CHAPTER 5
DYNAMIC WEB PRE-FETCHING TECHNIQUE

5.1. Introduction

In this chapter, a new integrated web caching and pre-fetching technique aimed at latency reduction, called dynamic pre-fetching technique is reported. Intelligent agents monitor the bandwidth usage and help the prediction engine to decide the number of sites to be pre-fetched. If the bandwidth usage is very less, then all the sites in the preference list and all the subsequent links in the currently accessed web pages are pre-fetched immediately by the pre-fetch engine. Otherwise the number of pages to be pre-fetched depends upon the bandwidth usage, maximum bandwidth capacity of the existing network and weights of URLs obtained from access pattern.

New dynamic pre-fetching model, components/processes involved, their features and benefits are outlined in section 2. Dynamic prediction of web pages are discussed in section 3. The sequence of steps involved in the dynamic pre-fetching technique and dynamic pre-fetching algorithm are given in section 4. In section 5, dynamic pre-fetching technique is simulated by building a browser and it is compared with some of the existing browsers in terms of bandwidth usage. Percentage of pre-fetch hits, cache hit ratio and latency reduction are analyzed in section 5. The work reported in this chapter appears in two publications, [228] and [229].
5.2 Dynamic Pre-fetching Model

A new web pre-fetching technique called dynamic pre-fetching technique is reported in this section. In dynamic pre-fetching technique, the number of subsequent links to be pre-fetched depends on the user's interests in accessing the documents, cache contents, current bandwidth usage and maximum capacity of the existing network. Web caching and pre-fetching techniques are integrated together in dynamic pre-fetching technique to save memory space, to increase the speed and reduce latency. In dynamic pre-fetching technique, the subsequent links of the web pages are pre-fetched only if these web pages are not available in cache and the bandwidth usage is less than a predetermined threshold value. In the subsequent section, the usage of dynamic pre-fetching technique for LAN users is discussed. The same method is suitable for users of stand alone machines by considering that node as the proxy server and storing the user's preference list and hash table in that node itself.

5.2.1 Components/processes

Top view of dynamic pre-fetching model is shown in Figure 5.1. Different components/processes in the dynamic pre-fetching model are user's preference list, intelligent agents, hash table, cache, pre-fetch area, prediction engine and pre-fetch engine. The functions of each components/processes are given below.

(a) User's Preference List

Each user can keep a list of sites to be accessed immediately called user's preference list. The preference list is stored in proxy server's database. If the user wants anonymity and privacy, there is no need for storing the preference list.
Username, password, bookmarks and preference list of users are stored in a database called user profile database.

![Diagram of Dynamic Pre-fetching Model](image)

**Figure 5.1 Top view of Dynamic Pre-fetching Model**

(b) **Intelligent Agents**

The intelligent agents monitor the bandwidth usage and weights of hash table for deciding the number of web pages to be pre-fetched. Intelligent agents are used for parsing the web pages, monitoring the bandwidth usage and maintaining hash table, preference lists and cache consistency. It controls the web traffic by reducing the number of pages to be pre-fetched at heavy traffic and increasing it at light traffic. Thus it reduces the idle time of the existing network and makes the traffic almost constant.

(c) **Hash Table**

A hash table is maintained for storing the list of accessed URLs and its weight information. Since the best case accessing time of data from the hash table is
$O(1)$, hash table data structure is selected in dynamic pre-fetching technique for storing history/log information. Weight of a URL in hash table specifies the number of times it is accessed by users of that network and the time of last accessed. The weight calculation is discussed in the next section. The maximum size of the hash table is either fixed based on the availability of the memory in the proxy server or it can set by the administrator. When the hash table becomes full, the URL entries with least weights are removed for making rooms for entering newly fetched URLs.

(d) Prediction Engine

Depending upon the bandwidth usage and weights in the hash table, the prediction engine decides the URLs to be pre-fetched and gives this list to the pre-fetch engine. Determination of number of links to be pre-fetched is discussed in the next section.

(e) Pre-fetch Engine

The pre-fetch engine maintains the pre-fetching process for pre-fetching the predicted web pages. After pre-fetching, the proxy server keeps the pre-fetched web pages in a separate area of its memory called pre-fetch area.

(f) Pre-fetch Area

The pre-fetch area contains the pre-fetched web pages that are not fetched by the user. Whenever the user actually fetches a web page, it is removed from pre-fetch area and stored it into the cache.
5.2.2 Features and benefits

The main features incorporated into the dynamic pre-fetching model and their benefits are listed below.

- An efficiently managed caching mechanism
- A user profile database at the proxy server to store user details and preferences list
- A hash table for storing the list of accessed URLs and its weight
- Intelligent agents for parsing web pages, monitoring the bandwidth usage and maintaining hash table, preference lists and cache consistency
- Dynamic page pre-fetching according to user browsing history and network bandwidth usage
- Efficient utilization of network idle time and making the bandwidth usage almost constant
- Dynamic pre-fetching increases cache hit ratio, reduces latency and avoids congestion by controlling the web traffic.

5.3 Dynamic Prediction of Web Pages

Whenever a user is logged in, the user name and password are sent to the proxy server. After authentication, the proxy server retrieves the preferences list of that user from user profile database. Proxy server checks in its local cache to see whether these web sites are cached there. Then it prepares a list of sites which are not cached there and sends that list to the prediction engine to decide which sites to be pre-fetched. If the user is not stored the preference list, this pre-
fetching is not needed. The detailed view of intelligent agent based dynamic pre-fetching system is shown in Figure 5.2.

Figure 5.2 Dynamic Pre-fetching System

Whenever a user gives request to a web page, that page is checked in browser cache of the client. If it is not there, the client request is forwarded through the proxy server. The proxy server first checks in local cache, if there is local miss, it checks in pre-fetch area to see whether that web page is pre-fetched earlier. If there is pre-fetch hit, that web page is removed from pre-fetch area and placed in local cache and sends it to the requested client. The client displays it in the browser and stores it in browser cache. If there is pre-fetch miss, that request is forwarded to the web server. Upon receipt of that web page, it is stored in proxy's local cache and sent it to the client. When the user is reading a web page, the intelligent agents in the client parse that web page and find out the subsequent
The client sends that list of subsequent links to the proxy's prediction engine.

The prediction engine searches in hash table for the weight information of these URLs. The prediction engine also gets the bandwidth usage information from the intelligent agents and decides the links to be pre-fetched. Prediction engine gives the list of URLs to be pre-fetched to the pre-fetch engine for pre-fetching it. Only cacheable documents are stored in the cache. Dynamic pre-fetching technique is useful only if the user always surfing the web in the same local area network using the same proxy server or the user surf the web using a single computer always.

5.3.1 Determination of Weights in hash table

While calculating weights, the preference is to be given for frequently accessed URLs and recently accessed URLs. Whenever a proxy server is powered on, the hash table is empty. All accessed URLs such as the URLs actually fetched by the user and pre-fetched URLs by the pre-fetch engine are entered into the hash table. Initially when URLs are entered into the hash table, the user fetched URLs are set to one and pre-fetched URLs are set to zero. The weight field of a particular URL in the hash table is increased by one whenever a user actually fetches that web page. URL's of objects that have been previously accessed frequently may have more weights even if they are not accessed any longer. For reducing the weights of such URLs, a timer is maintained by intelligent agents. When the timer expires, the weights of all URLs in the hash table are reduced by 0.5.
The weights of newly pre-fetched URLs are set to zero and weights of already entered URLs in hash table are not changed when pre-fetching. If any user actually fetches that web page, its weight is incremented by one. The current web page accessed by the user is parsed to find out the subsequent links in that web page. URLs of these links are searched in the hash table to find out its weights. If any URL is not already entered into the hash table, its weights are considered as zero. The number of subsequent links to be pre-fetched depends on weights information and bandwidth usage reported by the intelligent agents. URLs of embedded objects are not stored in the hash table. Because the access count of the embedded objects will be almost same as the web page it belongs to. In order to reduce space, only the URLs of web pages and its weights are stored in the hash table as history.

5.3.2 Determination of web pages to be pre-fetched

The number of web pages to be pre-fetched is decided by the intelligent agents and prediction engine based on the bandwidth usage. The number of links to be pre-fetched for each user depends on the current bandwidth usage and the weights of URLs stored in the hash table. Equations 5.1 and 5.2 are used for finding the number of URLs to be pre-fetched for each user.

\[
\text{MaxByte} = \frac{(\text{Max} - \text{Busage})}{\text{Num}}
\]

(5.1)

where MaxByte is the maximum bytes that can be pre-fetched for each user, Max is the maximum bandwidth capacity of the existing network, Busage is the current bandwidth usage and Num is the number of users using Internet through that network.
For finding the number of URLs to be pre-fetched for each user, the knowledge of the maximum size of the web page that concerned user will access in the current session is needed. It is difficult to find the actual size of the web pages to be pre-fetched without communicating with the origin web server, but the knowledge of the user's access pattern can be used to find the approximate size of web pages.

Number of URLs to be pre-fetched = \( \text{MaxByte} / \text{PageSize} \)  \( \quad (5.2) \)

where PageSize is the maximum size of web page accessed by that user over a period of time. A default value can be set by administrator to PageSize.

5.4 Dynamic Pre-fetching Algorithm

The sequence of steps involved in the dynamic prediction of web pages using dynamic pre-fetching technique is given below and the corresponding state transition diagram is shown in Figure 5.3.

1. Whenever a user is opening a browser, check whether that user is willing to accept authenticated access. If authentication is not required skip steps from 2 to 8 else client accepts username and password, sends it to the proxy server for authentication.

2. When proxy server gets username and password, searches it in user profile database for authentication.

3. After authentication, proxy server retrieves the user's preference list of URLs from the user profile database if preference list of that user is available otherwise skip steps from 4 to 8.
4. Proxy server checks these URLs in local cache and pre-fetch area and prepares a list of missed URLs that are not found in local cache and pre-fetch area.

5. Proxy server sends missed URLs to prediction engine.

6. Prediction engine find out the list of URLs to be pre-fetched after getting current bandwidth usage and weight information from hash table.

7. Prediction engine sends the list of URLs to be pre-fetched to the pre-fetch engine.

8. Pre-fetch engine pre-fetch the web pages corresponding to these URLs and store it in pre-fetch area of the proxy server.

9. Whenever a user is fetching a web page, intelligent agents at client side parse that web page and find out subsequent links.

10. Client checks these subsequent links in browser's cache and prepares a list of hyperlinks which are not found in browser cache and sends these list to proxy server for pre-fetching.

11. When proxy server gets a list of hyperlinks to be pre-fetched from clients, proxy prepares a list of these hyperlinks which are not found in local cache and pre-fetch area and steps from 6 to 8 are performed.

12. If multilevel pre-fetching is set up in the proxy server, prediction engine find out the recently stored web pages in the pre-fetch area and parsed it to find subsequent links then steps 6 to 8 are repeated for pre-fetching.

13. When the pre-fetch area becomes full, weights of pre-fetched web pages are find out from hash table and web pages with least weights are removed first. If more than one web page has equal weights, then First in First out (FIFO) algorithm is used.
14. When a user actually gives request to a web page which was already pre-fetched and available in the pre-fetch area, that URL's are removed from the pre-fetch area and placed it into the cache area.

15. When cache area becomes full, Least Recently Used algorithm is used for making space in the cache area.

---

**Figure 5.3 State Transition Diagram for Dynamic Pre-fetching Technique**
The pseudo code of dynamic pre-fetching algorithm is given in Table 5.1. Variables used in the dynamic pre-fetching algorithm are: MaxByte is the maximum bytes that can be pre-fetched for each user, Max is the maximum bandwidth capacity of the existing network, Busage is the current bandwidth usage, Num is the number of users using Internet through the network, X represents a user, Num_URL_Prefetch is the number of URLs to be pre-fetched, PageSize is the size of the web page, Anchor_Set is the list of subsequent links in the current web pages of user x fetched, Weight(URL) contains weight of the list of URLs in Anchor_Set, Hashtable(URL) contains list of URLs in hashtable and their weights, Count is the number of URLs pre-fetched by a user, Prefetch_URL is the next URL to be pre-fetched. SeekAnchor is used for finding the subsequent hyperlinks in a web page. Search is used for finding the weight of a URL from Hashtable. Largest_Weight_URL is used for finding the URL having largest weight in the Anchor_Set. If multilevel pre-fetching is permitted in the settings of browser, then subsequent links in the pre-fetched web pages are also to be pre-fetched. Recursive procedure is used for pre-fetching the subsequent links of pre-fetched URLs.

<table>
<thead>
<tr>
<th>Table 5.1 Dynamic Pre-fetching Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic_Prefetch (URL)</strong></td>
</tr>
<tr>
<td>Begin</td>
</tr>
<tr>
<td>MaxByte = (Max - Busage) / Num</td>
</tr>
<tr>
<td>For each URL</td>
</tr>
<tr>
<td>Begin</td>
</tr>
<tr>
<td>Count = 1</td>
</tr>
<tr>
<td>Num_URL_Prefetch = MaxByte / PageSize</td>
</tr>
<tr>
<td>Anchor_Set = SeekAnchor(URL)</td>
</tr>
</tbody>
</table>
Foreach URL in Anchor_Set do
Weight(URL) = Search Hashtable(URL)
While (Count <= Num_URL_Prefetch)
    Begin
        Prefetch_URL = Largest_Weight_URL from Weight(URL)
        Remove Prefetch_URL from Weight(URL)
        Pre-fetch(Prefetch_URL)
        Increment Count
        If multilevel pre-fetching is permitted
            CALL Dynamic_Prefetch(Prefetch_URL)
        End
    End
End

5.5 Simulation and Discussion of Results

Dynamic pre-fetching technique is simulated by building a new browser called
Dynamic Pre-fetching Browser (DPB) with dynamic pre-fetching, web caching,
traffic monitoring and user profile capabilities. This new technique is analyzed by
simulating the environment in a medium sized network containing 90 nodes and
online trace driven data are collected. Java language is used for developing the
browser and C# language is used for developing the proxy server in the
simulation environment. Built in commands of C# is used for monitoring the
bandwidth usage. The pseudo codes used for the simulation setup of dynamic
pre-fetching technique is given below.
If open(browser) verified = Authenticate(username, password)
If (verified)
Begin

140
PreferenceList = search(UserProfile)
Found = search(PreferenceList, local_cache)
If !(found) found = search(PreferenceList, prefetch_area)
If !(found) Call Dynamic_Prefetch(PreferenceList)
End
While (request != “Exit”)
Begin
    If (request = “URL”) search (URL, local_cache)
    If !(found) found = search (URL, prefetch_area)
    If !(found)
        Document = Fetch (URL, webserver)
        Send (Document, client)
        Bandwidth_usage = Monitor (web traffic)
        If (Bandwidth_usage < ThresholdLevel) Call Dynamic_Prefetch(URL)
    End
    If (request = “FreeCacheSpace”) UseLRU(SpaceNeed, clean_local_cache)
    If (request = “Prefetch_URL”) Call Dynamic_Prefetch(PrefetchURL)
End

The Parameters used for the simulation of Dynamic Pre-fetching Technique is given below.

Threshold Level for pre-fetching : 75% of maximum bandwidth capacity
Threshold Level < 75% , Allow Pre-fetching
Else Don’t Allow Pre-fetching
Proxy Cache Size : 10 GB
Maximum allowed space for Hash table : 32 MB
Maximum Bandwidth capacity : 1 Mbps Internet connection
Number of users required for testing, 50 to 90 users
Default PageSize : 1 MB
Multilevel pre-fetching : Yes, Level :3
Bandwidth usage of DPB is compared with Internet Explorer (IE) version 6.0 and Netscape Navigator version 7.0. The bandwidth usage in DPB is almost constant using dynamic pre-fetching technique as shown in Figure 5.4. Without considering the current network traffic, other browsers attempts to fetch more objects than the network can afford. But new dynamic pre-fetching technique makes the browser to monitor the current network traffic and user's preferences and helps the browser to utilize the maximum bandwidth available effectively.

![Figure 5.4 Percentage of Bandwidth Usage by Dynamic Pre-fetching Browser, Internet Explorer and Netscape Navigator](image)

Intelligent agents help the browser to monitor the bandwidth usage and adjust the pre-fetching of subsequent links to make constant traffic and to avoid the network bottleneck such as congestion. Percentage of pre-fetch hits by different users in a LAN is shown in Figure 5.5. Percentage of pre-fetch hits by a single user at different instances of time is shown in Figure 5.6.
When proxy is up and a user is logged in, proxy cache will be empty. So initial user requests are forwarded to the web server and fetched it from the web server. Upon receipt of that document, copy of these documents are stored in proxy's cache and send it to the user. Subsequent request for the same URL yields cache hits.

Through a web page, the subsequent hyperlinks in the current web page are fetched by dynamic pre-fetching algorithm. Prefetch-hit rate measures the performance of the pre-fetching algorithm. Prefetch-hit rate is very low especially for a new user because the cache is empty. For a new user, since the user is browsing for the first time, it is expected that most of the URLs in the web page would be already pre-fetched or cached in the proxy server. So pre-fetch hits of a new user are much lower than pre-fetch hits of a regular user. As seen in figures 5.5 and 5.6, the pre-fetch hits are increased to 100% for regular users and the cache hit ratio is increased to 40–75% and latency is reduced in 20–25% as seen in figures 5.5 and 5.6.

Figure 5.5 Percentage of Pre-fetch hits by different users

Figure 5.6 Percentage of Pre-fetch hits by a user at different instance of time
hit. After fetching a web page, the subsequent hyperlinks in the current web page are fetched based on dynamic pre-fetching algorithm. Prefetch-hit ratio measures the percentage of requested objects that were previously fetched. Comparison of the percentage of pre-fetch hits by a new user and old user are shown in Figure 5.7. For a new user initially pre-fetch hits will be zero. But if the user first access URL's in the preference list, then 100% of pre-fetch hits can be achieved. For an old user, since the user is surfing web for a long time, the required web pages would be already pre-fetched or cached in the proxy server. So pre-fetch hits of an old user may be almost constant varying from 50 to 85%. Using dynamic pre-fetching technique, cache hit ratio is increased to 40 – 75% and latency is reduced to 20 – 63% as seen in Figures 5.8 and 5.9.

Figure 5.7 Percentage of Pre-fetch Hits by a New User and a Old User
Figure 5.8 Analysis of Cache Hit Ratio Using Dynamic and without Pre-fetching Techniques

Figure 5.9 Analysis of Latency Reduction
5.6 Concluding Remarks

Web pre-fetching techniques are used for reducing latency, but most of the pre-fetching techniques increase the web traffic [175, 185, 213]. New dynamic pre-fetching technique reported in this chapter pre-fetch the subsequent links only if the bandwidth usage of existing network is less than a predefined threshold. Intelligent agents monitor the bandwidth usage and user's preferences and help the browser to identify the number of URLs to be pre-fetched. Since dynamic pre-fetching technique integrates the web caching technique, already cached documents are not pre-fetched again. The simulation results show that dynamic pre-fetching browser maintains almost constant web traffic even if pre-fetching is done and guarantees maximum utilization of existing capacity of the network. Using dynamic pre-fetching technique cache hit ratio is increased up to 75% and latency is reduced up to 63%.