ON IMPROVING PERFORMANCE AND CONSERVING POWER IN CLUSTER-BASED WEB SERVERS

A Thesis

by

GOPINATH VAGEESAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2005

Major Subject: Computer Engineering

ON IMPROVING PERFORMANCE AND CONSERVING POWER IN CLUSTER-BASED WEB SERVERS

A Thesis

by

GOPINATH VAGEESAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Eun Jung Kim
	Gwan Choi
Committee Member,	Scott Pike
Head of Department,	Chanan Singh

December 2005

Major Subject: Computer Engineering

ABSTRACT

On Improving Performance and Conserving Power

in Cluster-based Web Servers. (December 2005)

Gopinath Vageesan, B.E., Bharathiar University, Coimbatore

Co-Chairs of Advisory Committee: Dr. Eun Jung Kim
Dr. Gwan Choi

Efficiency and power conservation are critical issues in the design of cluster systems because these two parameters have direct implications on the user experience and the global need to conserve power. Widely adopted, distributor-based systems forward client requests to a balanced set of waiting servers in complete transparency to the clients. The policy employed in forwarding requests from the front-end distributor to the backend servers plays an important role in the overall system performance. Existing research separately addresses server performance and power conservation. The locality-aware request distribution (LARD) scheme improves the system response time by having the requests served by web servers which have the data in their cache. The power-aware request distribution aims at reducing the power consumption by turning the web servers OFF and ON according to the load.

This research tries to achieve power conservation while preserving the performance of the system. First, we prove that using both power-aware and locality-aware request distribution together provides optimum power conservation, while still maintaining the required QoS of the system. We apply the usage of pinned memory in the backend servers to boost performance along with a request distributor design based on power and locality considerations. Secondly, we employ an intelligent-proactive-distribution policy at the front-end to improve the distribution scheme and complementary pre-fetching at the back-end server nodes. The proactive distribution depends on both online and offline analysis of the website log files, which capture user navigation patterns on the website. The pre-fetching scheme pre-fetches the web pages into the memory based on a confidence value of the web page predicted by backend using the log file analysis. Designed to work with the prevailing web technologies, such as HTTP 1.1, our scheme provides reduced response time to the clients and improved power conservation at the backend server cluster. Simulations carried out with traces derived from the log files of real web servers witness performance boost of 15-45% and 10-40% power conservation in comparison to the existing distribution policies.

To my parents …

ACKNOWLEDGMENTS

I sincerely thank Heung Ki Lee of the Department of Computer Science for working with me all throughout my thesis. Without his help and invaluable contributions, this thesis would have never been a possibility.

I am ever grateful to my advisor, Dr. Eun Jung Kim, for providing strong guidance and building my faith all throughout the thesis. It has been a venture of great pleasure and honor working under her supervision.

I thank Dr. Scott Pike for providing invaluable suggestions and helping me to shape this thesis.

I sincerely thank Dr. Gwan Choi for being an immense pillar of support all through this endeavour.

I also thank the Department of Computer Science at Texas A&M University for providing their website's log files, which were critical for the simulation results.

Lastly, I am grateful to all my dear and near who have been a great support through these two years and also for enduring me with patience.

TABLE OF CONTENTS

Page

ABSTRACT……………………………………………………………………………..iii

DEDICATION..……………………………………………………………...…….v

ACKNOWLEDGMENTS…………………………………………………………………vi

TABLE OF CONTENTS…………………………………………………………………vii

LIST OF FIGURES……………………………………………………………………...ix

LIST OF TABLES……………………………………………………………………….x

I. INTRODUCTION……………………………………….………………………. 1

II.BACKGROUND…………………………………………………………………5

1. Weighted round robin……………………………………………………….5
2. Locality-aware request distribution………………………………………6
   2.1 Multiple TCP handoffs ………………………………………………….. 8
   2.2 Backend forwarding………………………………………………………9
3. Multi-speed disks…………………………………………………………… 10
4. Dynamic cluster re-configuration……………………………………… 10
5. Power-aware request distribution……………………………………….11
6. Web log mining………………………………………………………………12
   6.1 User navigation pattern……………………………………………….. 13
   6.2 Bundling requests……………………………………………………...14

III.POWER AND LOCALITY-AWARE REQUEST DISTRIBUTOR..…………………16

1. Combining power policy and LARD…………………………………………...17

Page

IV. ENHANCEMENTS IN LOCALITY-AWARE REQUEST DISTRIBUTION............20

1. Handling persistent HTTP connections……….. 20
2. Memory management scheme………….. 22
   2.1 Data placement.………….. 24
   2.2 Data replacement…………….. 24
   2.3 Migration in pin-down memory………. 25
3. Harnessing web log files………. 26
   3.1 Users' navigation pattern………. 26
   3.2 Popularity of web pages……………. 27
   3.3 Spotting bundles……….. 27
   3.4 Examples illustrating user categorization……………. 27
4. Application of web log information to improve LARD……………................ 28
   4.1 At the backend..………... 28
   4.2 At the front-end…………….. 34

V. ENHANCEMENTS IN POWER POLICY…………………………………............35

1. Power policy……..………... 35

VI. SIMULATION MODEL AND RESULTS……………………………………………39

1. Simulation model……..………… 39
2. Simulation results……..……….. 40

VII. SUMMARY AND CONCLUSIONS...…………………………………………....47

REFERENCES……………………... 48

VITA……………………………. 51

LIST OF FIGURES

FIGURE Page

1 Distributor based web server system …………………………………...………. 3

2 Weighted round robin ….………………………………………….……….... 6

3 Locality-aware request distribution ………………..…………………………... 7

4 Multiple TCP handoffs ……………..……………………………………… 8

5 Backend forwarding …………….…………………………………………….. 9

6 Throughput comparison (PLARD) ……………..…………………………… 18

7 Illustration of the distribution policy ……..……….............................…... 21

8 Pinned memory organization ……………..…………………………………….. 23

9 Building the confidence of the guesses …...…………………………………… 30

10 Algorithm for pre-fetching ……………...…………………………………….. 31

11 Replication algorithm ……………………...………………………………... 33

12 Power transition states: H-Hibernation mode ………………..…………………37

13 Simulation model ………………...…………………………………………. 39

14 Throughput comparison: (a) CS Trace, (b) World cup, (c) Synthetic …...……...42

15 Comparison with LARD …………………………………………..……….44

16 Throughput comparison for individual enhancement …....…………………… 45

17 Power conservation …...…………………………………………………... 46

LIST OF TABLES

TABLE Page

1 Categorization of websites ………………………………………………………….. 28

2 Simulation system parameters ……………………………………………...….. 40

I. 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. 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 backend servers. Thus, the delay at the web-server is a critical component which has to be reduced to achieve a better web-server performance. We present a modified locality-aware request distribution (LARD) [1] scheme, which reduces the delay at the web-server.

The power equipment, cooling and electricity form a significant fraction of the total cost of ownership (TCO) [2]. The effect of power consumption of the servers on the total ownership cost is bi-fold. First, the increased power consumption directly contributes to the increasing electricity bills. Second, the increased power consumption aids to the increase in temperature of the housing and hence the cooling cost of the system.

This thesis follows the style of *IEEE Transactions on Power Systems*.

From [2], these amount to more than 20% of the TCO, which span for the lifetime of the server system. Thus, in addition to the performance and scalability issues, design of energy-efficient cluster-based servers is also becoming increasingly more important from the economic standpoint.

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 data is located in the same backend server, the request is served directly. Else, 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 clients. A handoff protocol is employed in most cases to make the transition smooth and transparent [1]. The requests are forwarded to the set of backend servers based on a certain policy. LARD (Locality Aware Request Distribution) [1], 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.

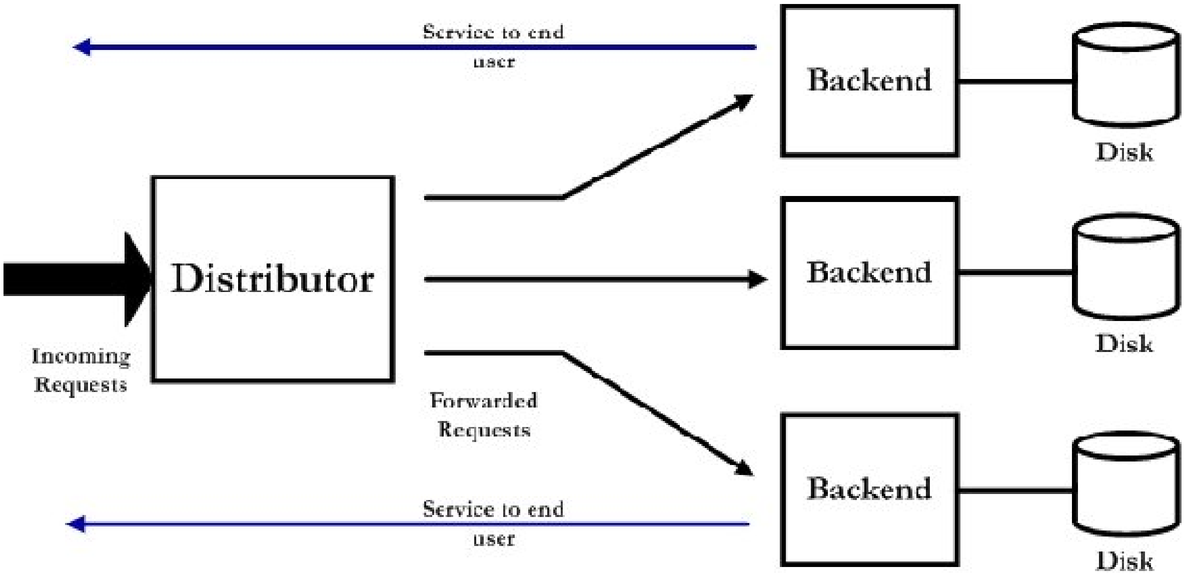


Fig. 1. Distributor based web server system

In this thesis, we try to achieve a balance between high efficiency and good power conservation. Thus, by combining an efficient locality-based distribution scheme and a power-aware request distribution scheme, we develop a system, which strikes a balance between efficiency and power conservation. But, the power conservation is obtained at a cost of performance degradation and an improvement in performance has to sacrifice power conservation. In order to overcome this design trade-off problem, we improve the performance of the system by increasing the locality of the data and including an efficient distribution policy. We introduce a new memory management technique using "pinned" memory to improve the locality, whose physical area is never paged out. We use a proactive request distribution policy based on website organization and website log mining which is supported by an aggressive webpage-based pre-fetching scheme at the backend. The web server logs record the request flow to the servers and are a rich source of information about the users' navigation pattern, page popularity, etc. The logs are mined for this information and it is used for the pre-fetching of probable page requests into the backend server's memory and for initiating proactive distribution at the front-end. This aims to improve the locality of the files in contrast to the LARD scheme which just distributes the requests based on the locality information. Thus, we improve the locality of the files to counter the trade-off that we suffer due to having the power policy.

The new policy is capable of working in conjunction with the prevailing HTTP 1.1 persistent connection technology. We present simulation results with trace files derived from the log files of real web servers to show the performance capabilities of our system.

II. BACKGROUND

The volume of published research in the area of cluster-based web servers and cluster-based application servers testifies to the interest of the numerous researchers. But, in this thesis, we restrict our comparison to WRR, LARD, PARD and Ext-LARD-PHTTP policies.

1. Weighted round robin

The weighted round robin (WRR) policy is one of the simple and widely adopted distribution policies, due to its excellent load balancing capabilities. The distributor maintains a record of the current load at the backend servers. The request is always forwarded to the least loaded backend server among the set of servers. The request forwarding is thus weighted, based on the current load on the servers. The main drawback of WRR is that it does not concern about the locality of the requests (increasing the response time through large disk latencies) and the power conservation among the servers. In case of large deployment of cluster systems, the power consumed becomes a very significant factor. So, power and locality-based request distribution policies have gained significance. Figure 2 illustrates the WRR policy.

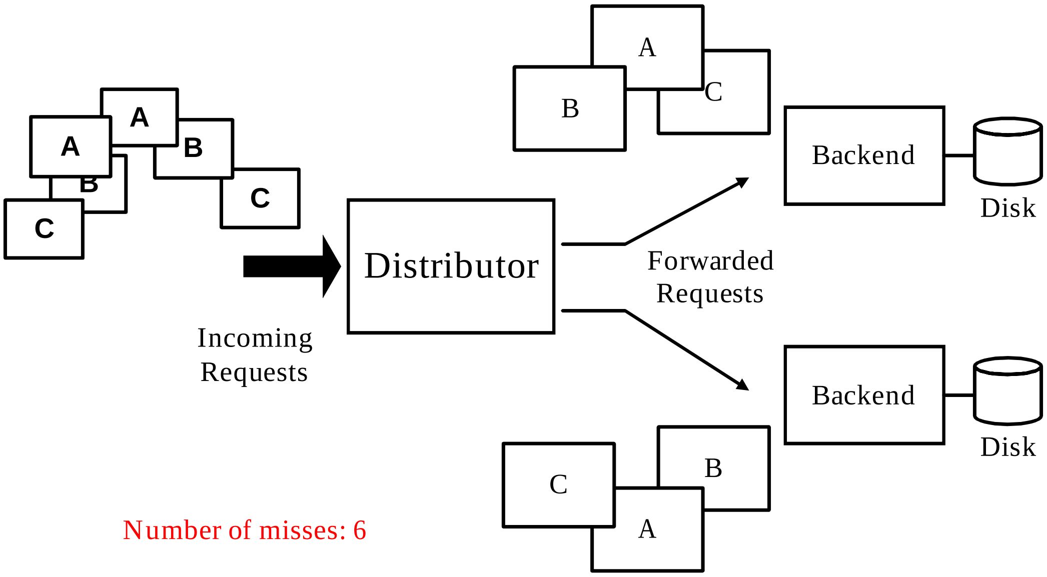


Fig. 2. Weighted round robin

2. Locality-aware request distribution

The LARD [1] 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. This policy is illustrated in Figure 3. In an improvement to this idea, Mohit Aron et al. [4] proposed a scalable content-aware distribution policy that minimizes the bottleneck of the front-end web-switch by using a de-centralized request distribution strategy. In this policy, a layer-4 web-switch is used to distribute the requests to backend nodes.

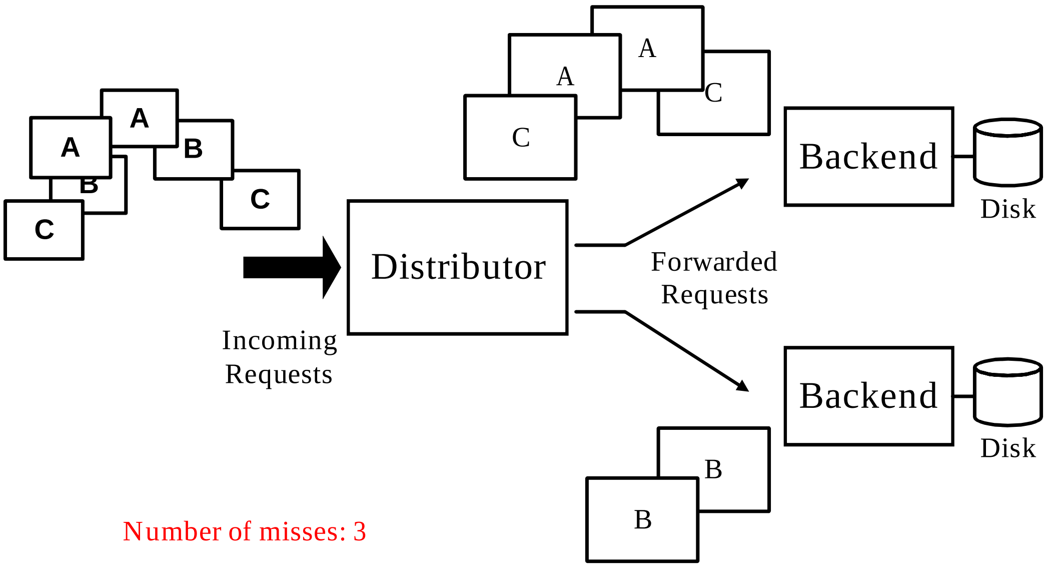


Fig. 3. Locality-aware request distribution [1]

This architecture is scalable over a large number of servers, but suffers from the following drawbacks: The high speed switch can prove to be 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 1.0 based web transactions, the user browser spawns multiple TCP connections for continuing requests to the same server. In this case, the above ideas incur considerable overhead in performing multiple distributions and handoffs. The papers did not consider the effect of their idea on HTTP 1.1 based web transactions, which adopt a persistent connection between the client and the server. In [5], the authors have considered two techniques to tackle this problem: multiple TCP handoff and backend forwarding. We shall examine them in detail in the following sections.

2.1 Multiple TCP handoffs

With HTTP 1.1, the client can request multiple data from the server on the same persistent connection. With multiple TCP handoffs [5], every incoming request is analyzed and dispatched 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 client. This is illustrated in Figure 4.

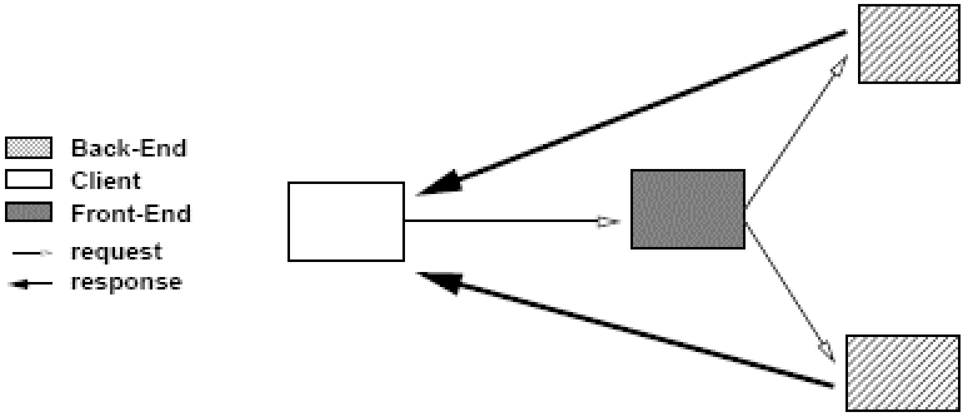


Fig. 4. Multiple TCP handoffs [5]

2.2 Backend forwarding

In backend forwarding scheme, the front end initiates a single handoff for every persistent HTTP connection. The backend servers are connected over a high speed network and the request can be internally forwarded and served among the backend server nodes. This is illustrated in Fig. 5.

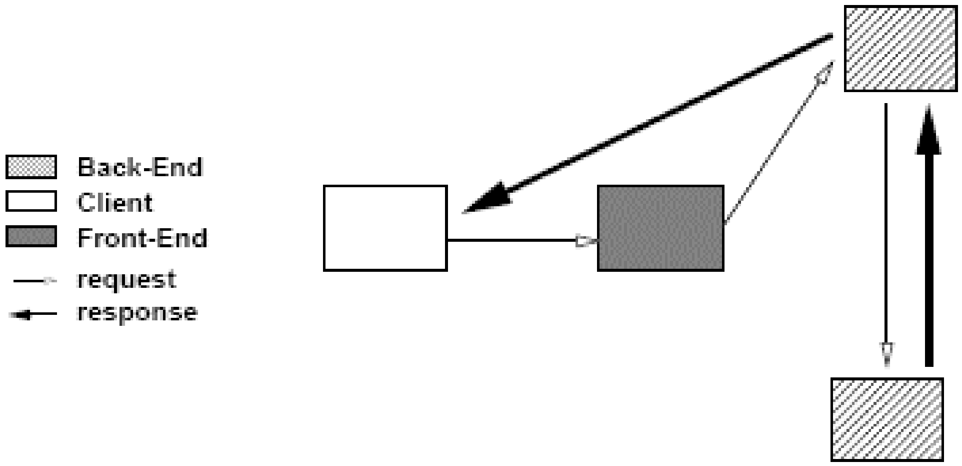


Fig. 5. Backend forwarding [5]

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

On power conservation standpoint, many researchers have previously proposed schemes to reduce power consumption.

3. Multi-speed disks

E.V. Carrera et al., [6] have proposed a technique to conserve power in network servers using a multi-speed disk technology. The idea is to use two disks with lower speeds to emulate a high-speed disk. They also propose the use of multi-speed disks to conserve power in network servers. Power consumed is directly proportional to the speed of the disk. This approach can save power up to 23%, in comparison to conventional servers. They also argue that the performance degradation is very negligible. A major setback to this approach is that it requires multi-speed disks, which are not very popular nowadays.

4. Dynamic cluster re-configuration

Pinheiro et al., [7] have proposed a dynamic cluster reconfiguration technique to bring down the power consumption in servers. In this technique, a cluster node is dynamically added or removed to the cluster system based on efficiency, performance and other power implications of the system. The cluster is dynamically re-configurable and intelligent to reconfigure itself, based on the load and other efficiency-related parameters. This work differs from ours, as we use locality-based scheme to improve the performance, whereas dynamic reconfiguration tries to preserve the performance of an existing system, on top of conserving power.

5. Power-aware request distribution

Our focus is limited to the Power-aware request distribution scheme. In Power-aware Request Distribution [3] (PARD), a simple power scheme is adopted to conserve power. It uses a simple ON-OFF Model, where any backend server, which is idle, is turned off. And backend servers are turned ON whenever the demand for service increases. However, this policy suffers from a performance viewpoint. The policy does not have performance enhancement measures and also reduces the locality of the files by wiping off the contents of the memory which is being turned OFF. Requests have to incur a startup delay when waiting on a server to turn ON from OFF state.

It has been proven that locality-based request distribution schemes bring exceedingly good performance boost to distributor based cluster systems. But, such systems do not have any scheme for power conservation. To overcome this shortcoming in the design, we propose the idea of combining locality based distribution and power aware distribution in cluster systems (PLARD). But, as there is a tradeoff between power and performance factors, our goal is to improve the performance with a state of considerable power conservation. Thus, we focus on improving the locality of the files that are being requested. By increasing the locality, we can achieve better hit rates in the backend servers' memory and hence a performance boost. To achieve this, we use two techniques: the application-level memory management technique using pinned memory and proactive distribution with the aid of analysis and mining of the website logs. Although the usage of pinned memory has been previously suggested in operating systems, distributed systems and web servers, its application in cluster-based servers is new. We use pinned memory to preserve the locality of the files when the servers are turned OFF by the power management policy.

6. Web log mining

Web log mining has been prolifically used in web services [8-14]; however, none of them include the idea 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.

6.1 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 [8], this information is used to improve the website organization by providing hyperlinks to users’ target document on the users’ expected location of the webpage.

Takehiro et al. [12] 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. The gap between these metrics is directly proportional to the inefficiency of the website. Once the metrics are evaluated, they can be used to improve the website organization and page layout to enhance the users’ navigation experience.

Mike and Oren [10-11] 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. The index page can be generated automatically in accordance to the users’ navigation pattern. The algorithm makes the website “adaptive” by equipping it the ability to re-configure the index page according to the user navigation pattern.

Myra et al [13-14] propose 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 WUM consists of a mining language that can be utilized by the website designer through various specifications corresponding to the required level of analysis. Thus, 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.

6.2 Bundling requests

In [9] 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. The bundle is transmitted to the user along with the requested page in an assumption that the embedded objects are bound to be requested at a later stage. The pre-fetching of the embedded objects reduces the wait at the end user and accelerates the browsing experience. The bundles can be easily identified by a cursory analysis of the web server log files. The user browser places subsequent requests to the embedded objects after the acknowledgement of the initial page is received.

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 is new. 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.

We plan to use the following policies as benchmark to compare the final results: Weighted Round Robin (WRR), Locality Aware Request Distribution (LARD), Power-Aware Request Distribution (PARD) and techniques proposed earlier for HTTP 1.1 (Ext-LARD-PHTTP) [5].

III. POWER AND LOCALITY-AWARE REQUEST DISTRIBUTOR

This section describes the conception of our scheme; combining power and locality-aware request distribution schemes. The basic PLARD system consists of a simple power aware distribution policy built over the LARD policy. The distributor forwards the requests to one of the backend server that is ON, based on the locality of the requested data. While the distributor transmits the requests to the backend servers, the power policy checks if there are any idle backend servers. If it happens that some of the backend servers are idle, the idle backend servers will be turned OFF for power conservation. Similarly, when the incoming request rate is high and the total load on the system increases, new servers are turned ON to compensate the increase in the incoming requests. During this turn ON phase, the requests that are scheduled to be serviced by the turning ON server, incur the startup delay of the server. This simple ON-OFF policy achieves very good power conservation, but only at the cost of performance of the system. In the consequent sections, we describe our enhanced power policy, which counters this performance degradation, and we present our algorithm for both locality-based request distribution and power conservation. In this section we will discuss about the simple power and locality aware request distribution presenting its advantages and shortcomings.

1. Combining power policy and LARD

In this sub-section, we describe the effects of coupling the simple ON-OFF power policy and the locality-aware request distribution scheme. Both these schemes have been individually proposed and implemented in [3] and [1] respectively. The background on these schemes has been presented earlier in section II. The motivation to couple these policies together is derived because of the unavailability of a distribution scheme that provides both power conservation and better performance. This scheme which consists of simple combination of the two policies [1] [3] will be referred to as “PLARD” – power and locality-aware request distribution - in the rest of the thesis.

The PLARD consists of an implementation of the LARD policy [1] which is in turn controlled by the simple ON-OFF power policy. The distribution of the requests follows the LARD policy and thus maximizing the memory hits on the backend servers’ memory and improving the system performance. The power policy sits at the front-end distributor and monitors the load on the backend servers. Servers operating during non-peak hours typically are very lightly loaded and hence are good candidates for being turned OFF and conserving power. On identifying a server with load below a certain threshold, the front-end stops distributing and allotting requests to ‘that’ server and initiates its shutdown.

Similarly, on the event of increasing system load, the servers in OFF state and turned ON to share the load and decrease the system delay. The servers introduce a startup delay of typically 45 seconds to up to 2 minutes. This delay is incurred by all the requests that are scheduled for the server in this transition phase and contributes directly to the delay experienced by the end user. This scheme is effective in power conservation, but this is obtained at the expense of performance degradation.

Fig. 6 illustrates the comparison of the results obtained by comparing PLARD with WRR, LARD and PARD. The algorithms are implemented in a C++ based simulator which is driven by trace files derived from real web servers. The same simulator and trace files are used throughout the thesis to present results and comparisons. The details of the simulation model and the trace files are provided in section VI.

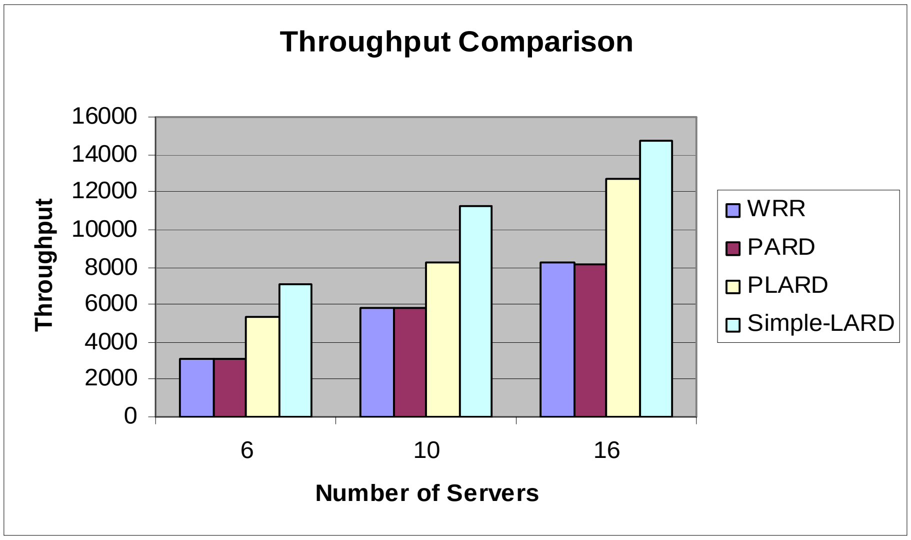


Fig. 6. Throughput comparison (PLARD)

From the above figure, it can be seen that PLARD suffers from performance degradation due to the trade-off incurred in improving the power conservation. The performance loss is largely due to the startup delay of the servers when they are turned ON to service the increasing request flow. Also, it is to be noted that both the simple-LARD and the LARD policy implemented in our scheme support only HTTP 1.0 based connections. By adopting a policy suitable for the prevailing HTTP 1.1 based connections, we can generalize the idea and also exploit the new prospects through the persistent connections.

Also, by applying intuitive analysis and making the distribution and power policy more proactive, we can avoid such problems and improve the performance of the system. Such schemes are described in the following sections.

IV. ENHANCEMENTS IN LOCALITY-AWARE REQUEST DISTRIBUTION

Considering the shortcomings of the PLARD scheme, we focus to improve the performance of the system in spite of providing considerable power conservation. The performance can be improved by enhancing the locality-aware request distribution scheme. We provide a LARD scheme which is compatible with persistent HTTP connections and describe methods to improve its performance.

1. Handling persistent HTTP connections

In a connection between a client and the server, the header of the “first” incoming request is read by the distributor, the lookup on the distributor table is made to determine the availability of the file in the backend server’s memory and then, the connection is handed-off to the backend server. Since, the connections are over HTTP 1.1, most of the following requests from the same client, arrive on the same connection. Usually, the user request for the file and the embedded objects are requested sequentially over the connection. The distribution policy is set to forward the requests from one client to one particular server, based on the analysis of the website log files. This minimizes the overhead of dispatching each of the requests individually to the backend servers. The locality of the requests that are being forwarded to the backend server is increased through pre-fetching and data placement at the backend server. On analyzing the real-time web log files of most websites, we observe that the client requests follow a predictable pattern and this information can be used to pre-fetch the next plausible request page into the memory. Thus, when a request for a page A is made over the persistent-HTTP connection, the related web pages (A1, A2, and A3) are pre-fetched to the pinned memory on the server handling client requesting page A. The backend server then replies back to the requesting client with the requested files. This is illustrated in Fig. 7.

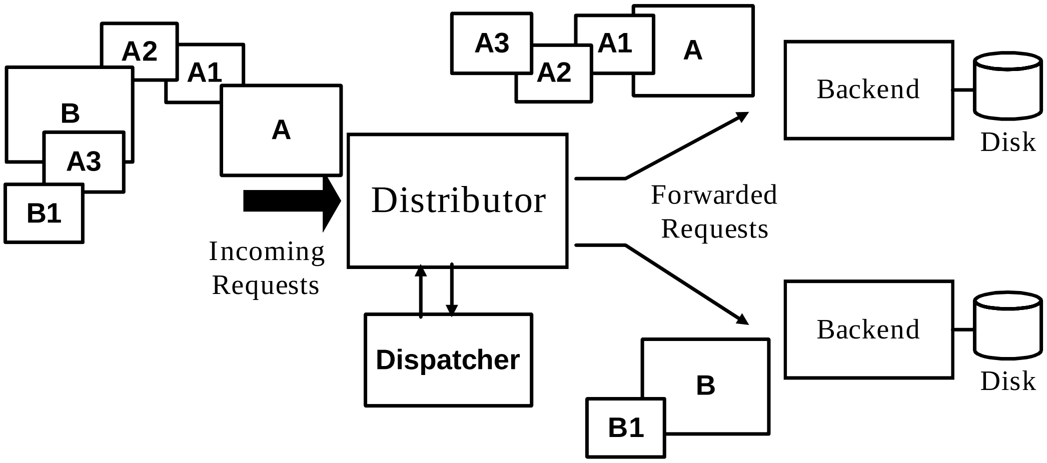


Fig. 7. Illustration of the distribution policy

It is to be noted that during the initial phase, after the start of the server, the requests need to be retrieved from the disk. As the pinned memory of the backend servers gets populated, the locality of the files is updated on the distributor. The initial choice of the backend server depends on the lookup at the distributor table and the current load conditions at the server.

2. Memory management scheme

Considering a system with simple locality-based distribution, integrated with the power policy, the system suffers from considerable performance loss (15% - from simulation results). LARD policies perform well due to the locality information preserved within the cluster system. When servers are turned OFF for power conservation, we lose all the information in the volatile memory and also the locality of the files. In order to overcome this drawback, a better memory management scheme should be in place, that improves the locality of the files and hence the performance of the system.

Pinned memory can be construed as a memory area, which is a part of the user memory. The pinned memory does not overlap with the rest of the memory area and is solely accessed by the application on the server (web server or file server application). It can be categorized to be in between cache and memory in memory hierarchy. With this arrangement, the application running on the server (file or web server) can be allotted maximum available memory and the locality of the files can be increased (when the locality of the files is improved, more memory hits can be initiated and the response time can be decreased). Note that, with the allocation of the pinned memory, no additional memory space is being consumed. The available memory is being effectively partitioned for the application. Files, which are accessed frequently, are located in the pinned memory. When the locality algorithm makes use of the pinned memory, it reduces the disk accesses required by the application process due to the improved locality of the files. One other feature of the pinned memory is that, the contents of the pinned memory can be migrated to another server, when the server is being turned OFF. Thus, their position in the memory and hence their locality being preserved. The organization of the pinned memory is illustrated in Fig. 8.

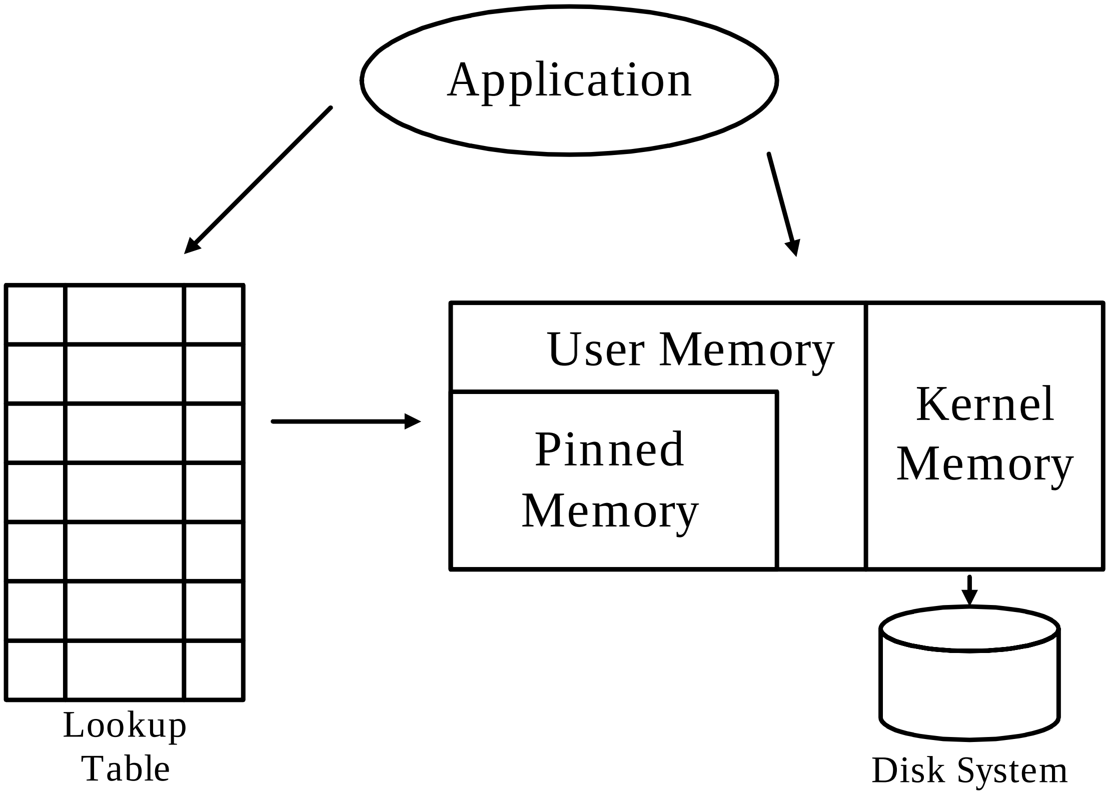


Fig. 8. Pinned memory organization

2.1 Data placement

The pin-down memory is fully associative in data placement. It is fully controlled by the user/application process, and it maps the cached file into the address of memory. The user/application process decides the size of the pin-down memory. The file is placed in blocks on the pin-down memory. If a file cannot be fully accommodated in the memory, it cannot be placed and it has to be had in the cache of the server. Fig. 4 illustrates the organization of pin-down memory and the mode of access of pin-down memory by the user/application process. User/application process obtains the information about the position of the data in the pin-down memory by looking up the table. *Block no, file name* and *link* are the important fields in the lookup table. '*Block no*' stands for the number of blocks for that particular file, '*file name*' is the name of the block to be located in the '*block no*' column and '*link*' holds the pointer to the next consecutive block.

2.2 Data replacement

The popularity of the file is used to place/replace files in the pin-down memory. The pin-down memory occupies fraction of the main memory and so, utilizing this small memory area effectively, becomes very important. Two facts decide the importance of the cached file: frequency of access f_{file} and size of file S_{file}. Frequency of access relates to the number of times the file is accessed on the cluster system. Thus, it becomes necessary to place/replace the important files in the pin-down memory. Additionally, the size of the files S_{file} can be critical in deciding the utilization of the pin-down memory. Loading several smaller and most sought for files is considered more critical than one large file. The importance of a file I_{file}, which can be denoted as f_{file} / S_{file}, decides its position in the pin-down memory. Based on the value of I_{file}, the file is placed in the pin-down memory or is kicked out from the pin-down memory. For every request, the I_{file} is calculated at the backend to reorganize the pin-down memory. Once, the file is placed/replaced in the pin-down memory area, this information is sent to the front-end distributor. Front-end server uses this information about the position of the file and forwards the incoming requests.

2.3 Migration in pin-down memory

Once a server is turned OFF, the user and kernel memories are wiped off. In conventional systems, this decreases the locality of the files and increases the disk accesses. In our policy, once a server is about to be turned OFF, the contents of the pin-down memory are migrated to other servers, which are capable of accommodating the data. This information is updated at the front-end and the distribution is still capable of generating hits at the backend servers, which would otherwise have ended up as disk accesses.

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 data. 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. We use a simple Perl-based script to analyze the log files.

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. Towards the end of Section 3, we illustrate and generalize the ability of categorize the website users on most of the common website types.

3.2 Popularity of 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.

3.3 Spotting bundles

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 the next section.

3.4 Examples illustrating user categorization

Table 1 illustrates the possible categorization of popular website types. The argument is that the users visiting most of the websites can be categorized into well know user groups and use this information for enhancing the locality of the files.

Table 1. Categorization of websites

Website Type	**Category 1**	**Category 2**	**Category 3**	**Category 4**
Univesity	Prospective Students	Current Students	Faculty/Staff	Recruiter/Industry Personnel
News	Politics	Technology	Sports	Business
Technology	Software	Hardware	Security	Networking
Sports	Country1	Country2	Country2	Country4

4. Application of web log information to improve LARD

The primary purpose of the web log mining is to enhance the distribution policy at the front-end. The front-end forwards the requests to the backend server nodes based on the locality of the data in the backend servers' memory. We use the web log mining information to make the distribution proactive and provide effective pre-fetching at the backend server nodes. The following sub-sections detail the exact application of the web log information in the context of backend server nodes and the front-end.

4.1 At the backend

As explained earlier, the users visiting the website are categorized into the specific groups using our web log mining script. The requests from a particular user can be monitored and identified to be belonging 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. Longer the comparison paths, better is the confidence of the predicted category.

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. This mechanism is clearly illustrated on the next page in Figure 9.

The algorithm to perform this compare and pre-fetch approach is illustrated in Fig. 10. For every incoming request, a comparison is made with the corresponding branch of the website. As an example, let us assume that the user is visiting a portal and is interested in technology (refer to Fig. 9). More particularly in information related to "windows.html."

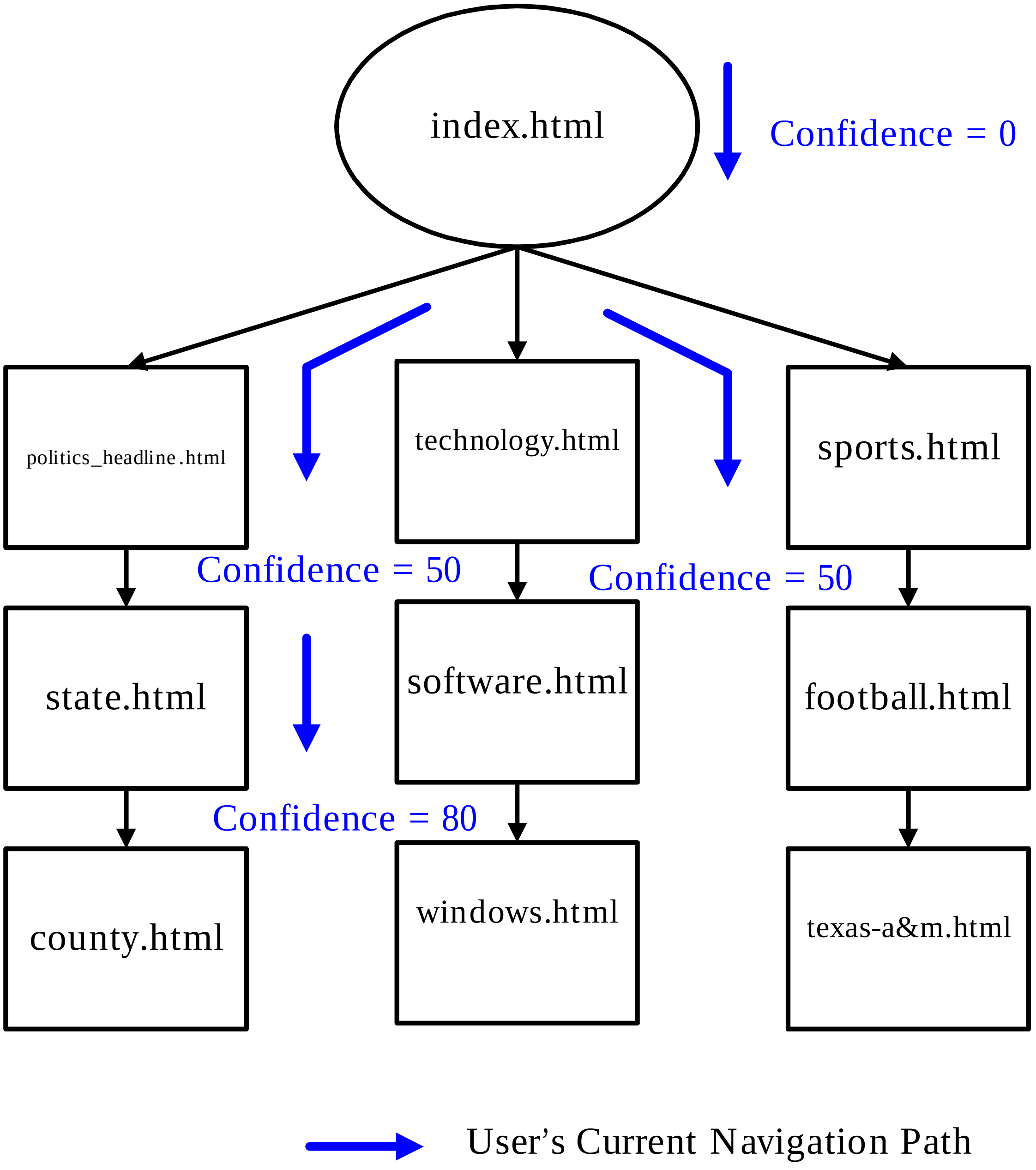
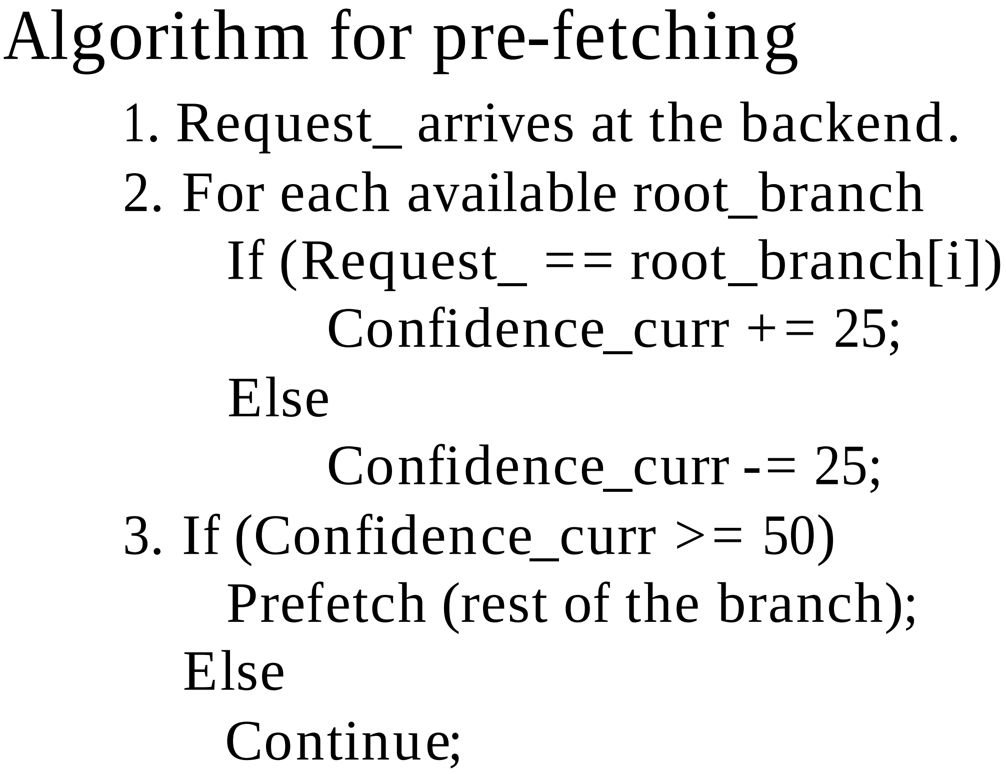


Fig. 9. Building the confidence of the guesses

When the user requests to view “technology.html,” the request arriving at the backend server would be “root/technology.html” and the corresponding comparison branch would be “root/technology.html,” which are the same. This increments the confidence for the current session (Confidence_curr) to 25. This means that, our prediction on the user’s browsing behavior is getting better and vice versa.



```
Algorithm for pre-fetching
    1. Request_ arrives at the backend.
    2. For each available root_branch
           If (Request_ == root_branch[i])
               Confidence_curr += 25;
           Else
               Confidence_curr -= 25;
    3. If (Confidence_curr >= 50)
           Prefetch (rest of the branch);
       Else
           Continue;
```

Fig. 10. Algorithm for pre-fetching

Once the confidence value for the current session increases 50, we ascertain that the user is more interested in the data belonging to the branch below his current navigation position. This is a vital information used for pre-fetching the data in the underlying branches (referred as “rest of the branch” in Fig. 10). Such a pre-fetching will reduce the number of disk accesses which increases the response time of the system. The algorithm is simplified here for the clarity of the concept. The parameters are fine tuned in the actual implementation to achieve maximum performance.

Also, when a request for a data page arrives at the backend, the embedded objects associated with that page are pre-fetched into the cache. The subsequent requests from the client 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, larger the replication of these pages on the backend servers’ memory. Fig. 11 (next page) illustrates the replication algorithm.

Replication algorithm

```
1. 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 T1½ & T1¾ )
                   Replicate(rank_table[i].file, all¾ );
               Else if (rank_table[i].rank is btw T1¼ & T1½ )
                   Replicate(rank_table[i].file, all½ );
               Else if(rank_table[i].rank is btw T1/8 & T1¼ )
                   Replicate(rank_table[i].file, NO_CHANGE);
               Else
                   Replicate(rank_table[i].file, NONE);
       (iii) Return to step 1.
```

Fig. 11. Replication algorithm

The replication algorithm is set to run centrally on all the whole cluster system. The interval of operation (‘t’ seconds) is decided based the current operating conditions of the system (load, service time, etc) or a fixed interval of 15 minutes, whichever is earlier. A rank table is built (rank_table) based on the number of hits registered for each of the data pages (through dynamic log mining of the recent history). Each one of the files have 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. The replication is carried out by the Replicate( ) function. The attributes ‘all’, ‘NO_CHANGE,’ and ‘NONE’ are self explanatory.

4.2 At the front-end

The mechanism described at the backend will ensure that the backend servers' cache is populated with the files that are being requested by the clients and also with files that might be requested by the clients in the near future. By default, the front-end would forward the requests for the embedded objects to the same server where it had dispatched the request for the associated web page. In addition, it forwards the regular requests following the LARD policy. It is to be noted that the data in the backend servers' memory is not the same as it would have been with a simple LARD policy. The dataset has been refined using the web log mining information.

V. ENHANCEMENTS IN POWER POLICY

The power policy resides over the locality aware request distribution policy. The request distribution is carried out as explained in the previous section. It uses the web log mining information to improve the locality of the files in the backend servers' memory. The front-end and the backend keep track of the location of the files in the memory and have this information available for the request distribution algorithm. The request distribution is done for servers that are either ON or in hibernation. The servers that are turned OFF are not considered for request distribution.

1. Power policy

The power policy is a variation of the PARD policy proposed by Rajamani et al [3]. In PARD, the authors propose the use of two power states – ON and OFF. When the servers are turned OFF during non-peak hours, the memory is wiped out and the locality of the files is lost. Also, the requests that are forwarded to a server that is turned OFF and is in the transition state of being turned ON incur a startup delay which is reflected to end user. This can be detrimental to a website which has undulating load characteristics. We have enhanced the power policy to address these issues.

First, we introduce an intermediate power state of "Hibernation," in between the ON and OFF states. The intermediate state of "hibernation" is preferred over the "OFF" state for the following reasons: (i) the start-up delay for a turned OFF server varies from 60 to 90 seconds, whereas the system to turn ON from the hibernation requires only 10 to 15 seconds. (ii) Volatile content is completely lost when it is turned OFF, whereas the content is preserved when the server enters the hibernation state. (iii) Power consumption in the hibernation state is roughly 5%, compared to the active state, and is nearly as good as the turned OFF state. Fig. 12 gives the state transition diagram for the power policy.

Second, the “OFF” servers are turned “ON” proactively through anticipation and careful monitoring of the current system load. The web log mining information can give an idea of the general load characteristics of the cluster web server system and this characterization can be used to anticipate the website load. The distribution of the requests in the load characterization of most of the web servers is never Gaussian and this element of predictable behavior of the user requests can be used to anticipate changes in the incoming request flow. Thus proactive switching improves the response of the server by nullifying the start-up delay which will otherwise worsen the QoS.

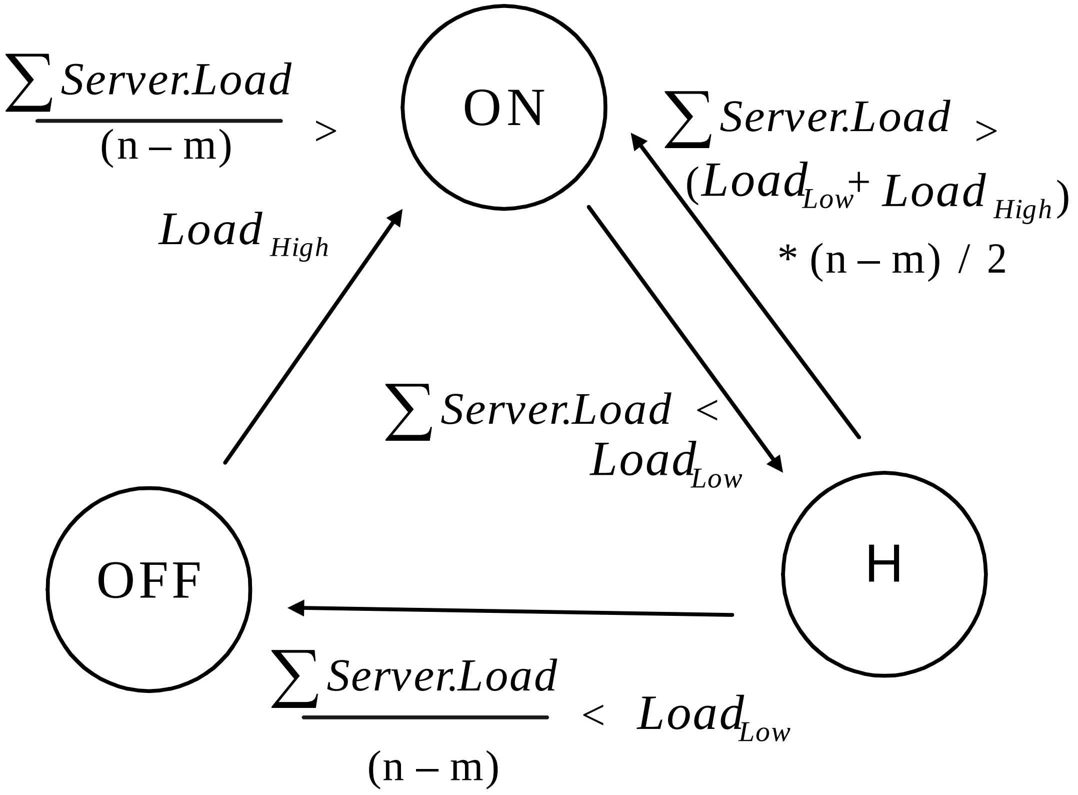

ON
OFF
H
$\frac{\sum Server.Load}{(n-m)} > Load_{High}$
$\sum Server.Load > (Load_{Low} + Load_{High}) * (n-m) / 2$
$\sum Server.Load < Load_{Low}$
$\frac{\sum Server.Load}{(n-m)} < Load_{Low}$

Fig. 12. Power transition states: H- Hibernation mode

The power policy is applied based on the load thresholds of each server. If the load of a server drops below $Load_{low}$ (the minimum load required on a server below which the server could become underutilized), the distributor is notified to stop scheduling requests to that server; once the load on the system becomes zero, the server is put into hibernation. If the total load of the system increases above the threshold [(n-m)* 2 ($Load_{low}$ +$Load_{high}$)] ($Load_{high}$ is the maximum load that can be applied on a server without any performance degradation) the server is woken up from the hibernation state. A server is turned OFF if ΣServer.load (total load of the system) goes below $Load_{low}$ *(n-m) and it is woken up from OFF state if the total load of the system exceeds $\text{Load}_{\text{high}}$ *(n-m). Based on this simple policy, the servers are manipulated for power conservation. Wake-on LAN system, where the servers can be issued commands (for turning OFF/ON/Hibernation) from a remote machine via the LAN, is used for forcing a server to enter the desired state.

VI. SIMULATION MODEL AND RESULTS

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 Figure 13.

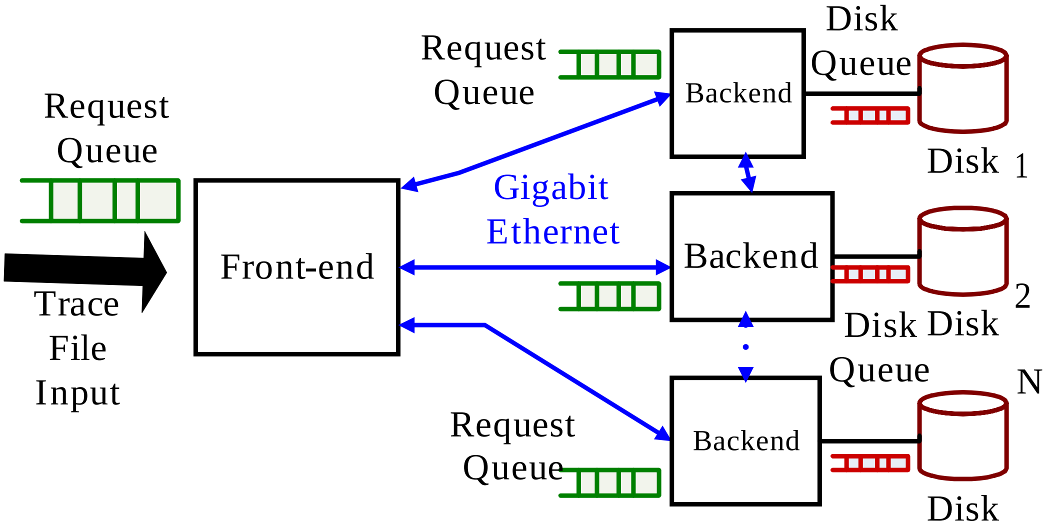


Fig. 13. Simulation model

The simulation system parameters are enumerated in Table 2.

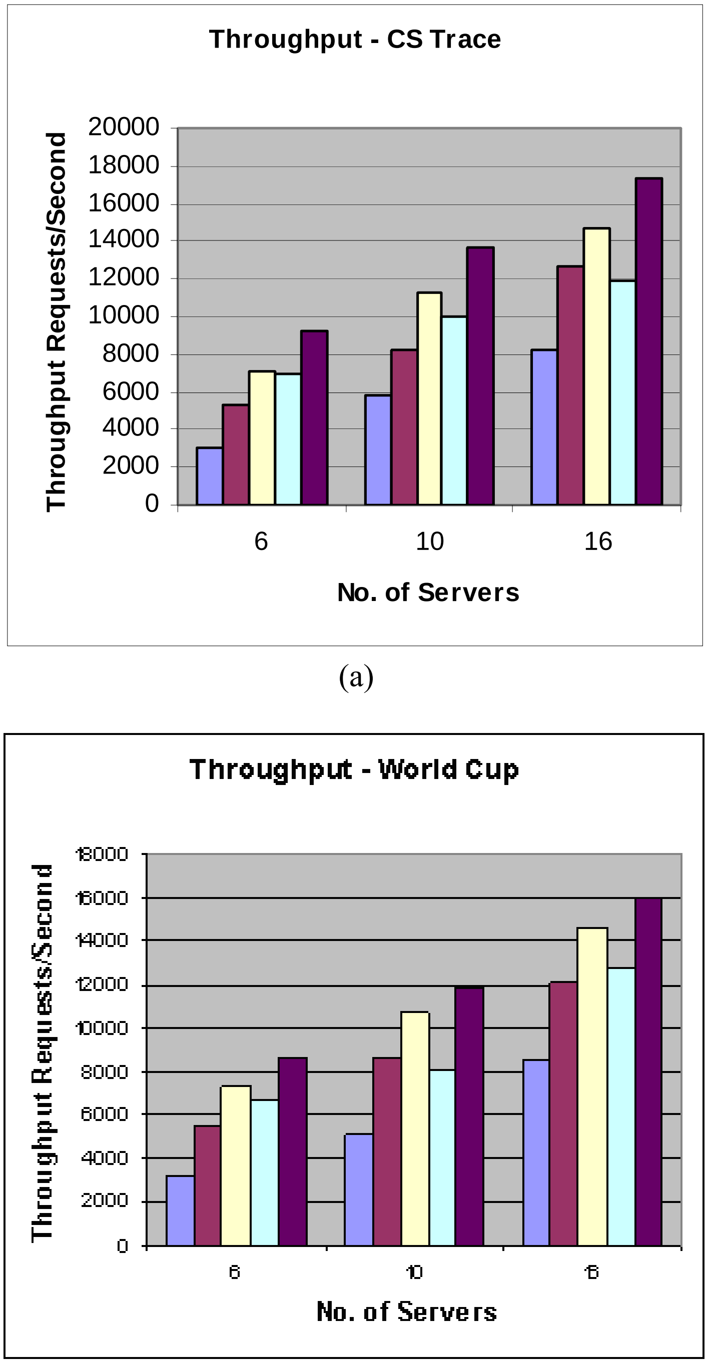
Table 2. Simulation system parameters

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

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. Fig. 13 illustrates the simulation model. 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 are compared. In the second section, the following metrics are closely monitored for evaluating the performance of the system: Throughput and Power conservation. We compare our policy (PLARD-web-log) against WRR, LARD, PLARD, and Ext-LARD-PHTTP.

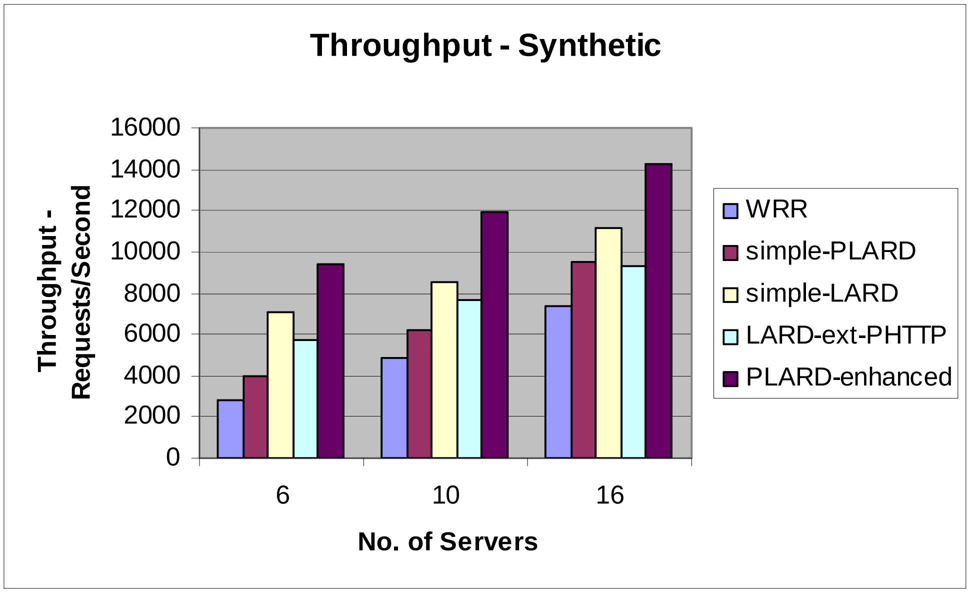
The throughput of all the algorithms for each of the trace is compared in Fig. 14. The throughput is the summation of the number of requests processed by each of the backend servers . Our scheme performs slightly better than the LARD system with a marginal improvement of 5-10 %. The improvement in both LARD and PLARD-enhanced 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 yields 85% hit rates with LARD and 5-10% boost with our scheme.



(a)

(b)

Fig. 14. Throughput comparison: (a) – CS Trace, (b) – World Cup, (c) - Synthetic



(c)

Fig. 14. Continued

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. 15 (next page).

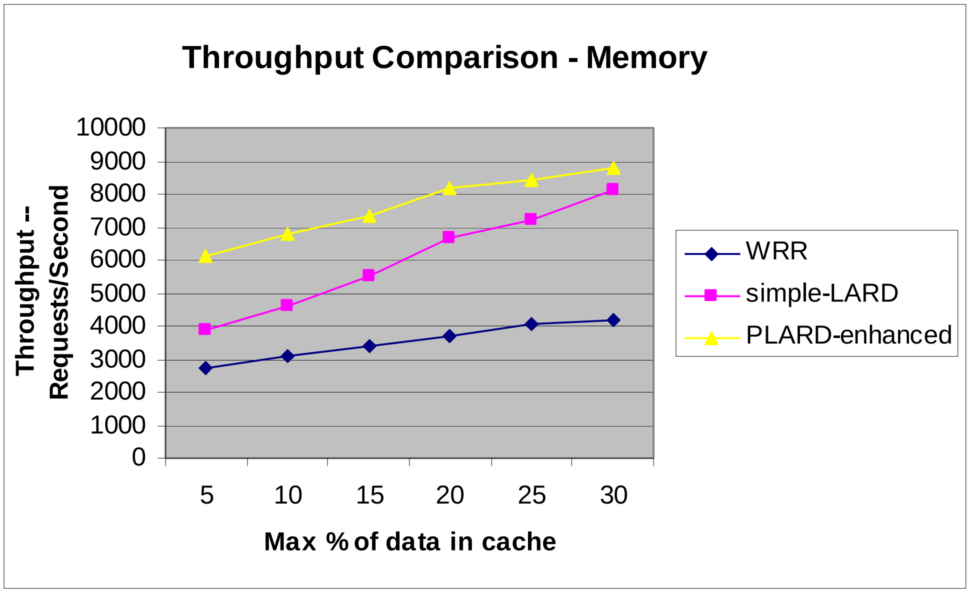


Fig. 15. Comparison with LARD

This illustration shows PLARD-enhanced is more consistent in preserving the locality of the files than LARD. This comparison has been necessary to portray the efficiency of PLARD-enhanced. The scenarios depicted here can be a possibility with large websites with large data contents, where 30% of the website data cannot be accommodated in the cache.

As we explained earlier, PLARD-enhanced consists of the enhancements outlined in section IV 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 enhancments. Figure 16 illustrates the throughput comparison of each of the enhancment schemes. PLARD-bundles denotes the bundle-based distribution scheme. PLARD-distribution stands for the improvement achieved through the dynamic distribution of the files on the backend servers' memory based on their popularity. Finally, PLARD-prefetch-nav denotes the enhancement achieved through proactive prefetching in the backend servers' memory through web log mining. It can be seen that prefetching complemented by web log mining provides the best improvement clearly outperforming the other schemes by 100%. Also, PLARD-enhanced-overall is the combination of these schemes and performs better as the schemes are complementary among themselves.

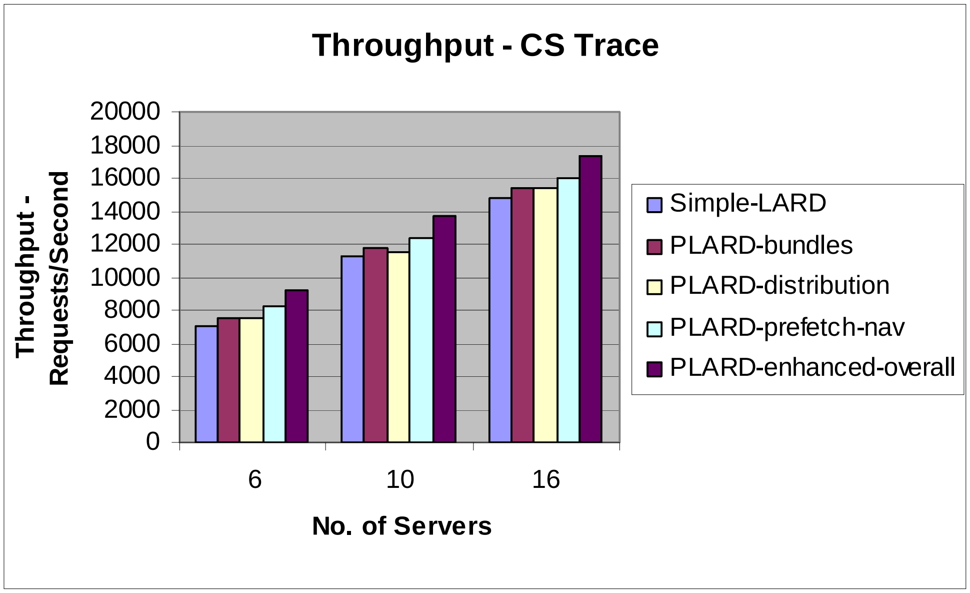


Fig. 16. Throughput comparison for individual enhancement

Fig. 17 shows the comparison of power conservation among PLARD-enhanced and simple-PARD [3]. All the other policies lack power conservation and are not considered in the evaluation. PARD and simple-PLARD achieve good power conservation in comparison to PLARD-enhanced as they have more aggressive power conservation strategy. They simply trade performance for the gain of power conservation. In contrast, PLARD-enhanced focuses to provide good performance and the power conservation is a plus.

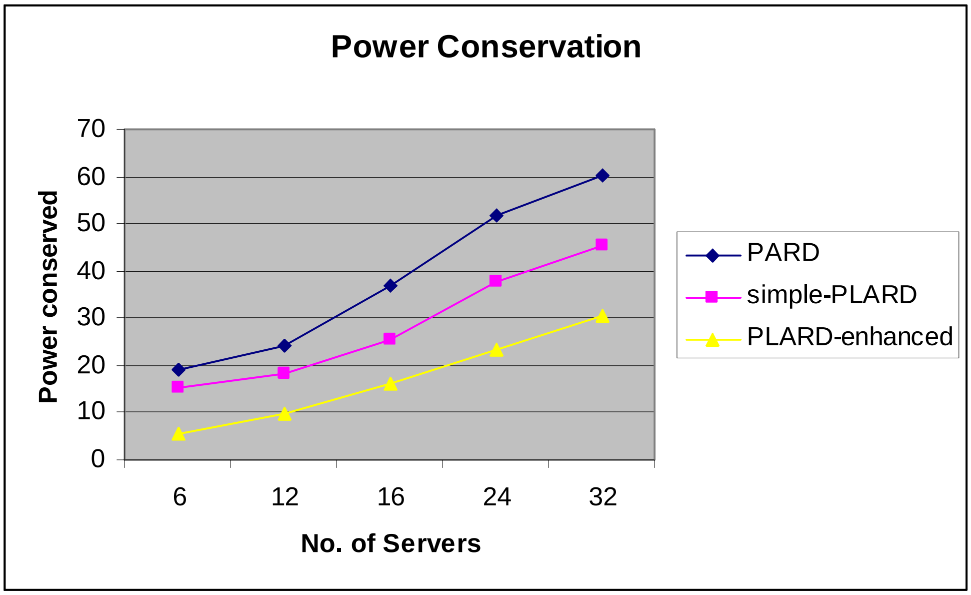


Fig. 17. Power conservation

VII. SUMMARY AND CONCLUSIONS

As the use of cluster systems increases, conserving power and improving performance have been a critical issue. In this thesis, we compare four different policies: WRR, LARD, Ext-LARD-PHTTP and PLARD-enhanced to determine the policy that provides best results in terms of power and efficiency. 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 is used. However, WRR and LARD save zero power in the cluster system. Thus, we propose PLARD that employs not only content-based request distribution, but also On-Off policy to have good efficiency and power conservation. But, the power conservation comes at the cost of performance of the system. To overcome this, we propose a modification in the memory organization and its usage in the context of the application server. Also, our policy provides support to HTTP 1.1 based connections through proactive distribution and pre-fetching. The simulation results indicate that our system provides considerable power conservation (5 – 30%) in spite of improved performance (15 -40%). As a future extension, we can explore the possibility of providing support for dynamic content.

REFERENCES

[1] 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 *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 2-7, 1998, pp. 205-216.

[2] APC – American Power Conversion. “Determining total cost of ownership for data center and network room infrastructure.” ftp://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf, December 2003.

[3] K. Rajamani and C. Lefurgy, “On evaluating request-distribution schemes for saving energy in server clusters,” in *Proceedings of International 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 *Proceedings of the USENIX 2000 Annual Technical Conference*, San Diego, CA, June 2000, pp. 323-336.

[5] M. Aron, P. Druschel and W. Zwaenepoel, “Efficient Support for P-HTTP in Cluster-Based Web Servers,” in *Proceedings of the Annual USENIX Technical Conference*, Monterey, CA, 1999, pp. 185-198.

[6] E. V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," in *Proceedings of the 17th Annual International Conference on Supercomputing*, San Francisco, CA, June 2003, pp. 86-97.

[7] E. Pinheiro, R. Bianchini, E. V. Carrera and T. Heath, "Dynamic Cluster Reconfiguration for Power and Performance," Norwell, MA: Kluwer Academic Publishers, 2002.

[8] S. Ramakrishnan and Y. Yinghui, "Mining Web Logs to Improve Website Organization," in *Proceedings of WWW-10*, Hong Kong, May 2001, pp. 430-437.

[9] C. E. Wills, G. Trott and M. Mikhailov, "Using Bundles for Web Content Delivery," in *Proceedings of ACM Computer Networks*, New York, NY, August 2003, pp. 797-817.

[10] P. Mike and E. Oren, "Adaptive Web Sites: Automatically Synthesizing Web Pages," in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, Madison, Wisconsin, 1998, pp. 727-732.

[11] P. Mike and E. Oren, "Towards Adaptive Web Sites: Conceptual Framework and Case Study," in *Proceedings of WWW-8*, Toronto, Canada, May 1999, pp. 152-158.

[12] N. Takehiro, K. Hiroki, Y. Yohei, "Discovering the Gap Between Website Designers' Expectations and Users' Behavior," in *Proceedings of WWW-9*, Amsterdam, May 2000, pp. 811-822.

[13] S. Myra and F. C. Lukas, “WUM: A Web Utilization Miner,” in *Proceedings of EDBT Workshop WebDB98*, Valencia, Spain, March 1998, LNCS 1590.

[14] S. Myra, F. C. Lukas and W. Karsten, “A DataMiner Analyzing the Navigational Behaviour of Web Users,” in *Workshop on Machine Learning in User Modeling*, Chania, Greece, June 1999, ACAI’99.

VITA

Gopinath Vageesan

72 Krishnaswamy nagar, 2nd layout,

Ramanathapuram, Coimbatore,

Tamilnadu, India – 641045.

Educational Background

Gopinath Vageesan received his B.E. in electronics and communication engineering from Tamilnadu College of Engineering, Bharathiar University, Coimbatore, India in May 2002. He earned his M.S. in computer engineering from the Department of Electrical and Computer Engineering, Texas A&M University.