***Chapter 3***

**WEB CACHING AND PREFETCHING ALGORITHMS**

**3.1 Proposed approach:** As discussed in earlier, combining the both web caching and prefetching features gives better performance when compared with any of those features alone.

In this thesis, we applied both these features at proxy level cache. In this, we run two threads simultaneously: one for caching and other for prefetching. While retrieving data from either cache if it is present in cache or from the origin server (if the object is not present in the cache or staled one), the prediction engine predicts the next subsequent user’s request based on the past web history and fetches that request to store at the proxy cache.

Client-1

Prefetch module

Prediction module

Server-1

Client-2

Cache module

Server-2

Client-3

Figure 3.1:Proxy server cache along with prefetching

From the above figure, client requests will directly goes to the proxy server. At proxy level a cache module and prefetching module works simultaneously. For every request we checks the cache module for a valid copy. If it present, we increment the hit count otherwise miss count.

**3.2 Web caching:** In this, we added extra constraint “object life time” to basic Least Recently Used (LRU) algorithm to check whether the object is staled one or not. So that it becomes easy to clear the staled web objects present in the cache.

Web cache algorithm:

1. C<- Web cache
2. HC<- Hit count
3. MC<- Miss count
4. Initialize an empty queue Q to store page requests.
5. For each client i
6. Store the page requests P1, 2, 3…n  in Q
7. While (!Q empty())
8. Pi =front (Q)
9. If ((Pi is in web C)&& Check\_for\_freshness(Pi))
10. HC++
11. Else
12. MC++ and fetch the request from the origin server.
13. If(!C empty())
14. Store the request in C
15. Else
16. Remove the bottom object and store the new object in C at top position.

Check\_for\_freshness(Pi):

1. Ti<- Threshold value in milliseconds.
2. If (Pi>= Ti)
3. Return true
4. Else
5. Return false

**3.3 Prefetching:** The major part of the prefetching is prediction algorithm. Since we are assuming every web object is of same size and we are considering the content of web object, it is not possible to develop a prediction algorithm based on aspects like content based and clustered based etc. So the only possible way to predict the next user request is history based.

In this we developed the prediction algorithm based on the standard Markov patterns.

**Prediction model:**

In the figure given below we consider five web objects (P1, P2, P3, P4 and P5). By capturing the past history web sessions we developed the training sets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1ST-Order | Freq. | P1 | P2 | P3 | P4 | P5 |
| P1 | 5 | 0 | 0 | 0 | 2 | 1 |
| P2 | 5 | 4 | 0 | 0 | 0 | 1 |
| P3 | 3 | 0 | 1 | 0 | 1 | 1 |
| P4 | 6 | 0 | 1 | 0 | 0 | 1 |
| P5 | 5 | 0 | 3 | 0 | 2 | 0 |

WS1:< P3, P2, P1>

WS2:<P3, P5, P2, P1, P4>

WS3: <P4, P5, P2, P1, P5, P4>

WS4: <P3, P4, P5, P2, P1>

SW5: <P1, P4, P2, P5, P4>

Figure 3.2: Web sessions Figure 3.3: Prediction matrix

By looking at the web sessions, we constructed the above prediction matrix. In this matrix, every cell [i, j] represents the probability to access the jth column web page after the ith row web page and the frequency column presents how many times the corresponding page is accessed in the total web sessions.

If the current web page is p1, then by looking at the prediction matrix the algorithm gives p4 as output. Finally, the pre-fetch engine will fetch and stores this object in the cache. If two cell values in the same row are identical then the object with lowest id will be fetched.

By combing the above two approaches we did simulation in java language and compare the results with other algorithms.