membership Functions for Frequency**

The input parameter *Delay Time* is mapped to its membership functions {low, medium and high} as shown in Figure 5.4.
Figure 5.4 Membership Functions for Delay Time

Figure 5.5 represents the mapping of input ‘Object Size’ to its membership functions {small, medium and large}.

Figure 5.5 Membership Functions for Object Size
5.3.2 Fuzzy Rules

The rules are linguistic IF-THEN statements that constitute a key aspect in the performance of fuzzy inference system. It describes the relationship between input and output values. IF part is called as “antecedent” and THEN part is called as “consequent”.

Example:

\[
\text{IF \{Frequency is low\} THEN \{Removal is high\}}
\]

\[
\begin{array}{c}
\text{Antecedent} \\
\text{Consequent}
\end{array}
\]

{Frequency, Removal} are linguistic variables.

{low, high} are linguistic terms that correspond to membership function.

If antecedent of a rule has more than one part, then fuzzy operator (AND) is applied to obtain a single value that represents the antecedent result for that rule. The consequent is a fuzzy set represented by a membership function and it can be reshaped using a function associated with the antecedent.

Decisions are taken by testing all the rules in a Fuzzy Inference System, so the rules must be combined in order to generate the final output. Aggregation is the process by which the fuzzy sets that represent the output of each rule are combined into a single fuzzy set. It occurs only once for each output variable before performing defuzzification.
5.3.3 Defuzzification

It takes the aggregated output fuzzy set as input and produces a single output value (crisp data).

Commonly used methods for defuzzification as shown in Figure 5.6 are:

- **Centroid of Area (COA)**
  
  It is the most commonly used technique and is considered to be more accurate. It returns the center of area under the curve.

  \[
  z_{COA} = \frac{\int z \mu_A(z) dz}{\int \mu_A(z) dz}
  \]

  \(\mu_A(z)\) is the aggregated output membership function.

- **Bisector of Area (BOA)**

  It will divide the region into two sub-regions of equal area and it sometimes coincides with the centroid line.

  \[
  \int_a^{Z_{BOA}} \mu_A(z) dz = \int_{Z_{BOA}}^\beta \mu_A(z) dz
  \]

  where \(\alpha = \min \{z; z \in \mathbb{Z}\}\) and \(\beta = \max \{z; z \in \mathbb{Z}\}\).
- Mean of Maximum (MOM)

\[ Z_{MOM} = \frac{\int z' dz'}{\int dz'} \]

where \( Z' = \{ z; \mu_A(z) = \mu^* \} \)

if \( \max \mu_A(z) = [z_1, z_2] \) then \( Z_{MOM} = \frac{z_1 + z_2}{2} \)

- Smallest of Maximum (SOM)

Amongst all \( z \) that belong to \([z_1, z_2]\), the smallest is called \( z_{SOM} \)

- Largest of Maximum (LOM)

Amongst all \( z \) that belong to \([z_1, z_2]\), the largest value is called \( z_{LOM} \)

Figure 5.6 Methods to perform Defuzzification
5.4 PROPOSED FRAMEWORK

The framework shown in Figure 5.7 manages the client-side cache by partitioning them into two parts: regular cache and prefetch cache. Each part of the cache has its own storage space and is managed independently using separate replacement policy. Regular cache is managed using Fuzzy Inference System (FIS) algorithm and prefetch cache is managed using LRU algorithm.

**Figure 5.7 Framework for managing regular/prefetch cache**

Regular cache stores web objects that are demand requested by users and frequently accessed objects that are transferred from prefetch cache. Prefetch cache stores web objects that are downloaded from server using the predictions generated as part of web prefetching. When user frequently accesses the objects
stored in prefetch cache, they are moved to regular cache to ensure that the popular objects reside in cache for longer time duration. The scheme effectively removes the useless objects to alleviate cache pollution and maximize the hit ratio.

When users’ requests are satisfied using the contents of either regular or prefetch cache, then it indicates *cache hit* and the requests are not forwarded to the web server. In case of *cache miss*, the requests are forwarded to server for acquiring the required data.

When server receives the user request, it performs the following tasks:

- Records the details of user request in access log file
- Fetch the requested object and generate predictions for the request
- Sends the requested object and its predictions to client.

Server analyzes the user requests stored in access log file to generate the predictions and deliver it to client. Client on receiving the requested object along with list of predictions from server performs the following tasks:

- Received web object stored in regular cache and displayed to user
- Prefetch web objects based on the prediction list during browser idle time and store in prefetch cache.
Figure 5.8 Workflow of caching/prefetching system
The client can also take the responsibility of generating the predictions on its own and use them to prefetch web objects from server. These downloaded objects are then stored in prefetch cache. The web objects received based on demand requests are then stored in regular cache.

The workflow of caching system that also incorporates prefetching mechanism is illustrated in Figure 5.8. When a web object needs to be stored in the cache, the caching system first verifies if the object is cacheable or not. If it is cacheable, then it verifies whether it is a prefetched or demand requested object. In case of prefetched object, it will be stored in prefetch cache by verifying whether it is full or it has space to store the object. LRU algorithm is used to purge objects from the prefetch cache. When objects residing in prefetch cache are accessed frequently within a short time period, then they are moved to regular cache. The regular cache is verified whether it can accommodate the objects coming from prefetch cache or objects demand requested by user. When regular cache is full, then objects are purged based on the outcome of FIS algorithm. The objects stored in regular and prefetch cache is used to satisfy the client requests with minimal latency. In case if the object is not cacheable, then they are delivered directly to the client and displayed on the web browser.

The commonly used factors to determine the popularity of web objects are: frequency, recency and object size. Object popularity is a good estimator for verifying the cacheability of documents, since the objects that are more popular
have high probability of being referenced again by the user in near future resulting in increased cache hit rate.

The requests are considered as cacheable based on the following factors:

- It must have a defined size in bytes that should be greater than zero.
- It must use GET or HEAD method and the status code should be 200 (OK), 206 (Partial Content), 304 (Not Modified).

Dynamic requests are not cached, since they return unique objects every time they are accessed by the user.

5.4.1 Fuzzy System - Input / Output

The input parameters to Fuzzy Inference System are labeled as \{IP_1 to IP_4\} and the target output labeled as \{O_T\}.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_1</td>
<td>Recency of Web object</td>
</tr>
<tr>
<td>IP_2</td>
<td>Frequency of Web object</td>
</tr>
<tr>
<td>IP_3</td>
<td>Retrieval time of Web object</td>
</tr>
<tr>
<td>IP_4</td>
<td>Size of Web object</td>
</tr>
</tbody>
</table>

Frequency and Recency for the objects are estimated based on the sliding window mechanism discussed in (Romano and ElAarag 2011). Sliding window of a request represents the time before and after the request is made.
Table 5.2 Symbols used with their meanings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_i$</td>
<td>requested object</td>
</tr>
<tr>
<td>$\Delta T_i$</td>
<td>time period since object $O_i$ was last requested</td>
</tr>
<tr>
<td>$F_i$</td>
<td>Frequency of object $O_i$ within sliding window</td>
</tr>
<tr>
<td>SWL</td>
<td>Sliding Window length</td>
</tr>
<tr>
<td>$O_T$</td>
<td>Target Output</td>
</tr>
</tbody>
</table>

Recency (IP₁) of object $O_i$ computed as:

\[
\text{recency (} O_i \text{)} = \begin{cases} 
\max (\text{SWL}, \Delta T_i) & \text{if } O_i \text{ was requested before} \\
\text{SWL} & \text{if } O_i \text{ requested for the first time}
\end{cases}
\]

If an object is requested for the first time, then its recency will be fixed as SWL; else it will have the maximum value among SWL and $\Delta T_i$.

Frequency (IP₂) of object $O_i$ computed as:

\[
\text{frequency (} O_i \text{)} = \begin{cases} 
F_i + 1 & \text{if } \Delta T_i \leq \text{SWL} \\
F_i = 1 & \text{if } O_i \text{ accessed beyond SWL}
\end{cases}
\]

Frequency of object ($O_i$) is incremented by 1 with respect to the previous frequency value, if the request for $O_i$ is within backward-looking SWL; i.e. the time interval between the previous request and the new request is within the bounds of backward-looking SWL. Else, the frequency value will be reinitialized to 1.
Target output ($O_T$) will be set to 1, if the object is re-requested again within the forward looking sliding window; else, $O_T$ will be 0. The objective is to use the information of web object requested in the past to predict its revisit in the forward looking sliding window.

### 5.4.2 Managing Regular Cache

When a user requested object or an object transferred from prefetch cache need to be stored in regular cache, it checks whether there is sufficient storage space to accommodate the object. If storage space is available, then object will be stored in the cache. Else, it decides to evict objects based on the outcome generated by Fuzzy Inference System (FIS) algorithm. FIS takes the input parameters of object and decides whether it can reside in the cache or it should be purged. Input parameters are fuzzified by applying the bell membership function and the aggregated output is defuzzified using the centroid of area method.

When the object has high recency and frequency, then it has good chance of residing in the cache. If the outcome from FIS has a value greater than 0.5, then it indicates that the object can reside in cache; else it can be purged. The algorithm used for managing the contents of regular cache is as follows:
O_p = object in prefetch cache
O_R = object in regular cache
O_N = New object

Begin

1. Object to be stored in regular cache.
2. Check if it is O_N or O_p
3. If (O_N) go to step 5
4. If (O_p reference ≥ 2) in short duration, O_p moved to regular cache.
5. If (size of O_N / O_p > available free space in regular cache) {
   For each object O_R in regular cache {
   // apply FIS to find the popularity of object
   If (popularity of O_R ≥ 0.5)
       O_R.cache = 1; // object resides in cache
   else
       O_R.cache = 0; // purge the object
   }
   Do {
   Remove objects with O_R.cache = 0 from regular cache
   } while (size of O_N / O_p > free space in regular cache)
6. Store O_N / O_p in regular cache;
   Remove O_p from prefetch cache;

End
5.5 IMPLEMENTATION

It discusses the training data used for simulation and the process involved in extracting useful information to be given as input to the Fuzzy Inference System.

5.5.1 Training Data

The BU Web trace containing the records of HTTP requests from clients in Boston University Computer Science Department (BU Web Trace, 1995) is used for the simulation. The data collection consists of 9633 files comprising 1,143,839 requests representing a population of about 762 different users. The trace files contain sequence of web object requests that is served either from local cache or from the network.

Each line in the log file represents unique URL requested by the user. It consists of machine name, the timestamp when the request was made, User_ID number, requested URL, size of the document and the object retrieval time in seconds. If a log entry indicates the number of bytes as zero and the retrieval delay as zero, then it indicates that the request was satisfied using the contents of internal cache.

From the collection of requests representing a large number of users, we randomly select the traces of 15 different users to be used in the simulation.
Data Preprocessing

The log files to be used for simulation undergo preprocessing to extract useful information that reflects user navigational behavior. Figure 5.9 represents the sample log file that is used for preprocessing operation. The processed file with valid information is then used for simulation.
Steps involved in preprocessing are:

- Parse the log file to identify distinct fields in each record entry and to track the boundaries between successive records stored in the file.

- Assign unique identifier (URL_ID) to each URL that helps to track the events easily during simulation.

- Extract the useful fields from each line in the log file to be used for analysis.

The output file generated after preprocessing the log file contains the following fields for each request entry:

- Requested URL
- Unique ID assigned to each URL (URL_ID)
- Timestamp of the request
- Delay time
- Size of the requested object

Table 5.3 represents the sample preprocessed data created from the log file that will be used to obtain the training data to be given as input to the Fuzzy Inference System.
The information shown in Table 5.3 is further processed to create the training data as shown in Table 5.4. The recency and frequency values are assigned based on the sliding window mechanism discussed in section 5.4.1. Time period for sliding window length (SWL) in both the forward and backward scenario is taken as 20 minutes to simulate the user browsing patterns. Since the

<table>
<thead>
<tr>
<th>URL</th>
<th>URL_ID</th>
<th>Timestamp</th>
<th>Delay</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://cs-www.bu.edu/">http://cs-www.bu.edu/</a></td>
<td>1</td>
<td>791129602</td>
<td>1135</td>
<td>2009</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/">http://cs-www.bu.edu/</a></td>
<td>1</td>
<td>791129673</td>
<td>367</td>
<td>2009</td>
</tr>
<tr>
<td><a href="http://www.wired.com/">http://www.wired.com/</a></td>
<td>2</td>
<td>791129783</td>
<td>837</td>
<td>941</td>
</tr>
<tr>
<td><a href="http://www.wired.com/">http://www.wired.com/</a></td>
<td>3</td>
<td>791129785</td>
<td>503</td>
<td>277</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/">http://cs-www.bu.edu/</a></td>
<td>1</td>
<td>791497224</td>
<td>774</td>
<td>2087</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/pics/bu-logo.gif">http://cs-www.bu.edu/pics/bu-logo.gif</a></td>
<td>4</td>
<td>791497225</td>
<td>290</td>
<td>1803</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/pics/bu-label.gif">http://cs-www.bu.edu/pics/bu-label.gif</a></td>
<td>5</td>
<td>791497225</td>
<td>357</td>
<td>715</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/faculty/home.html">http://cs-www.bu.edu/faculty/home.html</a></td>
<td>6</td>
<td>791497229</td>
<td>408</td>
<td>1700</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/faculty/best/BestWeb.html">http://cs-www.bu.edu/faculty/best/BestWeb.html</a></td>
<td>7</td>
<td>791497735</td>
<td>969</td>
<td>7966</td>
</tr>
<tr>
<td><a href="http://cs-www.bu.edu/pics/bu-logo.gif">http://cs-www.bu.edu/pics/bu-logo.gif</a></td>
<td>4</td>
<td>791497790</td>
<td>208</td>
<td>1803</td>
</tr>
<tr>
<td><a href="http://www.wired.com/">http://www.wired.com/</a></td>
<td>2</td>
<td>791429783</td>
<td>737</td>
<td>941</td>
</tr>
</tbody>
</table>
user tends to change browsing patterns often and they may have short browsing sessions, we fix SWL to be 20 minutes (i.e. 1200sec) for the simulation.

Table 5.4 Training data created from preprocessed file

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recency</strong></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>3675.5</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Retrieval Time (ms)</strong></td>
<td></td>
</tr>
<tr>
<td>1135</td>
<td></td>
</tr>
<tr>
<td>367</td>
<td></td>
</tr>
<tr>
<td>837</td>
<td></td>
</tr>
<tr>
<td>503</td>
<td></td>
</tr>
<tr>
<td>774</td>
<td></td>
</tr>
<tr>
<td>290</td>
<td></td>
</tr>
<tr>
<td>357</td>
<td></td>
</tr>
<tr>
<td>408</td>
<td></td>
</tr>
<tr>
<td>969</td>
<td></td>
</tr>
<tr>
<td>208</td>
<td></td>
</tr>
<tr>
<td><strong>Size (bytes)</strong></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
<tr>
<td>941</td>
<td></td>
</tr>
<tr>
<td>277</td>
<td></td>
</tr>
<tr>
<td>2087</td>
<td></td>
</tr>
<tr>
<td>1803</td>
<td></td>
</tr>
<tr>
<td>715</td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td></td>
</tr>
<tr>
<td>7966</td>
<td></td>
</tr>
<tr>
<td>1803</td>
<td></td>
</tr>
<tr>
<td><strong>Size (bytes)</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

When an object is requested for the first time or its re-request is within the SWL length, then its recency will be set as 1200. If the time difference between the new request and the previous request to an object is greater than the SWL, then its recency will be the value representing the time difference.
The frequency for the object will be set to 1, if it is requested for the first time. If object is re-requested within the SWL length, then its frequency is incremented by 1; else, the frequency of object will be re-initialized to 1 irrespective of its previous value. The target output will be set to 1, if an object has future reference within the forward SWL length; else its value is set to 0.

5.6 PERFORMANCE EVALUATION

Trace driven simulations are used to evaluate the performance of cache replacement policies. The storage space allocated for cache in the client machine will be distributed equally between the regular and prefetch cache; i.e. 50% of the total capacity allocated to regular cache and remaining 50% to the prefetch cache. To simulate the prefetching of objects, client based prediction and prefetching discussed in chapter 3 is used. When user requested web page is displayed on the browser, predictions are made for that page and objects are prefetched and stored in prefetch cache. If the objects in prefetch cache are requested frequently, then they are moved to the regular cache.

5.6.1 Performance Metrics

The effectiveness of replacement algorithm in improving the performance of web caching and prefetching is evaluated using the metrics: Hit Rate (HR) and Byte Hit Rate (BHR). HR represents the percentage of user requests served using the objects available in cache. It characterizes improvement in availability and minimization of user latency. BHR represents
the percentage of bytes served from cache against the total number of bytes requested by users. It characterizes the reduction in network traffic and easing of link congestion. Increase in HR significantly contributes to the improvement in latency savings (Zhu and Hu 2007, Shi et al 2006).

Important point to note is that the Hit Rate and Byte Hit Rate cannot be optimized for at the same time (podlipnig 2003). Strategies that optimize Hit Rate give preference to smaller sized objects, which tend to decrease the Byte Hit Rate by giving less preference to larger objects.

5.6.2 Experimental Results

The performance of proposed scheme (FIS-LRU) compared with most common replacement policies: LRU and LFU in terms of HR and BHR. In LRU, least recently used objects are removed first and it is a simple and efficient scheme for uniform sized objects. In LFU, least frequently used objects are removed first and its advantage is its simplicity. It is also compared with NNPCR-2 (Romano and ElAarag 2011) an intelligent web caching approach that uses Back-Propagation Neural Network (BPNN) in making replacement decisions.

The algorithms are simulated by varying the cache size from 10MB to 100MB. Log files of 15 different users are split into three groups: user (1 to 5) in Group A, user (6 to 10) in Group B, user (11 to 15) in Group C. HR and BHR
for each group are evaluated separately to analyze the behavior of algorithms that uses the traces of different set of users in each group.

Figure 5.10 represents the hit rate of different polices using the traces of Group-A (user 1 to 5). Figure 5.11 represents the hit rate using traces of Group-B (user 6 to 10). Figure 5.12 represents hit rate using traces of Group-C (user 11 to 15).

![Graph](image)

**Figure 5.10 Hit Ratio using Traces of Group-A (user 1 to 5)**

When cache size increases, it improves the HR for all the replacement policies. It is due to the fact that the cache can store large number of web objects to satisfy the user requests. As observed in the graphs of different traces, the HR of proposed scheme (FIS-LRU) is better than the other approaches. LFU produces the least HR due to cache pollution. The performance of FIS-LRU is
better than NNPCR-2 in most cases and in few the results match with that of NNPCR-2.

Figure 5.11 Hit Ratio using Traces of Group-B (user 6 to 10)

Figure 5.12 Hit Ratio using Traces of Group-C (user 11 to 15)
Figure 5.13 Byte Hit Ratio using Traces of Group-A (user 1 to 5)

Figure 5.14 Byte Hit Ratio using Traces of Group-B (user 6 to 10)
Byte Hit Rate of different policies using the traces of Group-A is shown in Figure 5.13, using the traces of Group-B is shown in Figure 5.14 and using the traces of Group-C is shown in Figure 5.15. As observed in these graphs, BHR of the proposed scheme (FIS-LRU) is better in all the cases when compared to other replacement policies.

5.7 CONCLUSION

This chapter discusses a cache replacement scheme that efficiently manages the client-side cache, which is partitioned into regular and prefetch cache for handling the web caching and prefetching. The proposed scheme uses Fuzzy Inference System (FIS) based algorithm for managing the contents of regular cache and LRU algorithm for managing the contents of prefetch cache. When objects stored in prefetch cache are frequently accessed by users, then they
are moved to regular cache where they are managed efficiently based on the outcome of FIS algorithm. The scheme helps to retain useful objects for longer time period while effectively removing the unwanted objects from the cache.

The performance of proposed scheme (FIS-LRU) in terms of HR and BHR is compared with various algorithms (LRU, LFU and NNPCR-2), where LRU and LFU are basic algorithms and NNPCR-2 is an intelligent algorithm based on back-propagation neural network. HR and BHR for the proposed scheme are computed by considering both the regular and prefetch cache. Results clearly indicate that the proposed scheme (FIS-LRU) outperforms other algorithms in terms of HR and BHR.