

PARALLEL CACHE MANAGEMENT PROTOCOL FOR STATIC AND DYNAMIC WEB CONTENTS^{*}

Jaeyong Lim

*Dept. of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611
jlim@cise.ufl.edu*

Sanguthevar Rajasekaran⁺

*Dept. of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269
rajasek@engr.uconn.edu*

ABSTRACT

This paper studies Random Request Distribution and Load Balancing Cache protocol (RLC) and Randomly Selectd and Limited Look at neighbor (RSL) cache algorithm based on Byte Access Frequency Factor (BAFF) for internet services with multimedia and dynamic web content. The RSL cache algorithm considers reducing response time and achieving good load balance among web servers using limitation of look at neighbor web server's cached contents and random distribution of web client requests. Our cache protocol and algorithm achieve up to 63% gain in performance for response time and 9.3 times more memory hits compared with conventional LRU and LFU algorithms. This approach has good performance gain both in current web client's request pattern such as non-uniform request pattern and in ideal web client's request pattern such as uniform request pattern. The RLC cache protocol updates cached dynamic contents completely and consistently in distributed cooperative web servers. Recently, fast response time for web client request has become more and more important in web servers. Therefore, by using this cache protocol and cache algorithm for web servers, we can build scalable web servers supporting very fast response times, high memory hit rate, and excellent load balancing of distributed cooperative web servers.

KEYWORDS

Web Caching, Web Cache Algorithm, Web Cache Protocol, and Web Server Architecture

1. INTRODUCTION

Web servers provide two types of data [4,5,11]: static data from files stored on a server and dynamic data which are constructed by programs that execute at the time a request is made. Dynamic pages are essential at websites that provide frequently changing data [4,5]. Examples include sport sites, stock market sites, and virtual stores or auction sites where information on product availability or pricing is constantly changing. There are several problems with providing dynamic data to clients efficiently and consistently. A key problem with dynamic data is that dynamic pages can seriously reduce web servers performance. The performance bottleneck is often the CPU overhead associated with generating dynamic pages. Another problem is determining what pages should be cached and when a cached page has become obsolete. Thus, to improve the performance of web server, a key requirement for many websites providing static and dynamic data is to completely and consistently update pages which have changed.

Numerous papers [1,2,3,6,7,8,17,19,21,22] discuss caching of web content; however, they focus on caching static data and intentionally avoid caching dynamic data. Furthermore, they concentrate on proxy caching rather than server-side caching. Proxy caching of dynamic content may not be feasible because it does not permit caching of authenticated content. Proxy caching has also other problems such as an obsolete

^{*} This work was supported in part by the National Science Foundation under grant CCR-9912395.

⁺ This work was conducted while the author was at the University of Florida.

of cached data, long response time for web client request while proxy server searches requested data in multi-level hierarchical structure, and increase of network bandwidth for message communication among distributed proxy servers.

Currently the best ways of web cache protocol and algorithm design are to analyze web client's request pattern. Although many researchers [4,5,9,10,11,12,17,19,21] have tried to find a regular model to fit a web client's request pattern by analyzing some real log traces in several web servers, there are no satisfactory results in this area because there are so many different characteristics among log traces in web servers. Some general agreements in web client's request pattern are that hot documents are received higher requests than others and most of a web client's requests are concentrated in ~10 - 20% of total documents in web servers. From these results we propose a new caching protocol RLC (Random RequDistribution and Load Balancing Caching Protocol) and a new cache algorithm RSLL (Randomly SelectEd and Limited Look at neighbor). This protocol and algorithm achieves excellent cache hit rate, response time, load balancing, and memory utilization compared to LRU and LFU algorithms. The RLC protocol caches static and dynamic data cooperatively and allows duplication of a hot content among web servers. The RSLL cache algorithm is based on Byte Access Frequency Factor (BAFF) [14]. The RSLL also has a special feature treating hot documents in a web server. RSLL cache algorithm is not only suitable for current web client's request pattern such as non-uniform discrete random variation but also works well in ideal web client's request pattern such as uniform discrete random variation.

There are many web caching schemes proposed in the literature [1,2,3,4,5,6,7,8,12,17,19,21]. Due to space limitation, the overview of the approaches is omitted here. Please refer to the individual work for the description of the schemes. The layout of the paper is described as follows: in Section 2 we discuss our scalable web server architecture; in Section 3 we look at the RSLL cache algorithm; in Section 4 we discuss the distributed cache content management protocol; in Section 5 we explain our simulation environment and analytical results of our cache protocol and cache algorithm. Finally, we conclude the paper in Section 6.

2. DESIGN OF SCALABLE DYNAMIC WEB SERVER ARCHITECTURE

To support the growing number of web client's static and dynamic data requests, scalable WWW architecture that consists of a group of loosely coupled WWW servers is needed. We propose a multi-node web server architecture that consists of a Central Server, multiple web servers, and the Distributor as shown in Figure 1. The dynamic web server architecture can be easily scaled up by installing a new web server in the high-speed network.

The Central Server stores a complete set of static and dynamic web documents and performs all documents maintenance jobs. The Central Server also stores all information about static and dynamic documents using the hash table in the document table, including their BF (Byte access Frequency factor of document) [14] value, document identification number, version of the document, duplication of the documents, size of document, etc.

The web server implementation is based on a novel peer-to-peer process structure that enables one web server to access memory or persistent data files at other web servers. If a requested document is missing at the randomly selected web server's memory where the request originates, then the web server looks at the limited number of other web servers' memory to serve the requested data to a web client. The more details on functionality of web server will be discussed later.

Each web server contains a Cache Maintenance Process and a Web Server Process that talks to other web servers and serves a web client's request by passing it through a communication network such as the internet or intranet. A Cache Maintenance Process of each web server updates cached static and dynamic data consistently and completely when cached data has changed. Each web server's memory is normally managed locally at each web server. High BF value documents are kept in the local cache and the Cache Maintenance Process maintains them. Each web server also contains limited number of neighbor web servers' cache content information in its memory. This approach helps to reduce the response time, load of the Central Server, and the network bandwidth for serving the web client's request. It also increases the cache-hit rate for cached data among web servers. The detailed operations of web server using the RLC caching protocol and RSLL cache algorithm will be discussed in later sections.

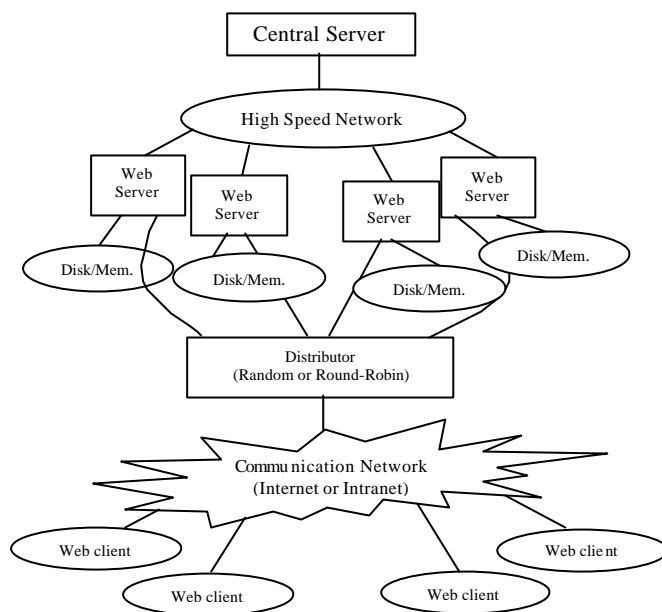


Figure 1. A Scalable Web Server Architecture

access logs to re-calculate the BF value of each document. Based on the new information, each web server will change its cache contents.

3. RSLC CACHE ALGORITHM

From the basis of the BAFF concept [14], we developed a new web caching algorithm; RSLC. When a web client's request enters through the network, the Distributor selects a web server randomly for distributing a web client's request. The selected web server first checks that it keeps the requested document in cache or not using hash table. If the document is kept in the cache, the server sends the requested document to the web client and updates the BF value of that document. If the server does not keep the document in its cache, it looks for the document in the nearest neighbor web servers' cached information which is stored in its memory. If the requested document is found in the nearest neighbor server's cache, it is forwarded to the requesting web server. The applicable web server serves the request to the web client and updates the information for the requested document. If the server does not find the requested document in its cached information, which includes its cache contents and its nearest web server's cache contents, it looks at the limited number of its next nearest neighbor web servers' cached information to find the requested document. If the server finds the requested document in its next nearest neighbor web server's cache, the server forwards the request to the applicable web server and the selected applicable web server serves the web client's request and updates some values. If the server finds the request in its next nearest neighbor web server's neighbor cache, the server forwards the request to applicable web server and the selected applicable web server serves the web client's request and updates the BF value. If after making these attempts the document can't be found, the requested document is requested from the Central Server.

Randomly Selected and Limited Look at neighbor Cache Algorithm (RSLC)

Step 1. Randomly select a web server by Distributor

Step 2. If the requested document is found in the selected web server's cached information,

If the requested document exists in selected web server's memory

Service the web client request,

Increase the value of number of request for that document,

Update the BF value of that document, and quit go to Step 1;

Else if the requested document exists in selected web server's the nearest neighbor's web server's memory

Incoming web client requests go through the Distributor which distributes the request randomly through among web servers. The intent of using the Distributor is to balance a load among web servers and to decrease response time for a web client's request. On our previous work [14] shows the functionality of the Distributor more detail.

Static and dynamic web contents that have a high BF value will be copied to and cached in the web servers. When web servers have cached a high BF value for both static and dynamic web contents they can serve most of web client's requests immediately without accessing the Central Server. Therefore, the Central Server will not be the performance bottleneck for serving static and dynamic web contents in the system.

All Web Server Processes in the web servers keep access logs, and at the end of each predefined period all access logs are sent to the Central Server for updating the document information. The Central Server uses these

```

Forward the request to applicable web server
The applicable web server serves the web client request,
Increase the value of number of request for that document,
Update the BF value of that document, and quit go to Step 1;
Else repeat
  Look at the neighbor web servers' cached information to find the request document
until reaching the limited number of nearest neighbors
  If found in the next nearest neighbor web servers' cached information,
    If the requested document exists in the next nearest neighbor web servers' memory
      Forward the request to applicable web server
      The applicable web server serves the web client request,
      Increase the value of number of request for that document,
      Update the BF value of that document, and quit go to Step 1;
    Else if the requested document exists in the next nearest neighbor web server's neighbor's cache
      Forward the request to applicable web server
      The applicable web server serves the web client request,
      Increase the value of number of request for that document,
      Update the BF value of that document, and quit go to Step 1;
    Else if requests the document to the Central Server,
      If receives the document from the Central Server,
        Service the web client request,
        Increase the value of number of request for that document,
        Update the BF value of that document,
        Compare the BF's value of the requested document with that of existing
        document in selected web server's memory based on the size of new document,
        If the BF value of the requested document is greater than the existing document,
          Replace the existing document with the requested document,
          Send this information to the Central Server and applicable neighbor
          web servers, and quit. go to Step 1.
        Else stores the requested document to the selected web server's disk,
          and quit. go to Step 1.
      Else sends a message "document not found" to the web client,
        and quit. go to Step 1.

```

4. RLC CACHE PROTOCOL

4.1 Central Server Mechanism

Communication between the Central Server and web servers is done using a broadcast mechanism. There are many reasons why a broadcast mechanism is used. First, the web server architecture can be easily scaled up by installing a web server to or removing it from a high-speed network. Second, establishment of a network connection and network overhead for communicating between the Central Server and web servers are minimal. Third, the Central Server can easily optimize its I/O for transmitting static and dynamic web documents. Finally, static and dynamic data consistency between the Central Server and web servers can be maintained easily, and completely. This can avoid stale information being accessed by web clients.

A more complicated cache mechanism must be employed when data are processed from many web servers in order to maintain cached static and dynamic data efficiently and consistently. A document table kept in the Central Server will be used for this caching mechanism. A document table keeps the Document Identification Number, Version of document, Number of request, Size of document, BF value of document, Duplicate of document, and Server ID of duplicated document information for maintaining all documents:

Our caching protocol allows duplication of document among web servers. Especially for dynamic data, the document generation time is much longer than that of static data. If several dynamic web contents are hot documents and cached on only one web server, then the cached web server suffers from a bottleneck of processing the dynamic data and service time to web clients longer than that for cached several web servers. If there are too many duplicated documents among web servers, then the overhead of redundant documents management is significant and efficiency of cache utilization of among web servers is decreased. Thus, we only allow the duplication of document among web servers at most 30% of total number of web servers, where the duplicated documents are cached.

After a predefined period, a new document table must be generated to reflect any change in documents. Starting with the highest BF value document, the Central Server will assign new document to web servers using a round-robin pattern until each web server fills out its memory. This approach achieves good load balance among web servers and faster response time to a web client, because hot document requests are

evenly distributed among web servers. The Central Server broadcasts static and dynamic web documents to web servers. There are six different types of broadcast messages. Each broadcast message is encapsulated within the header information, for example, type of broadcast and identification of web server the most up-to-date data can be accessed by the web client. The detail operation will be found in [16].

4.2 Web Server Mechanism

We design web servers which operate cooperatively with each other to serve a web client requests. Each web server has two functional processes called the Cache Maintenance Process and the Web Server Process. Each web server keeps its cached contents information in its memory. It also keeps the Selected Neighbor Web Servers' cached contents information in its memory. The Selected Neighbor Web Servers are the nearest left and right web servers. This approach achieves faster retrieval of the requested data and maintains the cached information easily. When each web server keeps so many other web servers' cached information in its memory, it consumes more memory to keep this information and it becomes difficult to cache real data because of fewer memory spaces. It is also difficult to maintain the cached information because of increased cached neighbor web servers' information. This approach also increases the communication overhead among web servers because increased number of broadcast messages are requested for making cache requests. Figure 2 explains the mechanism of the Web Server Process in web server.

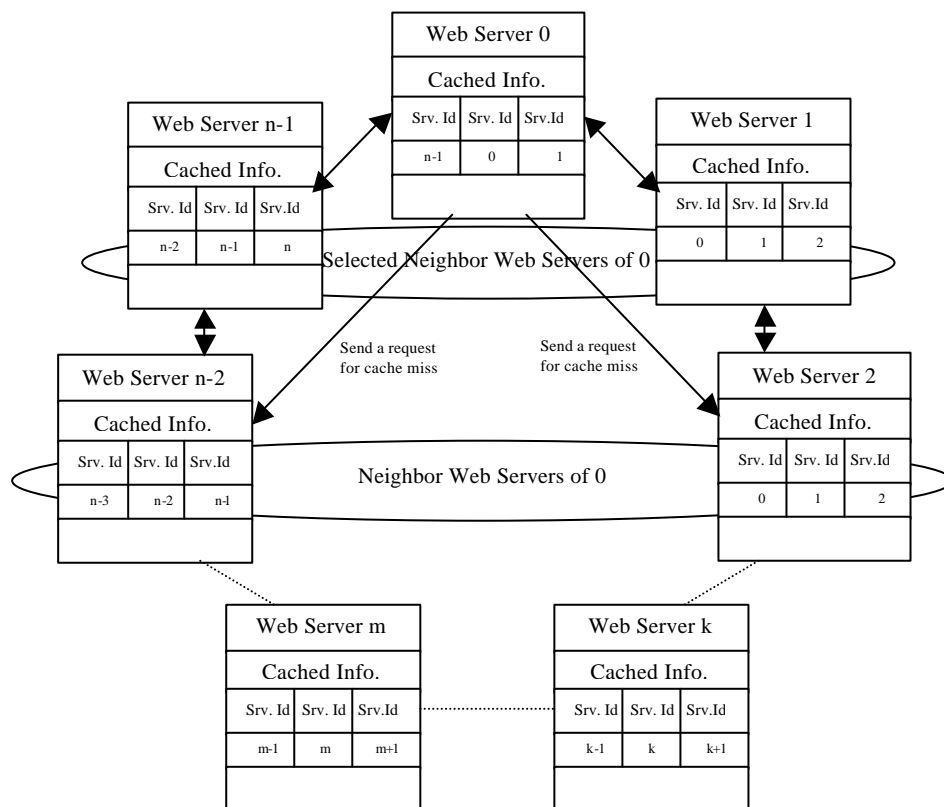


Figure 2. The mechanism of the Web Server Process in Web server

The total number of web servers (N) depends on each web servers' configuration. For example, we choose N=9. When a web client request arrives at web server 0 and a cache miss happens, then it looks at its Neighbor Web Servers web server 2 and web server 7 to find the requested document. With this procedure, web server 0 looks at only 7 of the 9 web servers' cached information such as web server 0, 1, 2, 3, 6, 7, 8. It covers 78% of the total web servers' cached information. This approach achieves an excellent cache hit rate

compared to fully storing all of the other web servers' cached information in each web server. This approach also reduces the Central Server's load, which occurs only in the case of a cache miss among web servers. This approach has several advantages compared with fully storing all other web servers' cached information in each web server such as less memory consumption, easy maintenance, and reduction of communication overhead among web servers.

The Cache Maintenance Process performs two main functions: 1) it maintains the cache contents of the local cache; 2) it performs the corresponding actions when it receives a broadcast message from the Central Server. When a web server receives a broadcast message from the Central Server, the Cache Maintenance Process buffers the message, and examines the identification of the web server field and the type of broadcast message from the header information. If the message is sent to it, and it takes the message and performs the appropriate procedure. The detail operation will be found in [16].

5. SIMULATION ENVIRONMENT AND RESULTS

We simulated our new caching protocol and algorithm to measure the performance gain compared with LRU and LFU algorithms. For simulation, we developed a web client access pattern generator instead of using real data access logs in specific web servers. The reason for using our own web client access pattern generator is that specific web servers' logs cannot stand as a general guide for web clients' request pattern and the performance gain of the cache algorithm depends on a specific web server's log. This access pattern generator is designed to follow current web clients' request patterns based on previous research [5,11,12,13,18,19,21]. We used Algorithm Generate 1 [20] to generate this current web clients' request pattern in the access pattern generator. In Algorithm Generate 1, we inserted the concept of High BF value factor to generate the non-uniform discrete random variation of web client requests.

We chose nine web servers. Each web server is connected via a high-speed network. To test for our caching protocol and cache algorithm, we made initial document information. Initial documents are composed of static and dynamic data with no more than 40% of the total documents containing dynamic data. To verify the performance of our caching protocol and cache algorithm, we simulated several cases. Each case changed the value of several factors such as the portion of dynamic data in total documents and the memory size of each server. The total memory size of the web servers is 10% ~ 20% of the total document size. To analyze the performance gain of our caching protocol and algorithm, we used two values such as the response time for total web client requests and cache hit rate. We assumed each web server ran concurrently and cooperatively to service a web client's request. Each web server's memory contents were updated by the RLC cache protocol. The redundancy of web contents among web servers was also controlled by RLC cache protocol. In graphs, RSL10 means that there are 10% dynamic data in total documents.

5.1 Performance Results of Variation of Dynamic Data

In the case of different memory sizes, we compare the conventional LRU and LFU algorithms with the RSL algorithm. The performance gain for total response time for RSL is increased by up to 63% in 512 Mbytes memory size compared with LRU in non-uniform case. By increasing the memory size, performance with RSL increased by up to 63% in total response time at 512 Mbytes compared with 64Mbytes. However, conventional LRU increased by up to 9% for performance and LFU increased up to 12%. For both non-uniform and uniform cases, the results were the same. Figure 3 shows the result of RSL algorithm's performance gain for total response time in case of different memory size.

Figure 4 shows the results of RSL algorithm's performance gain for memory hit rate in case of different memory size. In non-uniform, RSL had 9.1 times more memory hit rate in 64 Mbytes compared with LRU algorithm. When increasing the memory size, the gain was decreased in 6.9 times more in 512 Mbytes compared with LRU. It is a reasonable result because the memory size is increased. RSL has 6.3 times more memory hit according to increasing the memory size up to 512 Mbytes compared with 64 Mbytes in RSL10 case. LRU earned 7.8 times more memory hit in 512 Mbytes compared with 64 Mbytes. This means that LRU is more sensitive in memory hits than RSL. In uniform case, RSL has more memory hits compared with non-uniform. RSL earned up to 9.3 times more memory hits in 64Mbytes memory than the LRU algorithm. This result is the same in every different portion of dynamic data cases. In the rest of cases such as

128 Mbytes, 256 Mbytes, and 512 Mbytes the same performance gain was seen as for non-uniform. With an increase in memory size, RSSL had 6.5 times more memory hits in 512 Mbytes compared with 64 Mbytes in the RSSL10 case. LRU has eight times more memory hits in 512 Mbytes than compared with 64 Mbytes. It means that RSSL has more memory hit in small memory size and uniform web client requests.

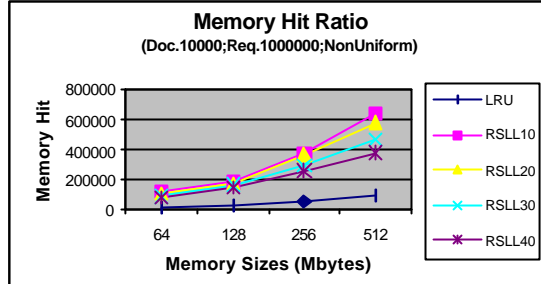
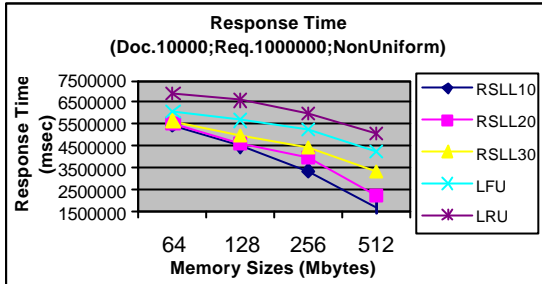


Figure 3. Total Response Time for Non-Uniform Case Figure 4. Total Memory Hit Rate for Non-Uniform Case

5.2 Performance Comparison between RSSL and DSLL

Now we show the performance comparison between RSSL and DSLL (Directly Selected and Limited Look up) cache algorithms. The DSLL was proposed by our other research work [15]. The function of DSLL is that DSLL has the Request Distributing and Load Balancing Manager and this manager distributes every web client's requests to a web server directly, which caches the requested document in its memory.

In the non-uniform case, the performance of total response time for RSSL algorithm was decreased by up to 5% with 64 Mbytes memory compared with the DSLL algorithm. In the cases of 128 Mbytes, 256 Mbytes, and 512 Mbytes, the performance loss of total response time is 4.5%, 3%, and 2.2%, respectively, compared with DSLL10 case. These results are the same in different portions of dynamic data. Figure 5 shows the results of RSSL and DSLL algorithms' performance difference in total response time at non-uniform case.

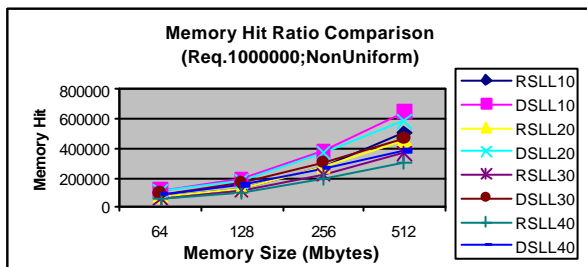
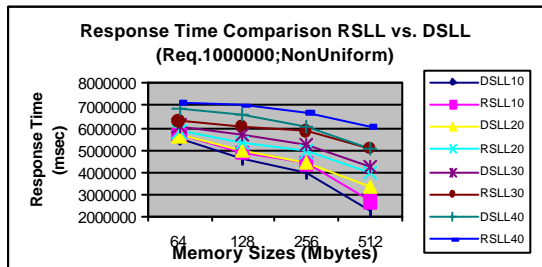


Figure 5. Total Response Time for Non-Uniform Case Figure 6. Total Memory Hit for Non-Uniform Case

We now compare the memory-hit rate between RSSL and DSLL algorithms. In the non-uniform case, the performance of memory-hit rate for RSSL algorithm was decreased by up to 5.6% in 64 Mbytes memory in the DSLL10 case. In the cases of 128 Mbytes, 256 Mbytes, and 512 Mbytes, performance loss was 4.8%, 3.2%, and 2.3%, respectively, compared with DSLL10 case. These results are the same in different portions of dynamic data. Figure 6 shows the results for RSSL and DSLL algorithms' performance difference in memory-hit rate for the non-uniform case.

Although the DSLL has a greater performance gain for response time and increased memory hit rate by up to 5.7% compared with RSSL, the RSSL has several advantages compared with the DSLL and these benefits cover the performance gap. First, the RSSL approach is more reliable than the DSLL approach. When the distribution manager in DSLL approach crashes the DSLL method can't serve the web client request anymore. However in RSSL, if any one of web servers are crashed, the RSSL method still serves the web client request. Second, the RSSL approach can expand its system power easily by adding a new web server into the network compared with the DSLL. Third, we can use the distribution manager in the DSLL

approach as a web server when we choose to use the RSLI approach. It reduces the configuration cost and increases the processing power. Finally, the RSLI approach reduces the communication overhead among web servers since it does not need to communicate between web servers and the distribution manager which are needed in the DSLI approach. The RSLI approach causes a reduction in network bandwidth usage in the network.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose the RLI cache protocol and the RSLI cache algorithm based on the Byte Access Frequency Factor concept. They have an intelligent cache algorithm, in-memory cache maintenance table, as well as a consistent and complete cache maintenance mechanism. This approach does not need extra applications for managing static and dynamic web contents. This approach also achieves less communication cost among web servers using broadcast mechanisms. Thus, by using this cache protocol and cache algorithm for web servers, we can build scalable web servers supporting very fast response times, high memory hit rates, and excellent load balancing of cooperative distributed web servers. In the next step, we hope to expand these concepts to a geographically distributed scalable web server environment. These systems must have more functionality like choosing the best network route to server the web client. These systems also combine the proxy servers.

REFERENCES

- [1] Takuya Asaka, Hiroyoshi Miwa, and Yoshiaki Tanaka, 'Distributed Web Caching using Hash-based Query Caching Method', IEEE International Conference on Control Applications, 1999
- [2] Martin F. Arlitt, and Carey L. Williamson, 'Internet Web Servers: Workload Characterization and Performance Implications', IEEE/ACM Transaction on Networking, 1997
- [3] Lee Breslau, Pei Cao, Li Fan, Graham Philips, and Scott Shenker, 'Web Caching and Zipf-like Distributions: Evidence and Implications', IEEE INFOCOM, 1999
- [4] Jim Challenger, Arun Iyengar, and Paul Dantzig, 'A Scalable System for Consistently Caching Dynamic Web Data', IEEE INFOCOM 1999
- [5] Jim Challenger, Arun Iyengar, and Karen Witting, 'A Publishing System for Efficiently Creating Dynamic Web Content', IEEE INFOCOM 2000
- [6] Pei Cao, and Sandy Irani, 'Cost-Aware WWW Proxy Caching Algorithms', Proceedings of the USENIX Symposium on Internet Technologies and Systems, December, 1997
- [7] Kai Cheng, and Yahiko Kambayashi, 'Multicache-based Content Management for Web Caching', International Conference on Web Information Systems Engineering (WISE), June, 2000
- [8] Cho-Yu Chiang, Yingjie Li, Ming T. Liu, and Mervin E. Muller, 'On Request Forwarding for Dynamic Web Caching Hierarchies', The 20th International Conference on Distributed Computing Systems, 2000
- [9] Brian D. Davison, 'A Web Caching Primer', IEEE Internet Computing August, 2001
- [10] Sandra G. Dykes, Clinton L. Jeffery, and Samir Das, 'Taxonomy and Design Analysis for Distributed Web Caching', Proceedings of the 32nd Hawaii International Conference on System Sciences, 1999
- [11] Vegard Holmedahl, Ben Smith, and Tao Yang, 'Cooperative Caching of Dynamic Content on a Distributed Web Server', Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, 1998
- [12] Arun Iyengar, Jim Challenger, Daniel Dias, and Paul Dantzig, 'High-Performance Web Site Design Techniques', IEEE Internet Computing, 2000
- [13] Thomas T. Kwan, Robert E. McGrath and Daniel A. Reed, 'NCSA' s World Wide Web Server: Design and Performance', IEEE Transactions on Computers, September, 1995
- [14] Jaeyong Lim and Sanguthevar Rajasekaran, 'Byte Access Frequency Factor Based Web Caching Algorithms for Scalable Web Servers', Technical Report, University of Florida, 2000
- [15] Jaeyong Lim and Sanguthevar Rajasekaran, 'Directly Selected and Limited Look up Cache Algorithm for Dynamic Web Contents', 7th International Conference on Networks, Parallel and Distributed Processing, and Applications, 2002

- [16] Jaeyong Lim and Sanguthevar Rajasekaran, 'Cache Algorithm and Protocol for Static and Dynamic Web Contents', Submitted in Journal of Parallel and Distributed Systems, 2002
- [17] Jean-Marc Menaud, Valerie Issarny, and Michel Banatre, 'A Scalable and Efficient Cooperative System for Web Caches', IEEE Concurrency 2000 September, Vol.8 No.3, 2000
- [18] Igor Tatarinov, Alex Rousskov, and Valery Soloviev, 'Static Caching in Web Servers', IEEE International Conference on Computer Communications and Networks, 1997
- [19] Renu Tewari, Michael Dahlin, Harrick M. Vin, and Jonathan S. Kay, 'Design Considerations for Distributed Caching on the Internet', IEEE International Conference on Distributed Computing Systems, 1998
- [20] Sanguthevar Rajasekaran, Keith W. Ross, 'Fast Algorithms for Generating Discrete Random Variates with Changing Distributions', ACM Transaction on Modeling and Computer Simulation, Volume 3, 1993
- [21] Amin Vahdat, and Thomas Anderson, 'Transparent Result Caching', Proceedings of the 1998 USENIX Technical Conference, 1998
- [22] Junbiao Zhang, Rauf Izmailov, Daniel Reininger, and Maximilian Ott, 'Web caching framework: Analytical models and beyond', IEEE Workshop on Internet Applications, 1999