Effective Web Access Latency Reduction Through Clustering Prefetching and Caching

M. B. Thulase¹, Dr. G. T. Raju²

¹Research Scholar, R&D Centre, RNSIT, VTU
²Professor , C S & E Department, RNSIT, Bangalore, Karnataka, India

Abstract-- The delay in serving a request will reduce the benefits of the information retrieved. Hence, it is a prime requirement to reduce the user perceived latency in accessing the web pages from a web site. Prefetching and caching techniques contribute a lot in reducing the user perceived latency. In this work a novel methodology that effectively reduces the web access latency through prefetching and caching is proposed. Seasonal web sites are taken into consideration and Web Objects are clustered based on the frequency of access and these clusters are prefetched and cached to facilitate the users near future requirement. An exhaustive study of effect of increase in size of the cache on hit ratio is made, and the results are analyzed.

Index Terms-- Web access latency, Prefetching and caching, Clustering and prefetching, Latency reduction

1. INTRODUCTION
With 40%[1] of the world population using the internet; increase in bandwidth will not completely address the delay problems[2]. Prefetching and caching techniques contribute a lot in reducing the user perceived latency.

Prefetching and caching
Prefetching proactively fetches the clusters from the server into the cache to facilitate the users near future needs. Caching alone caters to low latency improvement. Prefetching when combined with caching reduces the latency considerably. Studies have shown that when combined with caching, prefetching can improve latency upto 60%, while caching alone offers 26% latency improvement[3]. Prefetching involves prediction. The prediction mechanism plays a very important role in cache prefetching. Prefetching algorithms are either content based or access history based. Content based prefetching algorithm makes prediction based on the structure of the web site while, History based prefetching algorithms make prediction based on the page access behavior of the user in the past. In history based prefetching, prefetching can be made based on Mixed access pattern or Per client access pattern. In the proposed methodology mixed access pattern i.e. access pattern of all the clients together is used.

Caching reduces the load on the server, and by holding the clusters that are required in near future it also reduces the access latency. Cache can be located in client, server, or proxy server. As the cache size is limited caching involves replacement policies to handle the cache contents. It is required to purge the object that may not be needed in the near future and replace it with the predicted needy object. Efficient cache replacement algorithm needs to be chosen for a better performance. Least Recently Used (LRU) cache replacement algorithm offers a very good performance[3]. LRU replacement algorithm used in the proposed methodology, takes care of refreshing the cache regularly according to the user needs.

The two main components of the methodology proposed are prediction engine and prefetching engine. Prediction engine predicts the clusters to be prefetched based on the analysis of the user access pattern for a specified period of time. Prefetching engine prefetches the predicted clusters from the server during the server idle time. It is required to minimize the difference between the users requests(R) and the predictions(P) made by prediction engine, shown in fig 1. The cache hit(H) increases if the probability of the prediction tends to 1. Hence, H is directly proportional to Prediction P.

\[ H \propto P, \text{ therefore, } H = aP, \text{ where } a \text{ is a constant} \]

H also depends on the size of the cache(S).

\[ H \propto S, \text{ therefore, } H = bS, \text{ where } b \text{ is a constant} \]

From Eq. 1 and Eq. 2

\[ H = k\sqrt{SP} \text{ where } k = \sqrt{ab} \]

With \( P = 1 \), H depends only on the size of the cache. A study of variation in the value of H with respect to values of C is conducted in this work.

2. RELATED WORK
Caching reduces web latency. Web prefetching acts complimentary to caching; it can significantly improve cache performance and reduce the user perceived latency[9]. Recently few researchers used mining techniques to explore...
the browsing behavior of users in web services[10]. Few researchers take the advantage of the spatial locality shown by the web objects[11,12]. The prefetching and prediction engine are the two main components of prefetching algorithm for reducing the web latency. The prediction algorithm based on Maximum weight Matrix is proposed by Wenying Feng et al[4]. Prediction is made on the next request that follows. If it is correct the probability of the prediction is increased. The probability is used to modify the probability matrix, which is called pre-fetching weight matrix, used for prefetching. They claim 70% system hit ratio. A dynamic pre-fetching technique in which Web caching and pre-fetching techniques are integrated, is proposed by Achuthsankar S Nair et al[5]. In their technique, the number of subsequent links to be pre-fetched depends on the user’s interest in accessing the documents, cache contents, current bandwidth usage and maximum capacity of the existing network. Authors claim an increase in hit ratio of 40% – 75% and latency reduction by 20% - 63%. The adaptive prefetching scheme using web log mining in cluster based web, proposed by Heung Ki Lee et al.[6] suggests the Memory Aware Request Distribution component to distribute web workload to improve the efficiency in web pre-fetch. In semantics prefetching[7], the semantics hidden in web documents is used. Based on the document semantics, this approach is capable of prefetching documents whose URLs have never been accessed. Farhan et al.[8] proposed a combined Back-Propagation Neural Network as caching decision policy, and LRU as replacement policy.

From the studies, it is observed that caching with a good cache replacement policy can be used to prefetch the web object to reduce the web latency. In this work, with clustering on frequency and LRU replacement policy; the methodology ensures the use of both recency and frequency, the two important features of web caching and prefetching.

3. PROPOSED METHODOLOGY

The cache is divided into two levels; object-cluster cache(OC_cache) and cluster cache(C_cache). OC_cache stores the predicted clusters and also some of those objects whose locality of frequency is below the threshold(OC_threshold). Later if the access count of these objects exceeds the Move-To-Cache threshold(MTC_threshold), it is considered as most required cluster and the cluster is prefetched during server idle time. If the OC_cache is full then LRU replacement policy is applied to purge the cluster and store that into C_cache, if and only if the cluster is classified as needed by the classifier. The classifier uses the recency and frequency of access as the criteria for classifying. If the C_cache is full then again LRU replacement policy is used to purge the cluster for replacement. As LRU Cache replacement algorithm is employed, and clustering is made on frequency: frequency and recency, the two important factors of prefetching and caching are considered in the methodology proposed. Block diagram of the proposed methodology is shown in Fig2. and the algorithm proposed is shown in fig 3.

Notations used in the algorithm

CC : Cluster Center.
CC_threshold : Cluster Center threshold.
O : object that is requested by the cluster.
MTC : Access count of the objects whose locality of frequency is below the threshold.
MTC_threshold : Move To Cache Threshold, Minimum access count for considering the cluster of the object whose locality of frequency is below the threshold to be prefetched. Set to 3 in the experiments conducted.
RECLUSTER : Access count of the object whose locality of frequency is below the threshold, used to suggest reClustering requirement.
RECLUSTER_threshold : Minimum RECLUSTER count to suggest reclustering.
OC_cache : First level cache, stores the clusters and also those objects whose locality of frequency is below the threshold.
C_cache : Second level cache, stores the clusters that are purged from the OC_cache but are classified as needed.
Cachehit : Hit Count; total number of cache hits.
Update_Cache(C, OC_cache, C_cache) : Function which inserts object or cluster fetched from the server into OC_cache. If OC_cache is full, then LRU is applied on OC_cache and the expelled cluster is stored into C_cache, based on the output of classify(). If C_cache is full, LRU is applied on C_cache and the cluster selected is expelled from the cache.
Classify(expel_item) : Function to classify the cluster. If CC(expel_item) > CC_threshold and the number of times the cluster is accessed is greater than the frequency threshold, the expel_item is classified as needed.
If CC(expel_item) < CC_threshold but the frequency of access of that cluster is maximum amongst all the clusters in C_cache then the cluster is classified as needed.
get_cluster(O) : Function that returns the cluster center of the object requested by the user.
Fig. 2. Block diagram for clustering, prefetching and caching mechanism
Proposed Algorithm

1. Web log of the website under consideration is preprocessed. The required information such as the recency and frequency of access of the web objects are extracted. Web objects are clustered and the cluster centers are identified.

2. For each Web Object O requested by the user
   Begin
   "C = get_cluster(O)"
   If O in object cache
   "Serve it to the user"
   Update MTC
   Update RECLUSTER
   Update cachehit
   If (MTC > MTC_threshold)
   "Fetch the cluster C of object O during the browser idle time"
   Update the information of C
   "Update_Cache(C, OC_cache, C_cache)"
   Else
   "Found = search(C, OC_cache, C_cache)"
   If (Found)
   "Update cachehit"
   Update the information of C
   Else
   "Fetch O from the server and serve it to the user"
   If (CC(O) >= CC_threshold)
   "Fetch the cluster C of object O during the browser idle time"
   Update the information of C
   "Update_Cache(C, OC_cache, C_cache)"
   Else
   "Store O in the Object cache"
   Update MTC
   Update RECLUSTER
   End

3. If (RECLUSTER > RECLUSTER_threshold)
   Display “Reclustering suggested”

Algorithm to update l1 and l2 cache whenever a cluster has to be brought into the cache memory
Update_Cache(C, OC_cache, C_cache)
Begin
If (OC_cache is FULL)

Exbel_item = object identified for replacement using LRU replacement algorithm on OC_cache
If (Classify(exbel_item) = need)

then

Use LRU replacement algorithm on C_cache and store Exbel_item into C_cache
else

Expel(Exbel_item)
End

Algorithm to classify whether the expelled object from OC_cache has to be stored into C_cache or not.
Classify(exbel_item)
Begin
Based on the Recency and Frequency information, if Exbel_item is classified as need
then

return (true)
else

return (false)
End

Fig. 3. Algorithm for clustering prefetching and caching
4. EXPERIMENTAL RESULTS
Experiments are conducted on the synthetic data. A website with 100 pages with 10 clusters is considered. Log files for a week, month, quarter, and half yearly are considered for the analysis, to make the clusters, and to record the variations of the hit ratio with respect to the cache size. As the methodology is developed for a seasonal website, clusters are made based on the frequency of access. LRU replacement policy used, takes care of the recency of access and refreshing the cache. For the experimental purpose the cache size is measured in terms of the number of clusters the cache can hold. The results of the experiments for varied cache size, for varied analysis period, are tabulated and are depicted using the charts in Fig 3.

5. CONCLUSIONS
In this work, user’s log file is preprocessed and clusters are constructed based on the frequency of access. Prefetching the clusters during the server idle time, is decided based on the frequency of access of the objects in the cluster. LRU cache replacement algorithm takes care of cache refreshing.

The results show that in each case the cache hit ratio considerably increases with the increase in the size of the cache. As the seasonal website is taken into consideration, the time period used for analysis also plays an important role on the hit ratio. As the analysis period increases the hit ratio drops due to the effect of the changes in the season. For the website under consideration analysis period of one quarter shows the better performance. The time interval for

---

Fig. 3. Analysis of increase in hit ratio with respect to cache size
analysis is decided based on the contents of the web site and the period of change in the users access pattern. With a better analysis period, the methodology gives a good hitratio and thereby reduces the effective Web access latency.

REFERENCES