Web-based Energy-efficient Cache Invalidation in Wireless Mobile Environment

Y.-K. Chang, M.-H. Hong, and Y.-W. Ting
Dept. of Computer Science & Information Engineering,
National Cheng Kung University
{ykchang,p7691106,p7893113}@mail.ncku.edu.tw

Abstract

More and more users use mobile devices to retrieve dynamic web pages in the wireless networks. Caching dynamic pages becomes very important due to the power constraint of mobile devices. In this paper, we first introduce a framework to cache and manage the dynamic web pages on the server side such that these dynamic pages can also be cached in the mobile devices. Then we propose a stateful IR-based approach which only records two numbers, the number of web pages updated and the number of web pages updated and also queried after they are updated on the server in an IR interval. Recording these two numbers dramatically reduces the IR size. The experiments show that our proposed approach combined with the Timestamp and UIR algorithms consumes the power around 40~47% less than the original Timestamp and UIR. Also, our method performs better than the Perfect Server that has the full knowledge of the contents stored in all the mobile client's caches in terms of power consumption.

Keywords: Cache consistency, dynamic web pages, invalidation report, mobile environments, power conservation.

1. Introduction

HTTP is used by Web servers, proxies, and browsers for the transfer of Web documents. It was originally designed for browsing static documents. However, during the last decade, the development of World Wide Web is changing from static to dynamic pages. Dynamic contents are constructed based on personalized service and request parameters at the time the document is requested. For those dynamically generated documents that may change on every request, the expiration time is always set to "now" to disable cache. Although the web pages generated by server-side scripts are called "dynamic", they may not change in every second. A lot of dynamic web pages are intrinsically static, not changed in a period of time. The same pages have been transmitted over the same network links again and again to thousands of different users. Caching can be very effective at reducing network bandwidth consumption as well as balancing servers' load.

The challenge in designing applications that access dynamic data (e.g., stock, weather) is to ensure that displayed values are coherent with the data on the server. We address the coherent problems that arise from accessing dynamic web data when using mobile devices. Previous works focus on how to maintain consistency between the server and proxy [6], [7]. They proposed to use push, pull or hybrid schemes to maintain the data consistency. Since the proxy and server are on the wired network, the proxy can receive updates immediately. Proxy maintains the data consistency in wired network and informs stale pages to mobile clients through the wireless networks.

Because of the space limit, we do not describe the existing web cache schemes and invalidation report (IR) strategies [1] in this paper. Web caches are used to cache dynamic web documents while invalidation strategies broadcast invalidation reports to invalidate stale pages on the client side.

In this paper, we extend the dynamic page caching framework to cache dynamic pages accessing remote databases. Based on the access log of the web server and query and update logs of the database server, the dynamic web pages that access databases can be easily cached on the server side and the client side. In the proposed framework, Bloom filter is also used to manage the data consistency between the server and clients.

Next, we shall propose a stateful approach that can avoid broadcasting the timestamps of some web pages that are queried after their updates. The proposed stateful approach can be integrated to the existing IR-based caching schemes in the wireless environment to reduce the power consumption. We also conduct simulations to show that the proposed scheme can perform better than the original IR-based schemes.

The rest of this paper is organized as follows:. The Section 2 introduces a framework and proposed a stateful IR approach. The simulation model is

1550-445X/05 $20.00 © 2005 **IEEE**

described in Section 3. Finally, the conclusion is given in the last Section.

2. Proposed scheme

In this section, we first introduce our previous work [5] to how to make dynamic pages cacheable and then extend it by considering how the validity of cached pages is affected when databases queried by these web pages are updated. Second, we use Bloom filter to efficiently broadcast the hash values rather than full URLs to indicate the stale web pages. Finally, we propose a stateful approach to improve the performance of the methods using the invalidation report.

In order to cache dynamic pages on the mobile clients and maintain the cache consistency efficiently, the following tasks must be completed: (1) Making dynamic web page cacheable. (2) Knowing which cached pages on the server side are stale. (3) Maintaining page consistency on the client side.

2.1. Making dynamic web page cacheable

We make the dynamic pages cacheable by the following framework. The proposition we make is that the cached web pages must be fresh all the time. All the stalled web pages will be updated or deleted by the cache manager that acts as a backend process to perform the cache management asynchronously. The system architecture of the proposed caching system is shown in Fig. 1.

All cached pages are stored in a directory called *Cache Directory*. If the requested page exists in *Cache Directory, Web Server* could response it directly. Otherwise, we first trigger *Application Server* to generate the requested page. The newly generated page prefixed with appropriate HTTP Cache-Control headers is then replied to the client. The replied page is also stored in *Cache Directory* as a static file.

Two URL format types are used in proposed caching system. The URL format type A (e.g., http://host/abs_path/page?k1=v1&k2=v2) is the traditional URL format with the query string when the client requests a dynamic web page using the GET method. Since type A URL contains a question mark (?), the client side cache usually does not cache this page. In order to remove the question mark from the URL of a page and this make it cacheable, we define a static URL format called type B URL (e.g., http://host/abs_path/page!k1=v1&k2=v2.html). Type B URLs are the ones that are released to the public and used by users. Type A format is only used internally in the proposed caching system. We reply the client's requests in static page format, so they can be cache by Web browsers on the client sides and Proxy servers.

Embedding the pairs of keyword and value in URL using GET method loses the flexibility of users' inputs. This is where POST method comes from. To imitate the actions of POST method, we allow users input the keywords and values but still using type B format. This can be done by a simple javascript code [5].

The request URL could be for a dynamic page or for a static page. URL Switch first checks whether or not the requested page exists in *Cache Directory*. If the requested page exists, it is returned to the client by *Web Server* directly. When the requested page does not exist, and the type B URL is first converted to Type A format and the request is passed to *Application Server* to generate and cache the requested page.

For example, if the client requests for /cachedir/calculate.php!v1=2&v2=3.html and this does not in *Cache Directory*. Then URL Switch will translate the URL into /calculate.php?v1=2&v2=3. *Application Server* first call calculate.php to generate the page, attach it with appropriate Cache-Control headers, and store the dynamic page into Cache Directory with the name calculate.php!v1=2&v2=3.html. *Application Server* must inform *Cache Manager* that the dynamic page is stored as a static page in *Cache Directory*. So, *Cache Manager* can maintain this newly generated static page in *Cache Directory*.

Cache Manager is in charge of the cache files in *Cache Directory*. It is behind the web server. Only *Application Server* can inform it about the newly generated pages. Database will change with time, so we will get different result at different times if queried with the same input arguments. Our *Cache Manager* uses mapping file and update log file to maintain cache pages consistency in *Cache Directory*.

2.2. Knowing which cached pages are stale

Here we introduce the method for *Cache Manager* to maintain the cached pages in *Cache Directory*. Although *cache manager* can read database's update log, it is about which entry in the database is inserted, updated or deleted. The major challenge is for creating a mapping between the cached web pages and the changes of underlying data in the database [3]. We separate mapping into two parts: (1) Request-to-Query mapping: the mapping between web pages and queries that are used for generating these pages. (2) Query-to-Update mapping: mapping update log to query log. If database is updated, we use the update log to check

those affected queries and then invalidate the affected cache pages.

The framework of our method is shown in Fig. 2. The data flow is as follows. When *Web Server* receives a client request, if the requested page is not in *Cache Directory*, *Web Server* decides which application program should serve the request and passes it to the *Application Server*. Also *Web Server* logs the request in the access_log file. When serving this request, *Application Server* will send a query to Database to get data. When we turn on the function of Database's query log, database will log this query into the query log. So, *Logger* can read *Web Server*'s access_log and Database's update log to generate Request-to-Query mapping. *Cache Manager* reads Database's update log and mapping file to find out stale pages on the server.

2.3. Maintaining page consistency in client side

We adopt broadcasting invalidation report from server rather than sending If Modify Since from client to check the validity of cached pages. Also, the number of the web pages is not fixed. There is no fixed map table for mapping an id onto a data item. Therefore it is not possible to re-send the map table to all the clients when the map table is changed because broadcasting whole URL to indicate stale pages wastes too much energy. We reduce the invalidation report timestamp size by using the Bloom filter.

2.3.1. Managing URLs efficiently

Broadcasting complete URLs to indicate which pages are stale wastes much wireless bandwidth because the URL length is not fixed and the average URL length is longer than 40 bytes. To avoid broadcasting the complete URL of a web page to the clients, we proposed to use a fixed length encoded URL and Bloom filter [2], [8]. Bloom filter is a computationally efficient hash-based probabilistic scheme that can encode a set of strings of various lengths with minimum memory requirement. Checking the existence of a string incurs no false miss and a very small possibility of false hits.

Currently, for a URL, we select the first 64 bits from its 128-bit MD5 value as the fixed length encoded URL called *encURL*. Each client maintains a Bloom filter of 65536 bits to record which pages are stored in the client's cache. In the current design, we use four hash functions in the Bloom filter. Naturally, the 64-bit eURL is split into four 16-bit segments that are used for the four hash functions of the Bloom filter. Thus, an IR consists of a list of (eURL, TS) pairs to represent the changed URLs and their timestamps.

After receiving an IR, a client extracts four 16-bit values from each eURL and uses them to search its Bloom filter. The searching process is very efficient because only four bit positions are checked. If not all the four bits in the Bloom filter corresponding to these four extracted 16-bit values are turned on, then the client knows corresponding page is not in his cache. No further action is needed. Otherwise, the client uses the 64-bit eURL to locate the target page in the client's cache and compare TS with the timestamp of the cached page to determine the validity of the page. Locating a page using the 64-bit eURL can be efficiently performed by a hash-based implementation used in Squid proxy [9]. We do not pass a whole Bloom filter (65536 bits) to the client on every IR interval.

When a new web page is received and inserted into the client's cache, the four bits corresponding to the eURL in the client's Bloom filter must be turned on. To reduce the possibility of false hit, a simple counter associated with each bit in the Bloom filter can be employed as suggested in [10]. Also, the Bloom filter can be recomputed when the bit pressure (percentage of the set bits) of Bloom filter reaches a threshold or it can be recomputed periodically.

Using Bloom filter will result in a false hit caused by the fact that the four hash values of a eURL are contributed from the hash values of other eURLs. To find out the false hit ratio in real cases, we performed a simulation by using http traces from the web site. Our simulation parameters are in Table 1 and the simulation results are shown in Fig. 3. Since it is assumed that there are 2 to 4 updates per second on server and the IR interval is 20 seconds, the total number of updates in an IR interval is 40 to 80. We can see that even under high update rate the false hit rate is still very small. All fault hit rates in the experiments are acceptable.

To further reduce the size of broadcast IR, we propose a stateful approach that can save the bandwidth usage by removing the timestamps of some of the updated web pages in the next subsection.

2.3.2. Reducing IR's Timestamp size

Different from previous stateless approaches, we propose a stateful approach which only needs two numbers for each updated web page on server, but has a large bandwidth reduction. The proposed stateful approach tries to remove the timestamp used in a (eURL, TS) pair without violating the cache validity condition. First consider the situation shown in Fig. 4. Suppose a web page is queried by a client after it is updated at the server in the current interval. When

1550-445X/05 $20.00 © 2005 **IEEE**

receiving an IR, the timestamp of the web page maintained in client's cache must be fresher than that encoded in IR. In this case, the (eURL, TS) pair is kept unchanged. However, consider when, in current interval, no query is sent to the server after the data item is updated. For the timestamp of the web page, there is no difference between using the IR's issue timestamp and the real update time of the web page. This is because if the web page is cached in any client cache, it must be older than the timestamp encoded in IR. Therefore, for those updated web pages without query before next IR, we can use only one timestamp (the IR's timestamp).

In summary, the IR in the proposed scheme is augmented by two numbers, the number of web pages that use the old (eURL, TS) pair and the number of web pages that only use eURL. The web page with no timestamp encoded in IR uses the IR's issue time as its timestamp. Two numbers are needed for each web page on the server, independent of how many clients connecting to the server. Compared with the true stateful approach that maintains full information of which data is cached by which client, our method is scalable. The performance results will be provided in the next section.

3. Performance Evaluation

In order to analyze the performance of our invalidation algorithms, we develop a model similar to that in [10]. We assume there is only one server that serves multiple clients. The simulation model contains multiple clients, an uplink channel, a downlink channel, and a server. Clients send queries to the server via the uplink channel, and receive results from the server via the downlink channel. The database can only be updated by the server while the queries are generated on the client side. Table 2 shows the system parameters used in our simulation. The database contains D pages. The size of each web page is O bits. The size of web page ID is O_{id} bits and timestamp is T_{id} bits. 90% of web pages are in the hot update set, while 10% of web pages are in the cold update set. 90% of requests are issued for the pages in the hot update set while the remaining 10% requests are issued for the pages in the cold update set.

Effect of server's update rate

We run a simulation to find out the relation between the numbers of clients and update rates in Fig. 5. We calculate the number of updated pages that are queried after update denoted as U_{QAU} and the number of all updated pages denoted as U_{all} in the current interval. The Y-axis represents the percentile of $U_{QAU}/U_{all.}$

We can see from the figure that the number of web pages queried after update is proportional to the number of clients. When the number of client reaches 150, not shown in the figure, the percentage of web pages queried after update is 13.377% which is still acceptable. Update rates do not have much effect on our algorithm.

Power consumption

We also calculate the IR size by combining our approach with algorithms Timestamp [1] and UIR [4]. In Timestamp algorithm, the total IR size received by clients is $w\times(U_{all})\times(O_{id}+T_{id})$. By integrating our algorithm with Timestamp algorithm, we have the IR size as follows.

$$w\times U_{all}\times(O_{id})+w\times U_{QAU}\times T_{id}+w\times F_{size}.$$

F_{size} is the size of memory used to record two numbers, U_{all} and U_{QAU}, that are the number of updated web pages and the number of web pages queried after their updates in each interval, respectively.

In UIR algorithm, the total IR size received by clients is $w\times U_{all}\times(O_{id}+T_{id}) + \sum_{1}^{n-1}(P_i)\times(O_{id}+T_{id})$. By integrating our algorithm with UIR algorithm, we have the following.

$$w\times U_{all}\times O_{id} + w\times U_{QAU}\times T_{id} + w\times F_{size} + \sum_{1}^{n-1}(P_i)\times O_{id} + \sum_{1}^{n-1}(P_{iQAU})\times T_{id} + n\times F_{size}$$

IR interval is divided into n segments, so we will broadcast n−1 UIRs in each IR interval. Here,

$$P_i = \sum_{j=1}^{n-1} \text{total number of updates between } T_i \text{ and } T_{i,j}$$

P_{iQAU} is the number of updated pages queried after their updates at the time between in T_i and T_{ij}. We show that the simulated power consumption for the original Timestamp and UIR algorithms and Timestamp and UIR integrated with the proposed scheme in Fig. 6. Besides, an idealized cache invalidation scheme called *Perfect Server* is also developed for comparison. In *Perfect Server*, it is assumed that the Server has full knowledge of the contents in all clients' caches. Consequently, the invalidation reports generated by *Perfect Server* will only contain the update information of the pages in the clients' caches. As we can see in Fig. 6 that UIR and TimeStamp algorithms consume more energy than *Perfect Server*. The Timestamp and UIR algorithms integrated with our scheme consume power 40~47% less than the original counterparts. The

performance after integration is even better than Perfect Server that wastes more server load to record the individual client's caches. Power consumed by the original and modified Timestamp and UIR algorithms do not increase as the number of clients grows. But power consumed by Perfect Server increases as the number of clients increases. This is because more and more pages are cached on the client sides and so server needs to broadcast more IRs to the clients.

Effect of numbers of pages on the server

In order to know whether the number of pages stored on the server side has effects on the number of pages queried after update or not, we run simulations with different numbers of pages 100000, 150000 and 200000. The results are shown in Fig. 7. We observe that when the number of pages grows, the percentage of pages queried after update drops. As the number of pages on the server side grows, the probability that a client queries the same updated page decreases.

4. Conclusion

We proposed a framework to cache dynamic pages on the server side, proxy and client. Furthermore we use Bloom filter to efficiently encode which pages are stale. The proposed stateful approach combines the existing methods to reduce the IR size. The performance of the proposed method combined with existing Timestamp and UIR algorithms can consume power around 40~47% less than the original Timestamp and UIR Algorithms.

Reference

[1] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *Proc. of the 1994 ACM SIGMOD Conf*, pp. 1-12, 1994.

[2] B. Bloom, "Space/time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13 , pp. 422-426, Jul. 1970.

[3] K.S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven web sites. *Proc. of the SIGMOD*, 2001.

[4] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *ACM Int'l Conf. on Mobile Computing and Networking (MobiCom)*, pp. 200–209, Aug. 2000.

[5] K. L. Chiang, "Design and Implementation of Caching Dynamic Web Pages," Master thesis, Dept. of Information Management, Chung Hua University, Jul. 2003.

[6] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham and P. Shenoy, "Adaptive push-pull: disseminating dynamic web data," *Proc. of the tenth international conference on World Wide Web*, pp.265-274, May 2001.

[7] V. Duvvuri, P. Shenoy and R. Tewari "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," Tec. Re.t TR99-41, Dep. of Computer Science, University of Massachusetts at Amherst, Jun. 1999.

[8] M. Hamilton, A. Rouskov & D. Wessels. "Cache Digest Specification", http://squid.nlanr.net/Squid/CacheDigest/cache-digest-v5.txt

[9] Squid internet object cache, http://squid.nlanr.net/.

[10] K.-L. Tan, J. Cai and B. C. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol.12 , pp.789-807, Aug. 2001.

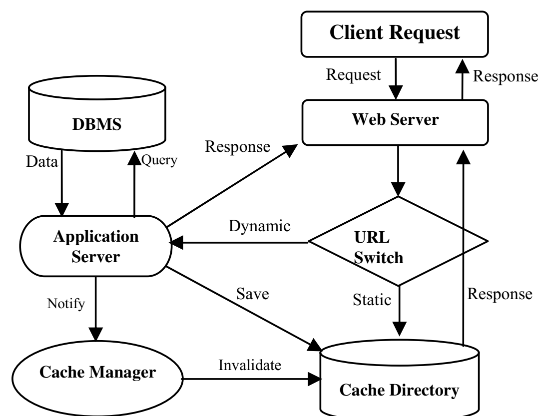

Fig. 1. The structure of dynamic web page caching system.

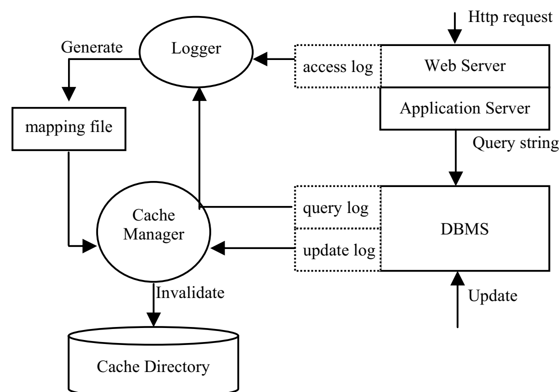

Fig. 2. Maintaining cache consistency on the server side.

1550-445X/05 $20.00 © 2005 **IEEE**

Table 1: Simulation parameters for Bloom filter.

Parameters	Default Values
Cache size	3000 pages
Mean update rate on server	2, 3 or 4 per second
% of pages in hot update set	10
IR interval	20 seconds
% of requests for pages in hot update set	90
Timestamp size	64 bits
IR window	10 intervals
Number of clients	60

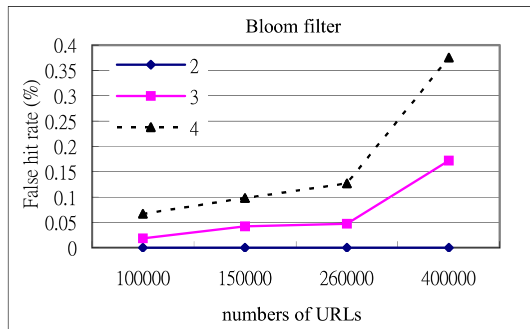

Fig. 3. False hit rate of using Bloom filters.

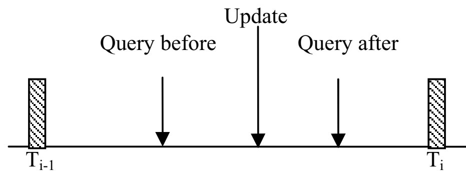

Fig. 4. The IR-based cache invalidation model.

Table 2: System parameters

Notation	Definition	Default values
D	Total web pages	100,000 pages
λ_q	Query arrival rate per client	1 query/sec
λ_u	Update arrival rate on the server	2 page/sec
β_1	% of web pages in the cold update set	10
β_2	% of requests for pages in cold update set	10
α_1	% of web pages in the hot update set	90
α_2	% of requests for pages in hot update set	90
C_{up}	Bandwidth of uplink channel	19.2 kB
C_{down}	Bandwidth of downlink channel	100 kB
L	Periodic broadcast interval	20 sec
w	IR window	10
O	Object size	5 kB
O_{id}	Object ID size	64 bits
T_{id}	Timestamp size	64 bits
C	Number of clients	20

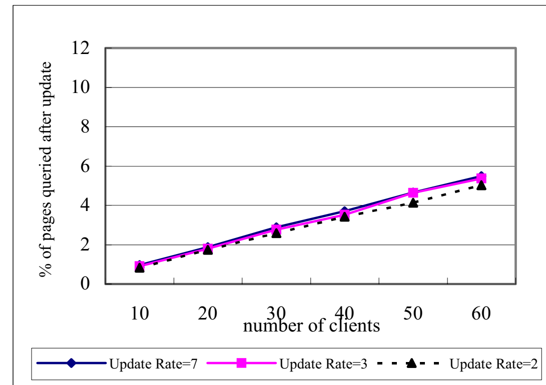

Fig. 5. Number of pages queried after update.

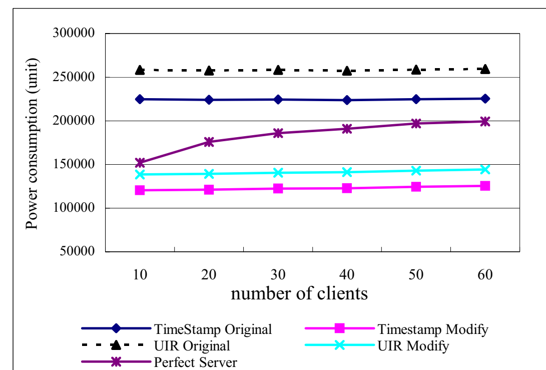

Fig. 6. Power consumption for various schemes.

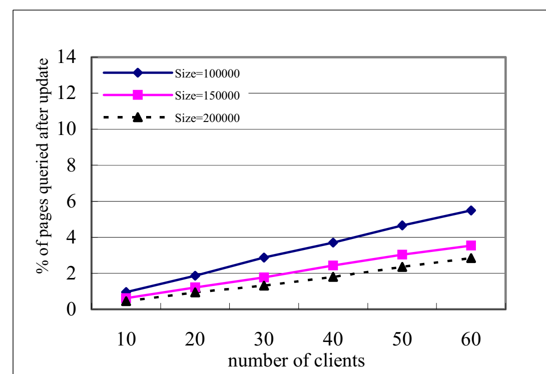

Fig. 7. The effects of number of cached pages in server on the number of pages queried after update.

1550-445X/05 $20.00 © 2005 **IEEE**