A PROactive Request Distribution (PRORD) Using Web Log Mining in a 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 dominates 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 caches. In this paper, we propose a proactive request distribution (PRORD) that applies an intelligent proactive-distribution at the front-end and complementary pre-fetching at the back-end server nodes to obtain the data of high relation to the previous requests in their caches. The pr-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. 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 experienced by the end-user is composed 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 to decode the incoming request and forward the request to one of the back-end servers. Thus, the delay at the web server is a critical component that has to be reduced to achieve better web-server performance.

Among the different architectures in the cluster-based servers, the distributor-based systems have been widely adopted as shown in Fig. 1. These systems have a front-end distributor that forwards the requests to any of the backend servers. In the locality-based request distribution schemes [2, 4], 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 better locality for the requested file. The role of the dispatcher is to notify the distributor of the locality of the requested files. The forwarding of the requests from the distributor to the backend servers is carried out in complete transparency with the users. A handoff protocol and TCP splicing are employed in most cases to make the transition smooth and transparent [2, 13, 14]. The requests are forwarded to a 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 a few of the most prolifically adopted policies. The policies focus on improving efficiency, power conservation, and load balancing, respectively.

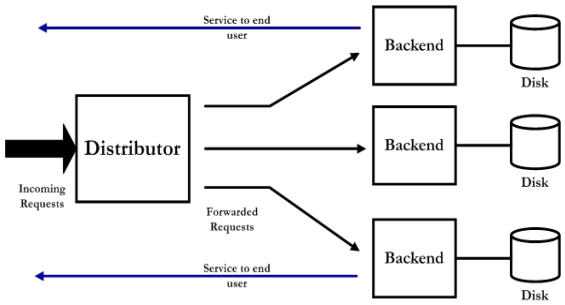

Fig. 1: Distributor-Based Web Server System

In this paper, we propose a new proactive request distribution scheme (PRORD), which reduces the delay at the cluster-based 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 server log files. Web server log files let us know the critical

information, including the general user navigation pattern, user behavior and general website organization. The extracted information from web log file is made available for the distributor at the front-end to discern and classify the incoming requests and performs the dispatch to the appropriate backend server. Besides, this extracted information is used to prefetch the data files into the web servers' cache [20]. Prefetching is thus complementary to the dispatches being made by the distributor at the front-end. Simulation results with traces from real web servers show that the proposed scheme, PRORD, outperforms other distribution policies. In particular, PRORD shows 10% ~ 45% improvement over LARD.

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

Among the distributor-based policies, weighted round robin (WRR) is considered to be a simple and efficient scheme for providing excellent load balancing of the requests arriving at the system. However, it does not affect the performance of the system. The locality-based request distribution schemes focus on improving the performance through intelligent distribution of the incoming requests [2, 4, 5].

2.1 Locality-Aware Request Distribution (LARD)

The LARD [2] overcomes the drawbacks faced by the WRR policy. It increases the memory hits at backend servers by considering both data locality and load balancing issues in a distributed cluster-based server. The distributor in the 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 the selected node is high, then the requests are forwarded to another lightly loaded node that has the contents in its disk. As an improvement on this idea, Aron, et al. [4] proposed a scalable content-aware distribution policy that decreases a heavy load of the front-end node through paralleling distribution work of the incoming requests using a de-centralized distributor. 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 studies are based on HTTP 0.9/1.0-based web transactions. With HTTP 1.1 based web transactions, the persistent connection suffers from inefficient distribution of the incoming requests among back-end servers. While the front-end in a cluster system with HTTP 0.9/1.0 can handle every incoming request as an individual connection, multiple requests from the same client are coming through one single connection in HTTP 1.1. Therefore, when the front-end for HTTP 0.9/1.0 is operating on HTTP 1.1, it can not guarantee that all the requests are distributed under the locality-aware distribution policy.

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 the 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 serviced among the back-end server nodes.

Both the above techniques suffer from high overhead. We try 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 frequently used in web services [7, 8, 9, 10, 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 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 large websites, the users' target document does not exist in the users' expected location.

In [6], web log information was 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] used the web log files to discover the gap between website users' behavior and the website designers' expectations. They evaluated these metrics using inter-page access co-occurrence and inter-page conceptual relevance, respectively. Perkowitz, et al. [8, 9] developed a clustering algorithm to identify web pages that occur together in a single user visit and built an index

page, which helps the users to effectively navigate the website. Spiliopoulou, et al. [11, 12] proposed a web mining tool called Web Utilization Miner (WUM) 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 patterns of the users can help re-organize the website such that the required target data is readily available to the users.

2.2.2 Bundling Requests

[7] shows that prefetching of the embedded objects associated with a particular page could provide considerable performance boost. The webpage and its associated embedded objects such as images, applets, etc. were grouped into a "bundle" and delivered to the user browser in a compressed form on the request of the web page. In [29], Cohen, et al. proposed an improved method for updating the cached webpage at the proxy servers. During the update of the bundled information, the update period for the stale webpage was prolonged due to a slight increase of update overhead. Log mining information was used for creating the update information based on the relationship between webpages, 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 files in the system. A considerable amount of schemes suggested for improvement of web search can be classified into two groups; association rule [23, 24] and sequence rule [25]. The association rule uses set-based operations for analyzing the log files, while, in the sequence rule, the system analyzes the sequence of the web log files. Kitsuregawa improved the scheme based on association rule [23, 24] and sequence rule [25] and implemented a Mobile Information Search (MIS) system through a PC cluster [17].

[20] presented another method for prefetching through web log mining. Embedded Object Table (EOT) and a set of association rules were constructed for prefetching and caching the page. Besides, they extended GDSF [30] through splitting frequency into future frequency and past frequency through an association rule. [21] compared three web mining approaches: association rules [23, 24], sequence rules [25, 27] and generalized sequence rules [28]. They proved that sequence rules outperform the other approaches.

Data structure for prefetching using web log mining can be categorized into two schemes: Dependency graph (DG) [19] and Prediction-by-Partial-Match (PPM) [26]. In [19], Padmanabhan, et al. proposed a scheme for prefetching through website log mining. The scheme used a prediction engine which kept track of web page relationship information. A weighted direct graph is constructed where nodes correspond to the web pages and the arcs represent the relationship between the web pages. In PPM [26], *j*-order Markov predictor was maintained for the prediction in the comparison of the previous *j* accessed pages. Even though the predictor generates more accurate result with higher orders, the overhead for maintaining the predictor increases, and it becomes the bottleneck of the scheme. [18] proposed a scheme for prefetching webpages through the PPM and the DG. In [17], MIS was described for collecting and clustering the web pages.

Though there have been a lot of studies carried out in web log mining, its usage in improving the distribution policy in cluster-based web servers has not been explored. Besides, the information extracted from log files by our algorithms and their usages to our system are unique.

3. UTILIZING WEB LOG FILES

We employ the web log files to collect a host of information: users' navigation pattern, popularity of the web pages and 'bundles' of pages. This web log information can be directly used for discerning the incoming requests and dispatching them to the appropriate backend server nodes. Web log information segments and their usage are elaborated in the following subsections.

3.1 Users' Navigation Pattern

We use the script to analyze the log files, garner the access patterns of the users on a website and group web pages using the collected information. Every website can be subdivided based on the different categories of web users who visited the website. For example, a university website will most likely cater to the needs of current students, prospective students, faculty members, support staff and other users. Thus, the users on the university website can be categorized into such well known groups including current students, prospective students, faculty, staff and others. Each of these groups' users has a highly directional and mostly unique access pattern. Thus, the web log 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 retrieved from the log files and number of requests 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 the realistic estimate for 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 main web page. The embedded 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

In this section, we present a new proactive request distribution scheme, called PRORD, which uses the web log information to improve the distribution at the front-end and provide prefetching at the back-end servers.

4.1 Prefetching at the Back-End Servers

A back-end server prefetches a specific group of data containing currently requested pages based on user's access pattern using web log mining scripts. The requests from a particular user can be monitored and identified as a particular group by correlating the user's current access path and the information from the log mining. This is achieved by comparing the current user access path with the predefined paths in correspondence with each of the group or category of the website. The longer the comparison paths are, the better the confidence of the predicted category is [18].

Once a user is categorized using the above matching, the related data files can be prefetched into the back-end servers' memory depending on the confidence of the data file under the current user's access path. The files with high confidence immediately below the current access location on the user navigation tree will be prefetched into the cache. An example of this mechanism is shown in Fig. 2.

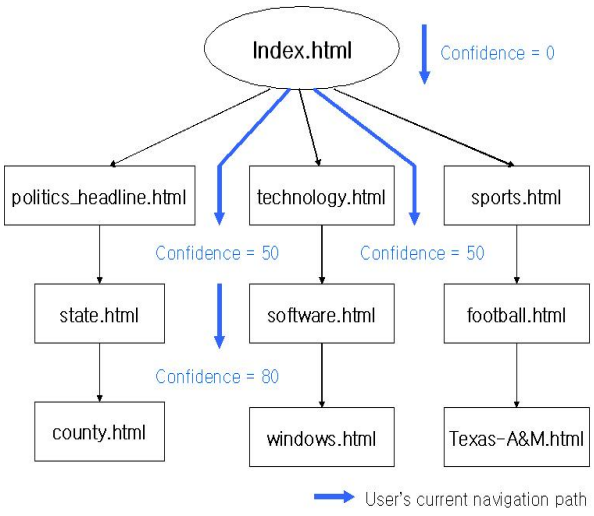

Fig. 2 An Example of Building the Confidence of the Guesses

4.1.1 Algorithm for Categorizing

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

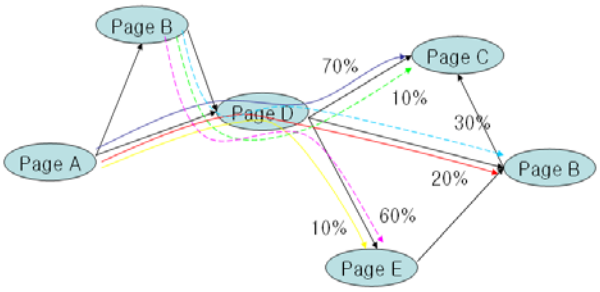

Fig. 3 2-Order Dependency Graph in PRORD

i) Constructing map

It is critical how to store a dependency graph representing the relation between pages in the limited memory space. 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 prefetching, it leads to severe memory constraints. For l web log mining sequences with the total number of n pages, all possible number of relations between pages stored on the memory is n^{l+1}. Thus memory for the web log information increases exponentially according to the order of the sequence. To avoid this space overhead, we propose to store relations between pages only when one page is directly linked to other pages.

ii) Finding a candidate path

We find the candidate paths for sequences of requested pages from users based on the link between pages. Algorithm 1 shows the details of 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*', both '*path*' and '*current_page*' are set to '*a*'. In each call, '*current_page*' 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 prefetching

In each request, Algorithm 2 selects the candidate page and updates the hit rate of each sequence. 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 hit rate of sequence with requested page is increased and '*previous_page*' is updated with requested page. Finally, the web-page with the highest possibility for next request from user is selected. If the possibility of the chosen page is bigger than threshold, this page is prefetched.

Also, when a request for a main page arrives at the backend, the embedded objects associated with main page are pre-fetched into the cache. The subsequent requests from the user for the embedded objects is forwarded by the distributor to backend server with pre-fetched embedded object and this scheme avoids a disk access and hence the latency.

```
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 Prefetching of pages

4.1.2 Replication at the Back-End Server

The popularity of the files, as registered by the recorded hits for each web page, is used to rank the web pages. The files are distributed and replicated across the back-end servers' memory based on their ranks. The higher the ranks of the pages are, the larger the replication of these pages on the back-end servers' memory is. Algorithm 3 explains the replication process.

```
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 the Back-End Server

The replication algorithm controls whole replicas of web documents in the system. The interval of the operation (*t* seconds) is decided based on the current operating conditions of the system (load, service time, etc.) or a fixed interval, whichever is smaller. A rank table (`rank_table`) is built based on the frequency of hits registered for each web page through dynamic log mining of the recent history. Each one of the files has a "rank" associated with the popularity of the file. Based on the value of "rank," the files are replicated across the back-end servers through the '*Replication*' algorithm.

4.2 Request Distribution at the Front-End

To solve the bottleneck formed by the distributor of a web cluster system, the efficient distribution through web log mining is suggested in this section. In any locality based distributed systems, we can divide the distribution process into several steps. Fig. 4 illustrates the steps involved in distributing the incoming requests. First, distributor read and analyzes the incoming request. Second, distributor determines whether current request is for the embedded object for previous request. If it is for embedded object, request is forwarded to backend server that processed previous request. Third, the distributor decides whether request is for pre-fetched object or is already distributed. If it is, request is distributed to backend server that has pre-fetched object or already processes it. If not, 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, interval between request and following request is short. User requests for the embedded objects arrive at the server as separate requests over a period of time. If these requests are processed individually, every request requires a dispatch and hence increases the processing overhead. Besides, the requests incur misses, when the content is not available on the memory at 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.

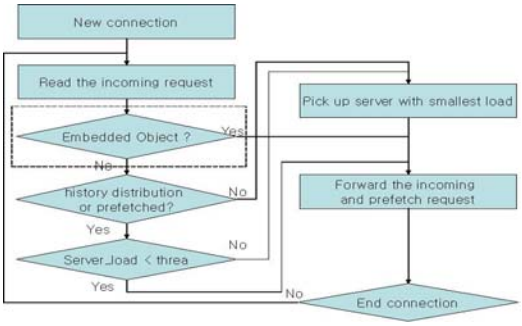

Fig. 4 Proactive Request Distribution Flow Chart

5. SIMULATION MODEL AND RESULTS

5.1 Simulation Model

The simulation model consists of 'n' backend servers and a front-end including distributor and dispatcher. 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.

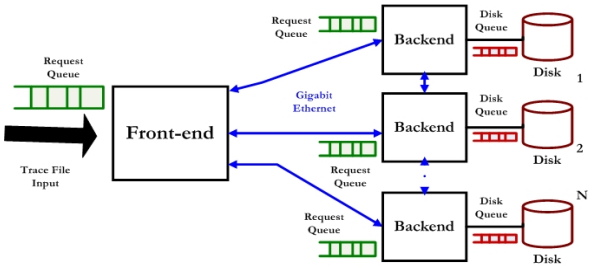

Fig. 5 Simulation Model

The simulation system parameters are enumerated in table 1.

Parameter	Value
Memory (Kernel memory + Application memory)	256 MB, 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 System Parameters

5.2 Simulation Results

Simulations have been carried out by implementing the proposed algorithms in C++. The program simulates 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 or comparison purposes. The simulation code takes any log file in common log format as the input. In our simulation, we obtain workload from logs of Texas A&M University CS department web site and from Soccer World cup 1998 web site. The data set for Texas A&M University CS departments contains 27,000 requests and has 4,700 files of average size 12Kb, while logs for Soccer World cup 1998 contains 897,498 requests for 3809 files We have also used a set of synthetic web trace for the simulations composed of 30,000 requests and 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 frequency 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.

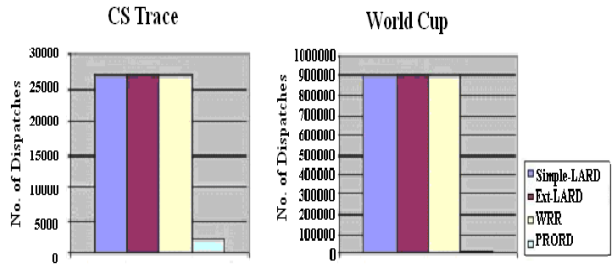

Fig. 6 Frequency of Dispatches

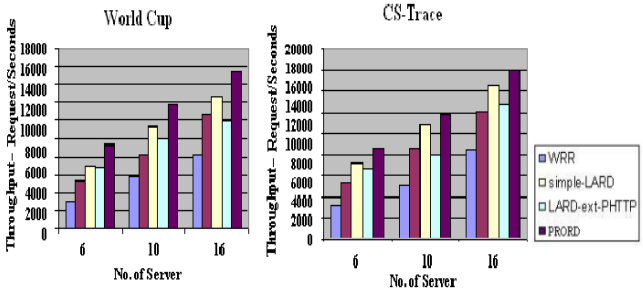

Fig. 7 Throughput Comparison

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. Also, 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 indicates 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.

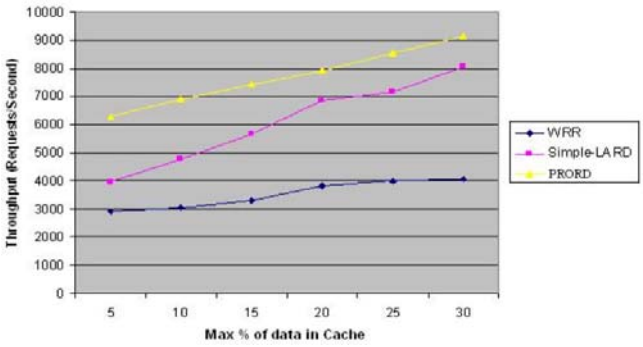

Fig. 8 Throughput varying data amount in memory

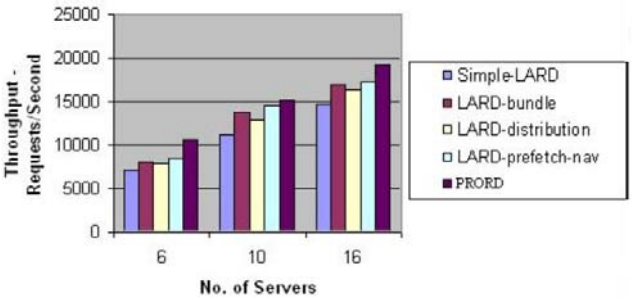

Fig. 9 Throughput Comparison for Individual Enhancements with CS-Trace

6. CONCLUSIONS

As the use of cluster systems increases, improving performance has been a critical issue. In this paper, we proposed a proactive request distribution scheme, called PRORD, compared it with three other policies including WRR, LARD and Ext-LARD-PHTTP, and showed which policy provides best results in terms of efficiency such as throughput and frequency of dispatches. 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 datasets, where caching considerable website contents

becomes impossible, the performance of LARD degrades. Thus, we propose PRORD that employs 'PROactive' locality-based request distribution which is complemented by prefetching at the back-end 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 of the system. The simulation results with original website logs indicate that our system provides considerable improvement in the performance of the system (10-45%). We are planning to explore the possibility of providing support for dynamic contents.

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, and 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, pp. 205-216, October 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, March 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, and 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, October 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, January 1993.

[15] J. Kim, G. S. Choi, and C. R. Das, "A Load Balancing Scheme for Cluster-based Secure Network Servers," in Proc. of Cluster 2005, 2005.

[16] E. Frias-Martinez and V. Karamcheti, "A Prediction Model for User Access Sequences," in Proc. of WEBKDD Workshop: Web Mining 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 and J. C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," ACM SIGCOMM Computer Communication Review, 1996.

[20] Q. Yang, H.H.Zhang, and 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 and 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, and 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 Association Rules," in Proc. of VLDB, 1994.

[25] R. Agrawal and R. Srikant, "Mining Sequential Patterns," in Proc. of 11th Intl. Conference on Data Engineering, 1995.

[26] T. Palpanas and A. Mendelzon, "Web Prefetching Using Partial Match Prediction," in Proc. 4th Web Caching Workshop, 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," HP Technical Report, 1998.

[31] L. Aversa and A. Bestavros, "Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting," in Proc. of the 2000 IEEE Intl. Performance, Computing, and Communications Conference, Feb 2000.

[32] E. V. Carrera and R. Bianchini. "PRESS: A Clustered Server Based on User-Level Communication". IEEE Transactions on Parallel and Distributed Systems, volume 16, number 5, May 2005.