ALLEVIATION OF STALE INFORMATION IN WEAK CACHE COHERENCE FOR WIRELESS WWW

*Nazrul M. Ahmad, **Kim-Geok, Tan, and ***Ali M. Abdelrahman

**Faculty of Information Science & Technology (FIST)*
Multimedia University, Jalan Ayer Keroh Lama, 75450 Melaka, Malaysia

***Faculty of Engineering & Technology (FET)*
Multimedia University, Jalan Ayer Keroh Lama, 75450 Melaka, Malaysia

****Faculty of Engineering & Technology*
University of Gezira, Wad Medani, Sudan

ali_abdelrahman@yahoo.com
nazrul.muhaimin@mmu.edu.my
kgtan@mmu.edu.my

ABSTRACT

In this paper, we identify the client polling as a cache coherence mechanism that best satisfies the requirements and improves the performance of web services over wireless networks. However, this mechanism suffers from degradation of client satisfaction by potentially sending stale information. Therefore, we propose a conceptual framework, Multilevel Pre-fetching (MLP), to alleviate the stale information forwarded by the web cache to clients. MLP is responsible for improving the level of freshness among the cached objects by pre-fetching the most frequently accessed objects.

Keywords: Cache Coherence, Pre-fetching, Client Polling, Wireless Networks, World Wide Web.

1.0 INTRODUCTION

Web caching is deployed to store web objects that are recently fetched from the remote server and make them close to the clients. It provides an opportunity to refine the HyperText Transfer Protocol (HTTP) limitations

over wireless networks (Cao, 2003; Wang, Das, Che, & Kumar, 2004). The HTTP causes web services to suffer from performance degradation (Tian, Xu, & Ansari, 2005), i.e. it needs an excessive number of round trip propagation times between the client and web server for transferring a document over a low bandwidth. Thus, it significantly increases the file transfer latency. Another factor that limits the capability of HTTP over wireless network is the radio signal dropout, caused by handoff, i.e. the HTTP is a stateless protocol that is unable to resume the file transfer when a connection is broken and re-established.

There are several incentives for having a caching system in a network (Wang, 1999). Web caching reduces bandwidth consumption, thereby decreases network traffic and lessens network congestion. It also minimizes access latency by storing frequently accessed documents at nearby caches. Furthermore, it potentially decreases the workload of a web server. However, there is a side effect whereby the client may receive stale data due to the lack of proper web cache updating. Therefore, cache consistency mechanisms are implemented to ensure that cached copies are frequently updated to keep consistency with the original data.

Basically, there are two cache coherence techniques (Wang, 1999; Cao & Ozsu, 2002; Cao & Liu, 1998): weak cache coherence and strong cache coherence. Weak cache coherence is the model in which the access time is significantly reduced, but a stale document might be returned to the client (Cao & Liu, 1998). However, with the fact bandwidth is limited and expensive in wireless networks, it is too costly for the clients to keep refreshing the web browser every time they receive stale documents. Therefore, if the clients have strict requirements on the freshness of documents or a stale copy is not tolerable, then a strong cache coherence mechanism is necessary (Doswell, Abrams, & Varadarajan, 2001). Strong cache coherence enforces the freshness of document all the time at the expense of more control messages over the network. However, the flooding of control messages substantially consumes bandwidth and adds to the server loads, not to mention that the client might experience longer response time (Cao & Ozsu, 2002).

Many caching systems apply weak cache coherence, believing that methods such as Time-to-Live (TTL) and client polling are sufficient and most appropriate for web caching (Cao & Liu; 1998; Gwertzman & Seltzer, 1996). In either case, modification to the web object before its TTL expires or between two successive polls causes the web cache to return stale objects. On top of that, weak cache coherence mechanisms are easily deployed and supported by HTTP

(Doswell et al., 2001; Gwertzman & Seltzer, 1996; Cohen & Kaplan, 2001). Based on these facts, we decided to investigate the possibility of deploying weak cache coherence mechanism particularly client polling in the web cache for wireless web services. Our motivation is to find the solution on how the mechanism could be improved to alleviate stale web objects forwarded to clients by the web cache. In particular, this paper proposes the conceptual framework of improving web weak cache coherence by introducing Multilevel Pre-fetching.

The rest of this paper is organised as follows. In Section 2, we briefly review the principle of weak cache coherence mechanisms and pre-fetching. In Section 3, the performance of weak cache coherence particularly client polling is evaluated and discussed. Section 4 introduces the framework of our proposed Multilevel Pre-fetching (MLP) to alleviate the stale information in web cache. Finally, this paper is rounded off with a conclusion.

2.0 LITERATURE REVIEW

A considerable amount of work has been done in web caching in order to improve its effectiveness. For instance, web cache location, cache routing, dynamic data caching, cache architecture, cache replacement, cache coherence, and pre-fetching (Wang, 1999). This section focuses on the previous work with primary emphasis on weak cache coherence and pre-fetching.

2.1 Weak Cache Coherence

Caches provide documents at lower access latency with a side effect: every cache sometimes provides clients with stale pages (Cao & Liu, 1998). To prevent stale information from being transferred to clients, a web cache must ensure that locally cached data is consistent with that stored on the server. The exact cache coherence mechanism must be employed by the web cache.

Under Time-To-Live (TTL) approach, the origin server assigns a TTL value for each object, which could be any value that is reasonable to the object itself, or to the content provider. This value is an estimated lifetime of the cached object, after which the object is regarded as invalid. Whenever TTL expires, the next request for the object will be forwarded to the origin server. Gwertzman and Seltzer (1996) initially used a flat distribution for their simulation, which means they assigned all objects with equal TTL values. This resulted in poor experimental performance.

On the other hand, an adaptive TTL takes the advantage of the fact that file lifetime distribution is not flat. If a file has not been modified for a long time, it tends to stay unchanged. Gwertzman and Seltzer (1996) have concluded that globally popular files are the least likely to change. By using adaptive TTL, the probability of stale documents has been kept within reasonable bounds (<5%) (Gwertzman & Seltzer, 1996).

A client polling (CP) (Gwertzman & Seltzer, 1996) is a typical example of weak cache coherence. This means the client (web cache for this context) periodically checks back with the server to determine if the cached objects are still fresh. The cache issues an if-modified-since (IMS) request with the LAST-MODIFIED response header value (indicating last modification time of the object). The server then checks to see if the object has changed since the timestamp. If so, a 200 HTTP response code is sent along the fresh object. Otherwise, the server returns a code of 304 (document not modified). Alex File Transfer Protocol (FTP) cache (Cate, 1992), a form of CP, uses an update threshold, θ, to determine how frequent the cache would poll the server. The θ is expressed as a percentage of the object's age. The age is the time since the last modified time of the object. An object is invalidated when the time since the last validation exceeds the θ times the object's age (Gwertzman & Seltzer, 1996). An invalidated object causes the cache to send an IMS to the server to check whether a file transfer is necessary. Otherwise, the cache sends the valid object to the client.

2.2 Pre-fetching Techniques

Previous research has shown that the maximum cache HIT rate can be achieved by any caching algorithm is usually no more than 40% to 50% (Wang, 1999). The reason is that most people browse and explore the web, trying to surf new information. One way to further raise the caching HIT ratio is to anticipate future document requests and pre-fetch these documents in a web cache.

Kroeger, Long, & Mogul (1997) investigated the performance limits of pre-fetching between web servers and web caches. They have shown that the combining of proper caching and prefect pre-fetching at the web caches can at least reduce the client latency by 60% for high bandwidth clients. Markatos and Chronaki (1998) claimed that the web servers regularly push their most popular documents to web caches, which in turn push those documents to clients. They evaluated the performance of the strategy using several web server traces, and found that this technique can anticipate more than 40% of the client request.

The technique requires cooperation from the web servers, although the study did not evaluate the client latency reduction. Gwertzman and Seltzer (1994) discussed a technique called Geographical Push-Caching where a web server selectively sends its documents to the caches that are closest to its clients. The focus of the study was on deriving reasonably accurate network topology information to select caches.

Padmanabhan and Mogul (1996) analysed the latency reduction and network traffic of pre-fetching using web server traces and trace driven simulation. The prediction algorithm they used is based on the Prediction-by-Partial-Matching (PPM) data compressor with prefix depth of one. The study showed that pre-fetching from web servers to individual clients can reduce client latency by 45% at the expense of doubling the network traffic. Crovella and Batford (1998) analysed the network effects of pre-fetching. They have shown that pre-fetching can reduce access latency at the cost of increasing network traffic and burst, which may increase the overall network delays. Thus, they proposed a rate-controlled pre-fetching scheme to minimise the negative effect on the network by minimising the transmission rate of the pre-fetched documents.

Most of the previous studies have shown that cache effectiveness can be significantly improved by deploying cache coherence or a pre-fetching mechanism in the web cache. This is due to the fact that, a cache coherence mechanism successfully minimises the number of stale documents that could be returned to the clients and increases the effectiveness of pre-fetching mechanism to reduce the document retrieval latency. With these significant improvements, we then motivate this work to merge both mechanisms as a new cache refreshment mechanism, Multilevel Pre-fetching (MLP).

3.0 PERFORMANCE OF WEAK CACHE COHERENCE

Using an event-driven simulator NS2 (Virtual InterNetwork Testbed, 1997), we replayed the server trace from EPA (ACM SIGCOMM, 2000) in the wireless LAN 2 Mbps. It consisted of 20 wireless nodes connected to a web cache at the edge of a wireless network. Then, the web cache is attached to the server. The web cache deploys a cache coherence mechanism to ensure the consistency of cached objects with those stored on the server. Specifically, we used CP that is based on the Alex protocol. We ran CP by varying θ in order to evaluate the impact of object's age on the effectiveness of CP.

3.1 Limitations of Weak Cache Coherence

Table 1: Simulation results based on EPA WWW Web Server Trace, NC [18,19]

Parameter	Weak Cache Consistency Update Threshold, θ								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
GET Requests	33285	33285	33285	33285	33285	33285	33285	33285	33285
Hits	14540	18061	20372	22350	24031	25118	26096	26940	27801
Stale Hits	876	2395	4157	7290	13482	14216	18763	19825	22520
IMS	14956	11452	9147	7172	5490	4410	3435	2600	1738

Table 1 presents the negative impact for deploying web cache that provides a weak cache coherence mechanism for wireless WWW. The results have shown that the mechanism could produce a stale rate (Stale Hits $\prod$ GET Requests) of less than 3% but at the expense of higher IMSs. That means, the web cache more often needs to check the validity of the requested objects. Thus, it significantly increases the response time due to extra processing in the web cache for invalidation process.

In general, this mechanism certainly satisfies the behaviour of static web pages, which change infrequently. But in order to meet the demand of exponential growth of the dynamic web pages, the mechanism needs to generate a lot of IMS control messages for invalidation purposes. Also the mechanism must ensure the return of fresh documents to the client. Thus, a trade-off has to be made between gaining the advantages of having web cache if most of the cached objects are static, and its negative impact on network traffic if the nature of the cached data is dynamic, where the client is satisfied by getting fresh documents.

3.2 Proposed Work

The primary interest of this paper is to develop a cache coherence mechanism for the wireless network. Fortunately, client polling offers the best foundation

for this purpose, where the lifetime of the cached object is not flat. However, this algorithm has two major drawbacks: firstly, the cache may return a stale copy if the object is changed in the original server while the cached copy is still considered valid. Secondly, the cache can invalidate a data copy, which is still valid in the original server. Therefore, we propose a framework of Multilevel Pre-fetching (MLP) algorithm to improve the level of freshness among the cached objects to refine these drawbacks.

4.0 MULTILEVEL PRE-FETCHING (MLP)

The numbers of stale HITs produced by the client polling drastically increase as the web cache increments its update threshold. This means, the client tends to receive a lot of stale information if the web cache minimizes the frequency of polling the server. Thus, some modifications to the algorithm have to be done to reduce the number of stale objects by improving the level of freshness among the cached objects. The following subsections elaborate the proposal in detail.

4.1 Adoption of Pre-fetching Algorithm

Pre-fetching is a technique to increase the chances of getting documents from the web cache. This work tends to exploit the pre-fetching concept to enhance the coherence mechanism, particularly client polling. Pre-fetching could be used to reduce the number of stale documents forwarded to clients. This is initiated by the web cache on the cache-server channel.

Multilevel Pre-fetching (MLP) is proposed in conjunction with the client polling mechanism to refine its drawbacks. The primary objective of MLP is to pre-fetch cached objects with their lifetimes still valid. This is due to the fact that cached objects whose lifetimes do not expire, but they might be modified in their origin server. Thus, there is a possibility for the client to retrieve stale documents mainly because of wrong decision by the client polling. The MLP is an extension to the client polling mechanism and responsible for validating the cached objects without waiting for requests from the clients.

Introduction of pre-fetching without a proper control mechanism may add new traffic and processing overhead to the cache and server. More aggressive pre-fetching not only keeps cache and servers synchronized for longer periods of

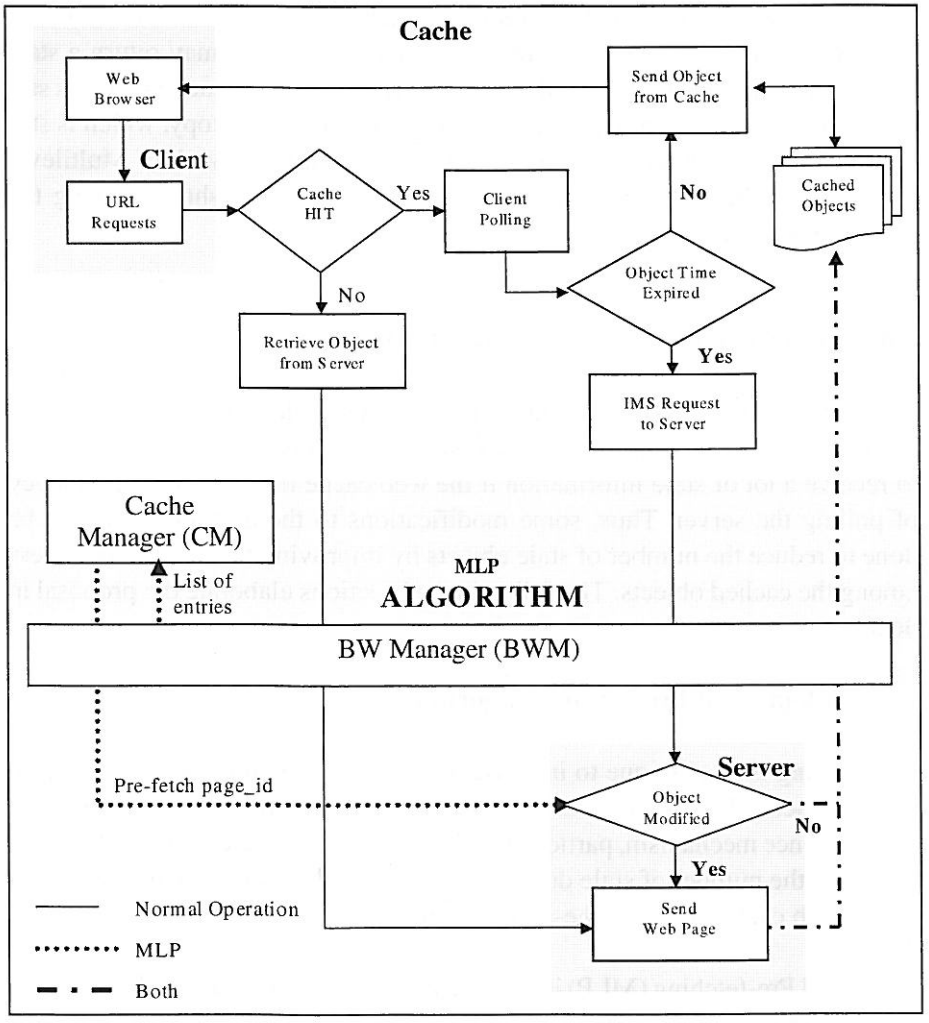

Fig. 1: Enhanced Client Polling with MLP Algorithm

time and improving the cache HIT rate while reducing the stale HIT rate, but it also increases network costs, server loads, and cache loads. Therefore, to avoid the cache-server being overloaded, MLP is only to be executed whenever the cache-server is idle or under utilized. Fig. 1 shows the enhanced client polling with MLP algorithm architecture. Basically, there are two major components added to the conventional client polling mechanism, Bandwidth Manager (BWM) and Cache Manager (CM).

4.2 Bandwidth Manager and Cache Manager

Bandwidth Manager (BWM) operates as a front-end of the web cache and it is assigned to take care of traffic monitoring especially on file transfer activities at the cache-server channel. In general, BWM monitors, computes link utilization, and estimates available bandwidth. Then, BWM is responsible for triggering the Cache Manager (CM) whenever traffic condition is good enough to perform pre-fetching operations. BWM keeps the records of all cache entries and prepares a list of entries that have been recently fetched from the server. It uses a heap as the data structure to store the cached object's header information. For performance reasons, BWM does not store the real object in it; only the object's header is stored. Meanwhile, CM is responsible for pre-fetching and validating cached objects before they are likely to be requested again by the client, i.e. it maintains freshness of web pages in the cache.

4.2.1 Maintaining Pre-Fetching List of Entries

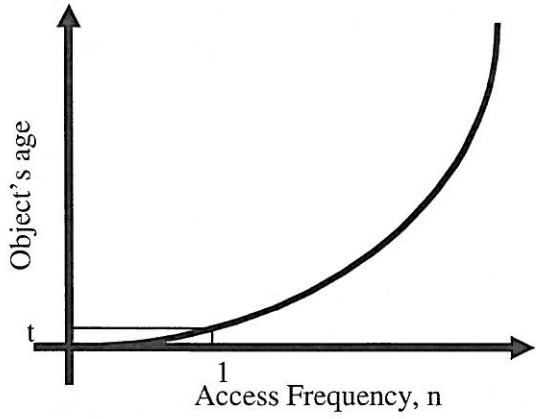

Fig. 2: Cached Object Lifetime

Every cached object is to be assigned with the page lifetime by BWM. Fig. 2 illustrates the relationship between the cached object's age and the access frequency. Cached object's age is defined as:

Cached Object Age $= t_0 \times f \times \exp(X)$

where t_0 is the initial age assigned to the cached object, n is the access frequency and

X = [(current_time – entry_time) / current_time]

BWM saves the following information of the cached objects: object id (*page_id*), object's size, access frequency (*f*), and the time when a web cache stores the object at the first entry (*entry_time*). If the object is accessed more frequently by the client, the age of the object will be allocated with a longer period of expiry. Each time the object is being accessed, a new object lifetime will be imposed.

Fig. 3 shows the expiration checking by BWM in order to prepare the list of entries. Every *mt* second or *validated_time*, the BWM will prepare a list of entries based on the existing cached objects. *Validate_time* is the time when BWM confirms the validity of the cached object. If the object's age is still valid, BWM assigns a *page_id* of the cached object into the list of entries. However, if the object's age has expired the object will be dropped from the list and re-considered again in the next cycle. After validating all the cached objects, BWM forwards the list of entries to CM for pre-fetching.

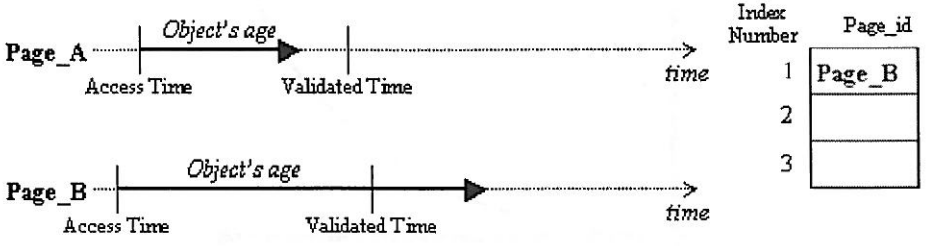

Fig. 3: Cached Object Validity Checking by BWM

For Example, consider two cached objects, *Page_A* and *Page_B*, whose *entry_time* is day 1 and day 15 respectively. The initial age assigned to those objects is assumed to be 2 days. At day 20, both cached objects are accessed again by the clients. Therefore, BWM will immediately impose new ages to those objects. This means, *Page_A* will be marked as valid for the next 10 days and *Page_B* for the next 5 days. Now let us assume that after the six days have elapsed, BWM is ready to prepare the list of entries for CM. If the object's age

is still valid (e.g. *Page_B*), BWM assigns the particular object to the list of entries as shown in Fig. 3.

4.3 Implementation of MLP

MLP invokes a timer to schedule the events as follows. After every *t* second, BWM initiates the computation on the bandwidth utilization of the web cache. BWM estimates network traffic generated by the web cache and message bytes sent by the web server to the web cache over a specific period of time. It is used to determine whether the bandwidth is sufficient enough to perform pre-fetching operations. Whenever the link utilization exceeds 20% of the available bandwidth, all pre-fetching activities will be suspended. Whenever the bandwidth is considered available, BWM forwards the list of entries to CM for pre-fetching at a regular interval, *validated_time*, i.e., *mt* second. However, if the bandwidth is unavailable, BWM will delay the forwarding until it computes the link utilization in the next *t* second.

Upon receipt of a list of entries from BWM, CM starts to create a pre-fetch tree which consists of an aggregate of multilevel sub-nodes. CM assigns each sub-node in the pre-fetch tree with one *page_id* in the list of entries. In the next step, CM traverses the pre-fetch tree by visiting sub-node by sub-node to validate the object's freshness based on the information obtained in the list of entries. At each *prefetch_interval*, i.e. *nt* where $nt << mt$, CM invokes the Alex protocol to determine whether object's freshness is valid. If the object is valid, CM pre-fetches the respective *page_id* in the web server by sending IMS. Otherwise, the sub-node is terminated and CM proceeds to the next sub-node in the upcoming *prefetch_interval*. CM repeats the same procedure until it reaches the last sub-node in the pre-fetch tree.

The CM expects the arrival of the next list of entries after *validated_time* or more, and accordingly it creates a new pre-fetch tree, even though the existing tree still performs the pre-fetching process. Thus, there is possibility of several trees to be run concurrently. The idea behind having this configuration is to cope with the fact that dynamic objects are frequently modified in the web server. Hence, web cache needs to validate the consistency of the cached objects regularly to make sure those objects are far from expiring. By lowering the *validated_time*, CM more aggressively performs the pre-fetching operations in order to keep both the web cache and web server synchronized, i.e. CM regularly updates all the cached objects listed in the list of entries. However, frequent

pre-fetching comes with the expense of overloading more traffic on the channel. Therefore, BWM is assigned to monitor these activities and once the channel is congested, BWM signals CM to terminate all existing pre-fetch trees. By eliminating the trees, bandwidth is completely reserved for normal operation. Once, the bandwidth is available again, the pre-fetching process will resume.

4.4 Construction of Multilevel Sub-Nodes

This section discusses the construction of pre-fetch tree for MLP, which is constructed as follows. Once CM receives a list of entries from BWM, the first sub-node is generated as a root and it is assigned with a random index number from the list of entries. For instance, as shown in Fig. 4, the sub-node A is created at level L_1 of the first pre-fetch tree T_1 and it is assigned a random index number Z which lies between 0 and y, where y is the last index number in the list of entries. This index number, Z, corresponds to the *page_id* of the cached object, which is in this case referred to page_09. Similarly, the second sub-node B is generated at level L_2 and assigned with any index number within the range from 0 to Z. This process is repeated until it covers the entire cached object's header information stored in the list of entries. After the pre-fetch tree is completely built, CM waits until the next *prefetch_interval* to perform pre-fetching operations. At *prefetch_interval*, starting with the root, CM picks *page_id* of the sub-node and it searches the corresponding object's header information in the list of entries. Then, CM checks the object's freshness by invoking the Alex protocol. If the object is valid, CM pre-fetches the respective *page_id* from the web server. Otherwise, the sub-node will be terminated. In the upcoming *prefetch_interval*, CM traverses to the next sub-node in the pre-fetch tree. CM repeats the same procedure until it visits the last sub-node of the tree.

Meanwhile, CM can start creating another pre-fetch tree whenever it receives a new list of entries from BWM. Fig. 5 illustrates the construction of the second pre-fetch tree T_2 at the next *validated_time*. For the second tree, CM expects the total number of cached objects in the list of entries equal or more than the first list. List of entries contains more cached objects if there are new GET requests from clients that constituted cache MISS within the first and second *validated_time*, and so forth. In brief, CM validates one sub-node on every *prefetch_interval*. With a short *prefetch_interval*, CM visits sub-node by sub-node in the pre-fetch tree at a faster rate.

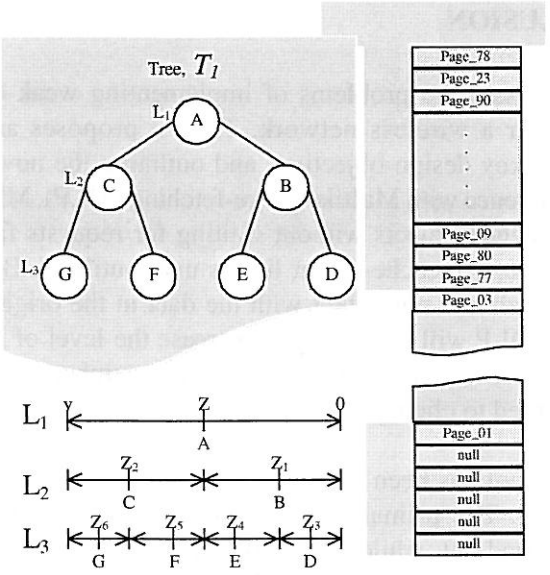

Fig. 4: Construction of Multilevel Sub-nodes

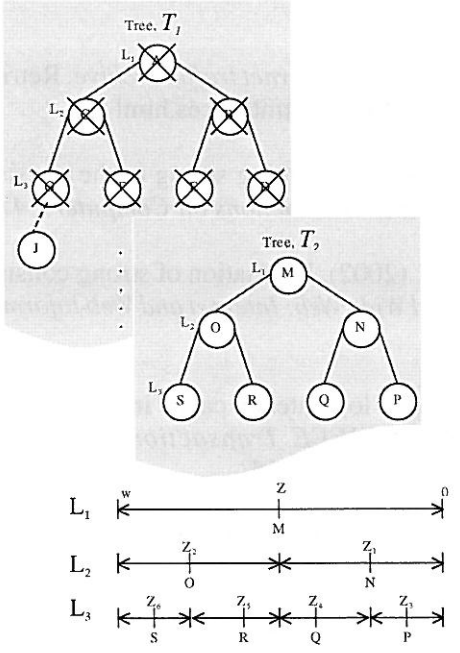

Fig. 5: Creation of T_2 in Concurrent with T_1

5.0 CONCLUSION

This paper addresses the problems of implementing weak cache coherence mechanisms over a wireless network. Then it proposes an alternative by highlighting the key design objectives and outlining the novel framework of weak cache coherence with Multilevel Pre-fetching (MLP). MLP is responsible for validating cached objects without waiting for requests from clients. It is executed whenever the cache-server link is under utilised. By doing this, the cached object is always consistent with the data in the original server. Thus, introduction of MLP will considerably increase the level of freshness among the cached objects, which in turn results in less number of stale objects that could be forwarded to clients.

The simulation tool has been enhanced with MLP in conjunction with weak cache coherence. The primary test of the proposed architecture has been successfully carried out, while an intensive performance evaluation is still in progress.

REFERENCES

ACM SIGCOMM (2000). *The internet traffic archive*. Retrieved Mar 25, 2002, from http://ita.ee.lbl.gov/html/traces.html

Cao, P. & Liu, C. (1998). Maintaining strong cache consistency in the world wide web. *The IEEE Transactions on Computers, 47*(4), 445-457.

Cao, L.Y. & Ozsu, M.T. (2002). Evaluation of strong consistency web caching techniques. *World Wide Web: Internet and Web Information Systems, 5*(2), 95-123.

Cao, G. (2003). A scalable low latency cache invalidation strategy for mobile environments. *The IEEE Transactions on Knowledge and Data Engineering, 15*(5), 1251-1265.

Cate, V. (1992). Alex – A global file system. *Proceedings of the USENIX File System Workshop, Ann Arbor,* 1-11.

Cohen, E. & Kaplan, H. (2001). Refreshment policies for web content caches. *The 20th Annual Joint Conference of the IEEE Computer and Communications Societies,* 1398-1406.

Crovella, M. & Batford, P. (1998). The network effects of pre-fetching. *Proceedings of the IEEE INFOCOM '98 Conference on Computer Communications*, 1232-1239.

Doswell, F., Abrams, M., & Varadarajan, S. (2001). The effectiveness of cache coherence implemented on the web. *Proceedings of the Workshop on Caching, Coherence & Consistency. Sorrento, Italy.*

Gwertzman, J. & Seltzer, M. (1994). The case for geographical pushing-caching. *Proceedings of The HotOS Conference.* Retrieved Jan 21, 2002, from ftp://das-ftp.harvadr.edn/techreports/tr-34-94.ps.gz.

Gwertzman, J. & Seltzer, M. (1996). World-wide web cache consistency. *Proceedings of the 1996 USENIX Technical Conference, San Diego,* 141-152.

Kroeger, T. M., Long, D. D. E., & Mogul, J. C. (1997). Exploring the bounds of web latency reduction from caching and pre-fetching. *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems, Monterey, CA.*

Markatos, E. P. & Chronaki, C. E. (1998). A TOP-10 approach to pre-fetching on web. *Proceedings of the 8th Annual Conference of the Internet Society, INET'98, Geneva, Switzerland.*

Padmanabhan, V. N. & Mogul, J. C. (1996). Using predictive pre-fetching to improve world wide web latency. *Proceedings of the ACM SIGCOMM '96 Conference on Communications Architectures and Protocols, Stanford University, California,* 22-36.

Tian, Y., Xu, K., & Ansari, N. (2005). TCP in wireless environment: Problems & solutions. *The IEEE Communications Magazine, 43*(3), S27-S32.

Virtual InterNetwork Testbed, VINT (1997). *Network simulator, NS-2.* Retrieved Aug 5, 2001, from http://www.isi.edu/nsnam/ns.

Wang, Z., Das, S. K., Che, H., & Kumar, M. (2004). A scalable asynchronous cache consistency scheme for mobile environment. *The IEEE Transactions on Parallel & Distributed Systems, 15*(11), 983-995.

Wang, J. (1999). A survey of web caching schemes for the internet. *The ACM SIGCOMM Computer Communication Review, 27*(5), 36-46.