A PROactive Request Distribution (PRORD) Using Web Log Mining in Cluster-Based Web Server

Heung Ki Lee[1], Gopinath Vageesan[1], Ki Hwan Yum[2] and Eun Jung Kim[1]
[1]Texas A&M University [2]University of Texas at San Antonio

Hklee@cs.tamu.edu vgopi@ee.tamu.edu ejkim@cs.tamu.edu yum@cs.utsa.edu

ABSTRACT

Widely adopted, distributor-based systems forward user requests to a balanced set of waiting servers in complete transparency to the users. The policy employed in forwarding requests from the front-end distributor to the backend servers plays an important role in the overall system performance. The locality-aware request distribution (LARD) scheme improves the system response time by having the requests serviced by the web servers that contain the data in their cache. In this paper, we propose a proactive request distribution (PRORD) to apply an intelligent proactive-distribution at the front-end and complementary pre-fetching at the back-end server nodes to acquire the data into their caches. The pre-fetching scheme fetches the web pages in advance into the memory based on a confidence value of the web page, which is predicted by the proactive distribution scheme. The proactive distribution depends on both online and offline analysis of the website log files, which capture user navigation patterns on the website. Designed to work with the prevailing web technologies, such as HTTP 1.1, our scheme aims to provide reduced response time to the users. Simulations carried out with traces derived from the log files of real web servers witness performance boost of 15-45% compared to the existing distribution policies

1. INTRODUCTION

Cluster systems are being increasingly used in the web-server management, file distribution and database transactions. The main reason for the large-scale deployment of the cluster systems is their load sharing and high-performance capabilities. The overall delay incurred by the end-user is the sum of network-link delay, routing delay, delay accrued during address resolution and finally the web-server service delay. It has been observed that web servers contribute to approximately 40% of the overall delay [1], and this delay is likely to grow with the increasing use of dynamic contents. The delay incurred at a web server consists of the processing time and data retrieval time. Cluster-based web servers incur an additional delay in analyzing the incoming request and forwarding the request to one of the back-end servers. Thus, the delay at the web server is a critical component which has to be reduced to achieve a better web-server performance.

Among the different architectures in the cluster-based servers, the distributor-based systems have been widely deployed as shown in Fig. 1. These systems have a front-end switch which forwards the requests to any of the backend/distributor. In locality-based request distribution schemes, the distributor contacts the dispatcher to obtain the locality information. If the page is located in the same backend server, the request is serviced directly. Otherwise, the distributor forwards the request to the backend server that has a better locality for the file. The role of the dispatcher is to specify the locality of the requested files to the distributor. The forwarding of the requests from the distributor to the backend servers is carried out in complete transparency to the users. A handoff protocol and TCP splicing are employed in most cases to make the transition smooth and transparent [13, 14, 15]. The requests are forwarded to the set of backend servers based on a certain policy. LARD (Locality Aware Request Distribution) [2], PARD (Power Aware Request Distribution) [3] and WRR (Weighted Round Robin) are few of the most prolifically adopted policies. The policies focus on improving efficiency, power conservation and load balancing respectively.

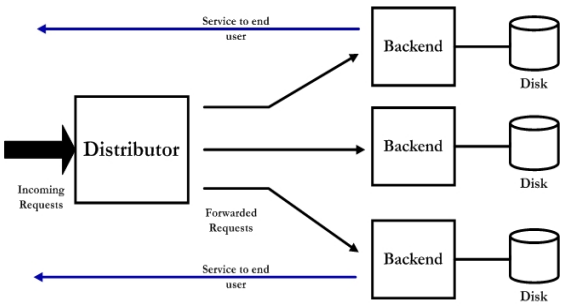

Fig. 1: Distributor Based Web Server System

In this paper, we propose a new proactive request distribution scheme (PRORD), which reduces the delay at the web server by bundling requests. We improve the

distribution policy at the front-end by dispatching the requests based on the analysis of the web servers' log files. The general user navigation pattern, user behavior and general website organization are some of the critical information that is extracted from the web server log files. This information is made available for the distributor at the front-end to discern and classify the incoming requests and perform the dispatch to the appropriate backend server. This information is also used to pre-fetch the data files into the web servers' cache [20]. Pre-fetching is thus complementary to the dispatches being made by the distributor at the front-end.

In Section 2, we discuss the existing technologies in detail. Section 3 describes the applications of web log mining in context of our research. Section 4 explains the usage of web log mining for enhancing the distribution policy at the front-end. While Section 5 shows the simulation model and the results, Section 6 concludes the paper.

2. RELATED WORK

The volume of published research in the area of cluster-based web servers and cluster-based application servers [2, 3, 4, 5] testifies to the interest of the numerous researchers. Among the distributor-based cluster systems, weighted round robin is considered to be simple and efficient scheme for providing excellent load balancing of the requests arriving to the system. However, such systems do not contribute to the performance of the system. Locality-based request distribution schemes focus on improving the performance through intelligent distribution of the incoming requests [2, 4, 5]. In this paper, we restrict our study to LARD [2], Ext-LARD-PHTTP [5] and web log mining policies, and compare our scheme against them.

2.1 Locality-Aware Request Distribution

The LARD [2] overcomes the drawbacks faced by the WRR policy in terms of performance. It uses locality-based distribution policy at the distributor and increases the memory hits at the backend server. It considers both data locality and load balancing issues in a distributed cluster-based server. The distributor in LARD forwards all requests for the same web object to a server node that has the requested file in its cache (memory). If the load on that node is high, then the request is forwarded to another lightly loaded node that has the contents on its disk. In an improvement to this idea, Aron, et al. [4] proposed a scalable content-aware distribution policy that minimizes frequency for accessing the backend server by using a de-centralized request distribution strategy. In this policy, a layer-4 web-switch is used to forward the incoming requests into one of distributors on the backend server. However, this architecture suffers from a single point of failure. Also, the overhead to dispatch all the requests can be very high. In addition to these drawbacks, both the above contributions limit their study to HTTP 0.9/1.0 based web transactions. In HTTP 1.1 based web transactions, the persistent connection suffer from inefficient distribution of the incoming requests among back-end servers. The front-end in a cluster system with HTTP 0.9/1.0 handles every incoming request and distributes it to the backend servers. However, a cluster system operating on HTTP 1.1 should handle the multiple requests from a single client in one single connection.

2.1.1 Persistent HTTP

With HTTP 1.1, the user can request multiple pages to the server on the same persistent connection. Two schemes have been proposed to address the problem of persistent HTTP: multiple TCP handoffs and back-end forwarding scheme. Multiple TCP handoffs [5] analyze and dispatch whole incoming requests at the front-end. The LARD policy is applied to each incoming request, requiring TCP handoffs for each request, even though the requests are from the same user. In back-end forwarding scheme [5], the front-end initiates a single handoff for every persistent HTTP connection. The back-end servers are connected over a high speed network and the request can be internally forwarded and served among the backend server nodes.

Both the above techniques suffer from high overhead. The primary goal of this paper is to provide a low overhead content-based request distribution at the distributor, while maintaining the QoS.

2.2 Website Log Mining

Web log mining has been prolifically used in web services [7, 8, 9, 10, and 11]; however, none of them consider the possibility of using the information from web log mining for improving the distribution policy in a cluster-based web server. The server logs can be analyzed for user browsing pattern, general website organization and other website statistics and can be used to improve the QoS of the website. The following sections describe the research that has been done in this context.

2.2.1 User Navigation Pattern

The user's navigation pattern is a rich source to understand the general user behavior on a website. This information can be easily gleaned from the web server log files. It can be used to categorize the users based on their interests and also to predict their intended navigation pattern. In most of the large websites, the users' target document does not exist in the users' expected location.

In [6], this information is used to improve the website organization by providing hyperlinks to users' target document on the users' expected location of the webpage. Nakayama, et al. [10] have used the web log files to discover the gap between website users' behavior and the website designers' expectations. They evaluate these

metrics using inter-page access co-occurrence and inter-page conceptual relevance respectively. Perkowitz, et al. [8, 9] have developed a clustering algorithm to identify web pages that occur together in single user visits and build an index page, which helps the users to effectively navigate the website. Spiliopoulou, et al. [11, 12] have proposed a web mining tool (WUM-Web Utilization Miner) for analyzing the log files. The tool analyzes the structure of the traversed paths of the website users to extract sub-paths which lead to a target item of interest.

The navigation pattern of the users can help to re-organize the website such that, the required target data is readily available to the users.

2.2.2 Bundling Requests

In [7], they show that pre-fetching of the embedded objects associated with a particular page can provide considerable performance boost. The webpage and its associated embedded objects such as images, applets, etc. are grouped into a "bundle" and delivered to the user browser in a compressed form on the request of the web page. In [29], Cohen proposes an improved method for updating the cached webpage at the proxy servers. During the update through bundle information, update period for the stale webpage is enlarged due to the slight increase of update overhead. Log mining information is used for creating the update information based on the relationship between web-pages and thus minimizing the update overhead.

2.2.3. Web Use Mining and Prediction

In web usage mining, it is critical how to analyze the log file in the system. For improvement for web search, two rules are suggested; association rule [23, 24] and sequence rule [25]. The association rule uses set-based operations for analyzing the log file, while the system analyzes the sequence of the web log file in the sequence rule. Kitsuregawa [17] improved the scheme based on association rule [23, 24] and sequence rule [25] and implemented an Mobile Information Search (MIS) system through PC cluster. In [20], the authors present another method for pre-fetching through web log mining. Embedded Object Table (EOT) and set of association rules are constructed for pre-fetching and caching the page. In addition, they extend GDSF [30] through splitting frequency into future frequency and past frequency through an association rule. In [21], the authors compare three web mining approaches: association rules [23, 24], sequence rules [25, 27] and generalized sequence rules [28]. They prove that sequence rules outperform the other approaches.

Data structure for pre-fetching using web log mining can be divided into two schemes roughly: Dependency graph (DG) [19] and PPM (Prediction-by-Partial-Match) [26]. In [19], Padmanabhan, et al. proposes a scheme for pre-fetching through website log mining. The scheme uses a prediction engine which keeps track of web page relationship information. Weighted direct graph are constructed where nodes constitute the web pages and the arcs represent the relationship between the web pages. In PPM[26], j-order Markov predictor is maintained for the prediction in the comparison of the previous j accessed pages. Even though the predictor gives the accurate result for higher orders, overhead for maintenance of predictor is increased and forms the bottleneck of the scheme. In [18], the authors propose a scheme for pre-fetching webpage through PPM and the DG. In [17], MIS is described for collecting and clustering the web pages.

Though the above researches testify the volume of the work that has been carried out in web log mining, its usage in improving the distribution policy in cluster-based web servers has not been explored. Also, the information extracted from log files by our algorithms and their usage to our system is unique. The following sections describe our idea in detail.

3. Harnessing Web Log Files

We employ the web log files to collect a host of information; the users' navigation pattern, the popularity of the web pages and spotting 'bundles' of pages. This information can be directly used for discerning the incoming requests and dispatching them to the appropriate backend server nodes. Each of these information segments and their uses are elaborated in the following sub sections.

3.1 Users' Navigation Pattern

We use the script to analyze the log files and garner the access pattern of the users on a website. Every website can be categorically sub-divided based on the different category of web users visiting the website. For example, a university website will most likely cater to the needs of current students, prospective students, faculty members, support staff people and other users. Thus, the users on such a website can be categorized into such well known groups. Each of these groups' users has a highly directional and mostly unique access pattern. Thus, this information can be used to categorize the users visiting the website into pre-defined groups. The information about the user's group can be insightful in predicting the possible data that would be requested by the user in the near future.

3.2 Popularity and Spotting Bundles for Web Pages

We also identify and rank the web pages based on their popularity and demand. The number of requests to a particular page can be easily read off the log files and this can be used to rank the web pages. We employ a two-fold system to rank the web pages; we have offline analysis of the log files and also dynamic online tracking of the page

hits to obtain a realistic estimate of the popularity of the web pages. As in [7], the web page and its associated embedded objects can be identified from the log files. Image files, applets, audio/video streams, etc. constitute a "bundle" for the respective web page. These objects are bound to be requested by the user's browser in the subsequent requests. Though spotting the bundles is similar to the method outlined in [7], the application of bundles differs in our system and is explained in detail in Section 4.

4. Proactive Request Distribution Scheme (PRORD) using Web Log Information

The primary purpose of the web log mining is to enhance the distribution policy at the front-end. We use the web log mining information to make the distribution proactive and provide effective pre-fetching at the backend server nodes. In this section, we present a new proactive request distribution scheme which uses the web log information to improve the distribution at the front-end and pre-fetching at the backend servers.

4.1 At the Backend

As explained earlier, the users visiting the websites are categorized into the specific groups using our web log mining script. The requests from a particular user can be monitored and identified to a particular group by correlating the user's current access path and the information from the log mining. This is achieved by correlating (a simple string matching) the current user access path with the pre-defined paths in correspondence with each of the group/category of the website. The longer the comparison paths, the better the confidence of the predicted category is [18].

Once the category of the user is established with the above matching, the related data files can be pre-fetched into the backend servers' memory. The data files can be pre-fetched into the backend servers' memory depending on the current length of the access path. The files immediately below the current access location on the user navigation tree will be pre-fetched into the cache. An example at this mechanism is shown in Fig. 2.

4.1.1 Algorithm for categorizing

We use n-order dependency graph in our system. In fig 3, 2-order dependency graph is illustrated. Each node in the graph represents a web page and the edges stand for the confidence value into the continuing sequence (user navigation pattern). Our algorithm analyzes and categorizes the user's request into specific groups. In the figure, we have two sequences which contain page 'D'. The 70% of sequences that start from page 'A' visit page 'C', while 60% of sequences that start from page 'B' visit page 'E'.

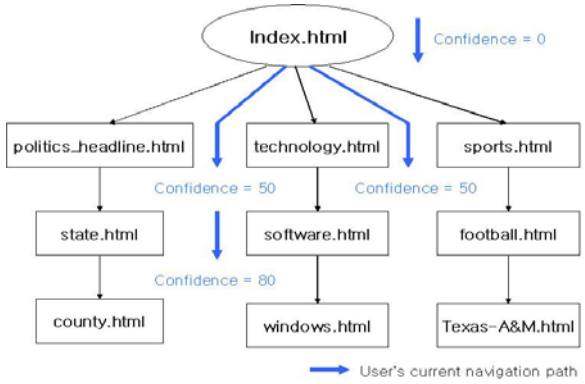

Fig. 2 An Example of Building the Confidence of the Guesses

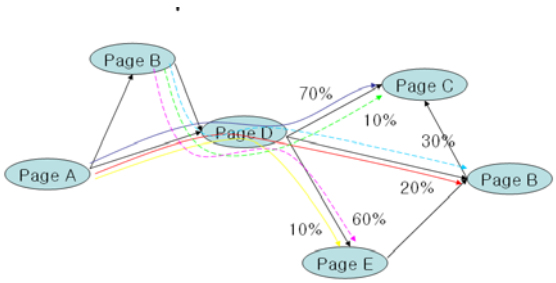

Fig 3. 2-Order Dependency Graph in PRORD

i) Constructing map

It is critical how relations between pages are stored. The necessary space for requested sequence from users depends on the data structure and the order of the sequence. Even though the amount of web log mining information contributes directly to better pre-fetching, it leads to severe memory constraints. For each *l* web log mining sequence and page *n*, the memory needed is '$n * \sum_{i=0}^{l-1} (n-1)^i$ '. Thus memory needed for the web log information increases exponentially based on the order of sequence. Hence only the relationship between the pages is stored instead of the pages themselves.

ii) Finding a candidate path

We find the candidate paths for sequences of requested pages from users based on the number of pre-accessed pages. Algorithm. 1 illustrates our approach. For all pages in the site, the function '*make_candidate_path*' is called. Initially, the path and current page is set to its own page. For example, in the call for page '*a*', '*path*' and '*current_page*' is set to '*a*'. In each call, candidate path is added into array '*candidate_ path*'.

```
Function: make_candidate_path (order, Path, current_page) {
    if (order > 0) {
        For ( current_page linked page b ) {
            Path <- Path U {b}
            make_candidate_path (order – 1, Path, 'b')
        }
    }
    else
        candidate_path[current_page] <-
            candidate_path{current_page} U {Path}
}
```

Algorithm. 1 Making candidate path

iii) Making set for pre-fetching

In each request, Algorithm 1 is run the graph is updated with the candidate paths. For every incoming request '*sequence'* and *'previous_page'* are assigned to each connection. *'Sequence'* stands for the sequence of the previous accessed pages and *'previous_page'* is for most recently accessed page. Function *'get_prefetch_ page'* is called on every request from the user. The current sequence and the candidate paths are compared to increase the hit rate for the incoming requested page. '*Sequence'* and the *'previous page'* are updated. Finally, the cluster searches for the web-page with the highest possibility for next request from user. If the possibility of the chosen page is bigger than threshold, this page is pre-fetched.

```
Function: get_prefetch_page (sequence, requested_page) {
    if (compared sequence from users c1 with candidate_path ) {
        hit_candidate_path [sequence][p1] <-
            hit_candidate_path [sequence][p1] + 1

        sequence <- sequence + requested_page;
        Accessed_Num[requested_page] <-
            Accessed_Num[requested_page] + 1;
        Previous_page = requested_page;
        Pick up the large value in hit_candidate_path [sequence].

        if ((Picked up value/Accessed_Num[requested_page])
            > Threshold)
            return Picked_Page.
        else
            return Null.
    }
}
```

Algorithm. 2 Pre-fetching of pages

4.1.2 Replication at Backend Server

Also, when a request for a web page arrives at the backend, the embedded objects associated with that page are pre-fetched into the cache. The subsequent requests from the user for the embedded objects is forwarded by the distributor to this backend server and this avoids a disk access and hence the latency. Additionally, the popularity of the files, as registered by the recorded hits for each of the web pages, is used to rank the web pages. The files are distributed and replicated across the backend servers' memory based on these rankings. The higher the ranking of the pages and requests to these pages, the larger the replication of these pages on the backend servers' memory is. Algorithm. 3 illustrates the replication algorithm.

```
For every 't' seconds do :

(i) Sort (rank_table)

(ii) For every element in rank_table do :
    if (rank_table[i]. Rank > T1)
        Replicate (rank_table[i].file, all);
    else if (rank_table[i].rank is btw T1_1/2 & T1_3/4)
        Replicate (rank_table[i].file, all_3/4);
    else if (rank_table[i].rank is btw T1_1/4 & T1_1/2)
        Replicate (rank_table[i].file, all_1/2);
    else if (rank_table[i].rank is btw T1_1/8 & T1_1/4)
        Replicate (rank_table[i].file, NO_CHANGE);
    else
        Replicate (rank_table[i].file, NONE);

(iii) Return to step 1.
```

Algorithm. 3 Replication at Backend Server

The replication algorithm is set to run centrally on the whole cluster system. The interval of operation ('t' seconds) is decided based on the current operating conditions of the system (load, service time, etc) or a fixed interval, whichever is earlier. A rank table (rank_table) is built based on the number of hits registered for each of the web pages through dynamic log mining of the recent history. Each one of the files has a "rank" associated with it which is calculated based on the popularity of the file. Based on the value of "rank," the files are replicated across the backend servers through the 'Replicate' function. The attributes 'all', 'NO_CHANGE,' and 'NONE' are self explanatory.

4.2 At the Front-end

As explained earlier in the related work section, the distributor of a web cluster systems is the bottleneck of the system and an efficient distribution policy is needed for improving the performance of the system. In any locality based distributed systems, we can divide the distribution process into several steps. First, distributor analyzes the incoming request. Fig. 4 illustrates the steps involved in distributing the incoming requests. The first step is shown as 'read the incoming request' process in Fig. 4. Second, the distributor should maintain a record of the already distributed files and their locality in the cache. Then, the distributor selects a least loaded backend server which hosts the file in the memory. Finally, the incoming requests are forwarded to the selected backend server.

From our observation, requests for the embedded objects and the web pages which will be requested in the near future (identified from web log mining) are expected to arrive in the immediate future. User requests for the embedded objects arrive at the server as individual requests spanned over a period of time. If these requests are processed individually, every request requires a dispatch and hence increases the processing overhead. Also, the requests incur misses, when the content is not available on the backend. For improving the processing, a module for tossing the request into 'forward module' is added in the flow at front-end. It is enclosed by the dash-line in Fig. 4. This mechanism decreases the number of dispatches dramatically and improves the performance of the cluster system. In section 5, we present the results of the simulation show the improvement achieved with the help of these schemes.

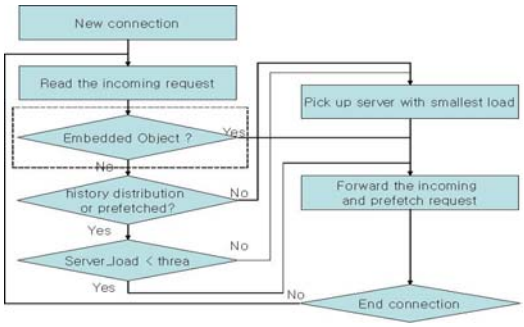

Fig. 4 Proactive Request Distribution Flow Chart

5. Simulation Model and Results

5.1 Simulation Model

The simulation model consists of a distributor/dispatcher and "n" backend servers. Our model is scalable to any number of backend servers and we show that results are consistent with 6 to 16 backend servers. The model emulates a real-time cluster system with request queues at the distributor and the backend servers. The simulation model is illustrated in Fig. 5.

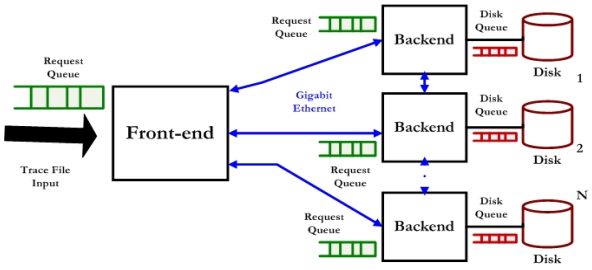

Fig. 5 Simulation Model

The simulation system parameters are enumerated in table 1.

Parameter	Value
Memory (Kernel memory + Application memory)	256, 133 MHz
Kernel Memory	128 MB
Application Memory	128 MB
Pinned Memory	72 MB (Variable)
Connection latency	150 µs
Disk latency	18.215 ms (fixed) + 15.5 µs per KB
Power Consumption	100% when ON, 0% when OFF and 5% in Hibernation
Interconnection Network	100 Mbps Fast Ethernet
TCP handoff latency	200 µs per request
Data transmission rate (across network – for migration)	80 µs per 1 KB block

.

Table 1: Simulation System Parameters

5.2 Simulation Results

Simulations have been carried out by implementing the proposed algorithms in C++. The program is a scalable, user configurable cluster with realistic system and disk queues. Additionally, we have implemented the WRR, LARD, and existing algorithms for P-HTTP (Ext-LARD-PHTTP) for benchmarking/comparison purposes. The simulation code emulates a cluster system, which takes any log file in common log format as the input. The log files used for the simulations are the request logs to the Texas A&M University CS department website (27,000 requests and 4,700 files of average size 12Kb) and the request logs of the Soccer World cup 1998 website (897,498 requests for 3809 files), for one full day. We have also used a set of synthetic web trace for the simulations (30,000 requests, 3000 files of average size 10Kb). In the first section of the results, the efficiency of the distributors of LARD and our system, PRORD, are compared. In the second section, the following metrics are closely monitored for evaluating the performance of the system: Average Response Time and Throughput. We compare our policy (PRORD) against WRR, LARD and Ext-LARD-PHTTP.

Fig. 6 shows the reduced number of dispatches with our policy. This is largely due to the effect of forwarding of requests to the embedded objects. The dispatcher does not have to be contacted for any of the requests comprising the embedded objects. The throughput of all the algorithms for

each of the trace is compared in Fig. 7. The throughput is the summation of the number of requests processed by each of the backend servers. Our scheme performs considerably better than the LARD system with an improvement of 10% to 45 %. The improvement in both LARD and PRORD over WRR is due to the reduced disk accesses or the improved hit rates in the memory of the backend servers. Generally, about 30% of the website's data can be accommodated in the backend servers' memory at any given point of time. This assumption yield 85% hit rates with LARD and 10% boost with our scheme.

To prove that our system has a better locality than LARD, we run simulations varying the amount of data that can be accommodated in backend servers' memory. We varied the amount of website's data that can be accommodated in the backend servers' memory and recorded the throughput. This is illustrated in Fig. 8.

This illustration shows that PRORD is more consistent in preserving the locality of the files than LARD. This comparison has been necessary to portray the efficiency of PRORD. The scenarios depicted here can be a possibility with large websites with large data contents.

PRORD consists of the enhancements outlined in section 4 which improve the locality of the web pages and files in the memory of the backend servers. To identify the individual improvements provided by each of the enhancement, we ran the simulations by turning ON/OFF these enhancements. Fig. 9 illustrates the throughput comparison of each of the enhancement schemes. LARD-bundle denotes the bundle-based distribution scheme. LARD-distribution stands for the improvement achieved through the dynamic distribution of the files on the backend servers' memory based on their popularity. Finally, LARD-prefetch-nav denotes the enhancement achieved through proactive pre-fetching in the backend servers' memory through web log mining. It can be seen that pre-fetching complemented by web log mining provides the best improvement clearly outperforming the other schemes by 100%. Also, PRORD is the combination of these schemes and performs better as the schemes are complementary among themselves.

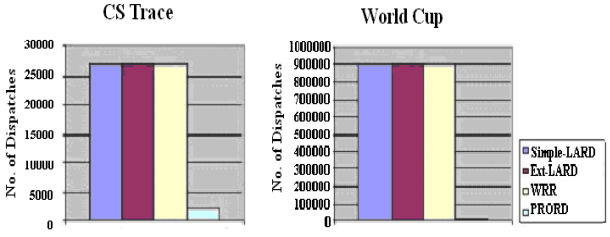

Fig. 6 Number of Dispatches

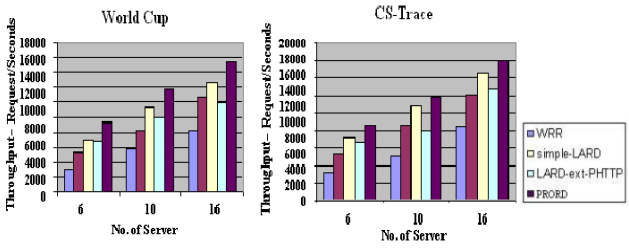

Fig. 7 Throughput Comparison

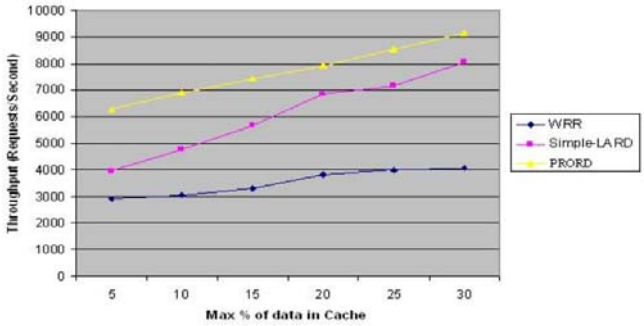

Fig. 8 Throughput varying data amount in memory

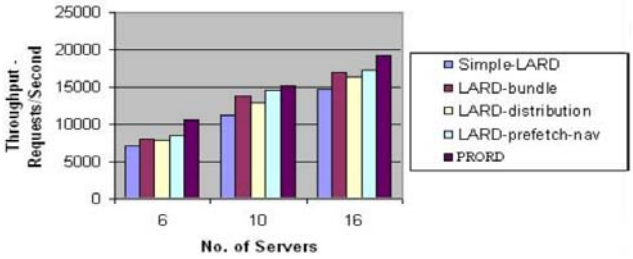

Fig. 9 Throughput Comparison for Individual Enhancements with CS-Trace

6. Conclusions

As the use of cluster systems increases, improving performance has been a critical issue. In this paper, we propose a proactive request distribution scheme, called PRORD, and compare this with three other policies: WRR, LARD and Ext-LARD-PHTTP determine the policy that provides best results in terms of efficiency (throughput). WRR has a good load balancing capability, but its locality is so poor that it increases miss rates. In order to reduce the miss rates and improve secondary storage scalability, LARD can be used.

However, for large websites with immensely huge dataset, where caching considerable website content

becomes impossible, performance of LARD degrades. Thus, we propose PRORD that employs "PROactive" locality-based request distribution which is complemented by prefetching at the backend servers. Such dynamic reconfiguration of the mining usage data in the web server's cache becomes a significant factor for the contribution of the performance the system. The simulation results with original website logs indicate that our system provides considerable improvement in the performance of the system (10-45%). As a future extension, we can explore the possibility of providing support for dynamic contents.

7. REFERENCES

[1] C. Huitema, "Network vs. server issues in end-to-end performance." Keynote speech at Performance and Architecture of Web Servers 2000, Santa Clara, CA. http://www.huitema.net/talks/server-and-networks.ppt.

[2] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, "Locality-aware request distribution in cluster-based network servers," In Proc. 8th intl conf. on Architectural support for programming languages and operating systems, p.205-216, Oct 1998.

[3] K. Rajamani and C. Lefurgy, "On evaluating request-distribution schemes for saving energy in server clusters," in Proc. Intl. Sym. Performance Analysis of Systems and Software, Mar 2003.

[4] M. Aron, D. Sanders, P. Druschel and W. Zwaenepoel, "Scalable Content-aware Request Distribution in Cluster-based Network Servers," In Proc. of the USENIX 2000 Annual Technical Conference , June 2000.

[5] M. Aron, P. Druschel and W. Zwaenepoel, "Efficient Support for P-HTTP in Cluster-Based Web Servers. In Proc. of the Annual Usenix Technical Conference, 1999.

[6] S. Ramakrishnan and Y. Yinghui, "Mining Web Logs to Improve Website Organization." In Proc. of WWW-10, May 2001.

[7] C. E. Wills, G. Trott and M. Mikhailov, "Using Bundles for Web Content Delivery." In Proc. ACM Computer Network, August 2003.

[8] P. Mike and E. Oren, "Adaptive Web Sites: Automatically Synthesizing Web Pages." In Proc. Of the 15th National Conf. on Artificial Intelligence (AAAI), 1998.

[9] P. Mike and E. Oren, "Towards Adaptive Web Sites: Conceptual Framework and Case Study." In Proc. of WWW-8, May 1999.

[10] N. Takehiro, K. Hiroki, Y. Yohei, "Discovering the Gap Between Website Designers' Expectations and User's Behavior.". In Proc. Of WWW-9, May 2000.

[11] S. Myra and F. C. Lukas "WUM: A Web Utilization Miner." In Proc. Of EDBT Workshop WebDB98, Spain, March 1998.

[12] S. Myra, F. C. Lukas and W. Karsten, "A DataMiner analyzing the Navigational Behaviour of Web Users." In Proc. of Workshop on Machine Learning in User Modeling, June 1999.

[13] A. Cohen, S. Rangarajan, and H. Slye, "On the Performance of TCP Splicing for URL-Aware Redirection," In Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems, Oct 1999.

[14] K. Fall and J. Pasquale, "Exploiting In-Kernel Data Paths to Improve I/O Throughput and CPU Availability," In Proc. of the winter 1993 USENIX Conference, Jan 1993.

[15] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum "Locality-Aware Request Distribution in Clutster-based Network Servers," In Proc. of the 8th Conference on Architectural Support for Programming Language and Operating Systems, Oct 1998.

[16] E. Frias-Martinez and V. Karamcheti, "A Prediction Model for User Access Sequences." In Proc. of WEBKDD Workshop: Web Minig for Usage Patterns and User Profiles, 2002.

[17] M. Kitsuregawa, M. Toyoda and I. Pramudiono, "Web Community mining and WEB log mining:Commodity cluster based Execution," In Australasian Database Conference, 2002

[18] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "A Data Mining Algorithm for Generalized Web Prefetching," In IEEE Tran. on Knowledge and data engineering, 2003.

[19] V.N. Padmanabhan, J.C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," ACM SIGCOMM Computer. Communication Review, 1996.

[20] Q. Yang, H.H.Zhang, T.Li, "Mining Web Logs for Prediction Models in WWW Caching and Prefetching," In Proc. of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001.

[21] M. Gery, H.Haddad, "Evaluation of Web Usage Mining Approaches for User's Next Request Prediction." In Proc. of the 5th ACM intl workshop on Web Information and Data Management(WIDM), 2003.

[22] J. Srivastava, R. Cooley, M.Deshpande and P.N.Ten, "Web Usage Mining: Discovery and Application of Usage patterns from Web Data," In SIGKDD Exploration, 2000.

[23] R. Agrawal, T. Lmielinski, A. Swami, "Mining association rules between sets of items in large databases," In Proc. of ACM SIGMOD conference on Management of data, 1993.

[24] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Assocation Rules," in Proc. VLDB, 1994.

[25] R. Agrawal and R. Srikant, "Mining Sequential Patterns," in Proc. of Intl Conference on Data Engineering.

[26] T. Palpanas and A. mendelzon, "Web Prefetching Using Partial Match Prediction," In Proc. 4th Web Caching Worshop, 1999.

[27] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Using sequential and non-sequential patterns for predictive web usage mining tasks," In Proc. of the IEEE Intl Conference on Data Mining (ICDM), 2002.

[28] W.Gaul and L. Schmidt-Thieme, "Mining web navigation path fragments," In Workshop on Web Mining for E-commerce – Challenges and Opportunities, 2000.

[29] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters," In Proc. of ACM SIGCOMM, 1998.

[30] L. Cherkasova, "Improving WWW proxies performance with greedy-dual-size-frequency caching policy," In HP Technical Report, 1998.