Adaptive Prefetching Scheme Using Web Log Mining in Cluster-based Web Systems

Heung Ki Lee, Baik Song An, and Eun Jung Kim
Department of Computer Science and Engineering
Texas A&M University
TAMU 3112 College Station, TX, USA
{hklee, baiksong, ejkim}@cs.tamu.edu

Abstract

The main memory management has been a critical issue to provide high performance in web cluster systems. To overcome the speed gap between processors and disks, many prefetch schemes have been proposed as memory management in web cluster systems. However, inefficient prefetch schemes can degrade the performance of the web cluster system. Dynamic access patterns due to the web cache mechanism in proxy servers increase mispredictions to waste the I/O bandwidth and available memory. Too aggressive prefetch schemes incur the shortage of available memory and performance degradation. Furthermore, modern web frameworks including persistent HTTP make the problem more challenging by reducing the available memory space with multiple connections from a client and web processes management in a prefork mode. Therefore, we attempt to design an adaptive web prefetch scheme by predicting memory status more accurately and dynamically.

First, we design Double Prediction-by-Partial-Match Scheme (DPS) that can be adapted to the modern web framework. Second, we propose Adaptive Rate Controller (ARC) to determine the prefetch rate depending on the memory status dynamically. Finally, we suggest Memory Aware Request Distribution (MARD) that distributes requests based on the available web processes and memory. For evaluating the prefetch gain in a server node, we implement an Apache module in Linux. In addition, we build a simulator for verifying our scheme with cluster environments. Simulation results show 10% performance improvement on average in various workloads.

1. Introduction

Cluster systems have been widely accepted as a cost effective solution for various applications such as web services and file/database management. However, web clients experience long and unpredictable delays when retrieving web pages from the cluster systems. To solve the delay problem, there have been many studies [22, 18, 29, 37] on the main memory management such as web cache schemes and web prefetch schemes. Although web cache schemes reduce the network and I/O bandwidth consumption, they still suffer from a low hit rate, stale data and inefficient resource management. [2] shows that an inefficient web cache management caused a major news web site crash, also called *the Slashdot effect*.

Web prefetch schemes overcome the limitation of web cache mechanisms through pre-processing contents before a user request comes. Web prefetch schemes expect future requests through web log file analysis and prepare the expected requests before receiving it. Compared with web cache schemes, web prefetch schemes focus on the spatial locality of objects when current requests are related with previous requests. Web prefetch schemes increase the bandwidth utilization and reduce or hide the latency due to bottleneck at web server. However, despite these benefits, three difficulties prevent prefetch schemes from being exploited in web cluster systems. First, it is difficult to find which objects are related with the incoming requests. At the server side, web access patterns are dynamic because of the web cache mechanism. Second, it is difficult to find an optimal prefetch rate. Too aggressive prefetch schemes may hurt overall performance due to the shortage of memory. Finally, a prefetch scheme in a web cluster system should be considered along with an efficient resource management. The inappropriate resource management drains the resource of one backend server, while other backend servers have the available resource.

To overcome these difficulties, we propose an adaptive web prefetch scheme. To the best of our knowledge, our scheme is the first attempt to provide an adaptive prefetch scheme in web cluster environments to support the modern

web framework. Our prefetch scheme decides which web objects are to be prefetched by considering memory status of the web cluster system. Our adaptive scheme consists of three components; Double Prediction-by-Partial-Match Scheme (*DPS*), Adaptive Rate Controller (*ARC*) and Memory Aware Request Distribution (*MARD*).

First, we propose a dynamic web prediction scheme called Double Prediction-by-Partial-Match Scheme (*DPS*). Web access patterns are dynamic depending on the location of a client. When web objects are stored in an intermediate node, requests to those cached objects do not reach the web server. The *DPS* scheme solves the problem by providing the adaptiveness that handles the client's random access pattern.

Second, we suggest Adaptive Rate Controller (*ARC*) that provides an adaptive prefetch rate at run time. There is a trade-off between consuming memory space and the performance of a web cluster system in modern web frameworks. In multiprocessing environments, web processes allocate memory by their needs. However, we cannot provide the system with unlimited memory, so aggressive prefetch schemes can interfere with demand requests from the same client or other clients. For improving the performance of prefetch schemes, the *ARC* scheme prefetches web objects depending on the memory status.

Our last contribution is Memory Aware Request Distribution (*MARD*) which distributes incoming requests to the prefetch-enabled backend servers efficiently. Locality-based distribution is commonly accepted to improve the performance using the locality of incoming requests. However, non-uniform distribution can use up the memory at the selected backend server and an aggressive prefetch scheme also consumes the memory for prefetching useless objects at the selected server. It can cause the delay in the overall web cluster system. *MARD* avoids the skewed distribution of requests at the web cluster system.

In this paper, we design a prefetch scheme for cluster environments that is well adapted to the modern web framework and workloads. In Section 2, we discuss the existing prefetch technologies in detail. Section 3 describes our motivation in prefetch at the web cluster system. Section 4 explains the detailed design of the proposed prefetch scheme. Section 5 shows the simulation model and results, while Section 6 concludes the paper.

2. Related Work

Web prefetch schemes are roughly classified into two groups; short-term and long-term prefetches. Short-term prefetch schemes predict future requests based on the recent history information. [26] proposes the prefetch algorithm for a general file system. [5, 13] predict the next incoming requests using the N-th order Markov model. [13] suggests heuristic schemes to reduce high complexity at a multi-level Markov model.

The Prediction-by-Partial-Match (P.P.M.) model [9, 12, 6, 15] complements N-th Markov models. Order of Markov model increases not only the accuracy but also the complexity at the same time. The P.P.M. schemes predict the future incoming requests based on the top-n schemes [12, 6] or a confident threshold [25, 9]. In [12], they search the long sequences for frequently accessed patterns. [4] suggests dynamic P.P.M. models. The recent research evaluates the latency of existing prediction schemes [15, 14].

The long-term prefetch scheme defines clusters of web objects using access pattern, then prefetches web objects in the unit of the cluster. [10, 28] provide replacement policies for Content Distribution Network (CDN) platform, while [35, 16] suggest the replacement algorithm for mobile environments. [17] provides a hierarchical clustering system that groups search results into several folders. In [11], a divide-and-merge scheme creates the cluster of web objects by combinational approaches between top-down and bottom-up schemes. [33] suggests the modified proxy model to prefetch the embedded objects from the web server. [24] provides a web cluster scheme using a vector model and semantic power.

Hybrid prefetch scheme integrates the short-term prefetch based on the Markov model and the long-term prefetch using cluster scheme. [20] suggests the combinational prediction scheme of existing models including Markov models, sequential association rules, association rules and cluster schemes. [23] generates *Significant Usage Patterns* based on abstraction techniques and also provides the path between *Significant Usage Patterns* using Markov model. [38, 8, 19] create the cluster based on their policy including *Expectation-Maximization* [8], *CitationCluster* [38] or K-means cluster scheme [19]. After generating the cluster, they use the Markov model to find the relationship between clusters of web objects. Although hybrid scheme provides the benefit of both short-term and long-term schemes, it does not support the memory-consuming modern web framework.

[21] provides Non-interfering Prefetch System (NPS) that manages the web prefetch rate based on the response time. [36] considers web prefetch in cluster environments. However, both of them are not proper to find the adaptive prefetch rate.

3. Motivation

Many studies [25, 10, 15, 14, 4, 28, 33] have been carried out to design effective web prefetch mechanisms. However, existing prefetch schemes fail to provide a solution that is suitable for modern web framework. They do not perform well in conjunction with persistent HTTP, web cache

scheme or request distribution policy in a cluster environment.

3.1. HTTP 1.1 Framework

Generally, an HTML file contains a number of embedded objects. Web clients request main HTML file to a server, then receive it. After analyzing the received main HTML file, web clients make the list of its embedded objects in the received main HTML file and request them to the web server. With HTTP 1.0, web client establishes new connection every time it requests an object. A client with HTTP 1.1 reuses a connection for multiple requests using persistent HTTP. In Figure 1, a web client with HTTP 1.0 has intervals between requests, while there is only one interval between the main object and its embedded objects with HTTP 1.1. These intervals between requests allow the web server to predict and prefetch the next requested web objects. In HTTP 1.0, there are intervals between consecutive requests that give the chance to prefetch the predicted embedded objects, while HTTP 1.1 makes it difficult to prefetch them. Web server with HTTP 1.1, therefore, prefetches embedded objects in the interval between the request to main object and the requests to embedded objects.

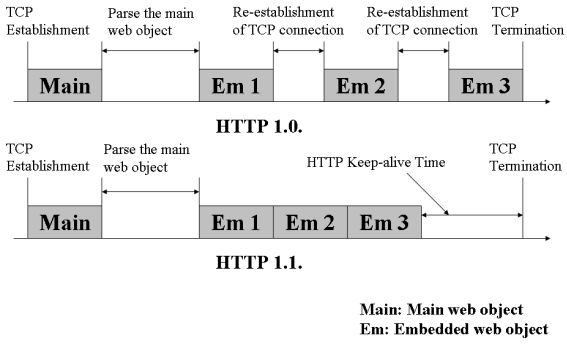

Figure 1. Persistent HTTP and Web Pipeline

3.2. Web Cache Scheme in Proxy Server

In general, web client requests the embedded objects after processing main object. However, we observe that not all embedded objects are requested after main object. Only a part of embedded objects have been requested after main object. Table 1 shows the request frequencies of 12 embedded objects after requesting *main.html* in Computer Science and Engineering at Texas A&M University. 5 embedded objects are frequently requested, while others have less than 100 requests.

One reason is that proxy servers provide some objects to the client directly if they are in the cache. The requests replied by proxy server will not be transferred to the web server. In addition, the configuration of web documents such as duration time and random objects can make the inequality of web access pattern. This random access pattern causes the misprediction. The misprediction does not only lose the chance to enhance the performance but also wastes the I/O and network bandwidth.

Table 1. The Frequency of Request to The Embedded Objects

Index	Name	Request
1	/html4/front.css	517
2	/html4/global.css	517
3	/images/bin.jpg	477
4	/images/header.jpg	473
5	/images/LOOK.gif	446
6	/images/random/01	75
7	/images/random/06	69
8	/images/random/07	66
9	/images/random/09	66
10	/images/random/010	62
11	/images/random/03	59
12	/images/random/04	58

3.3. Prefetch in Web Cluster

The distribution policy in a web cluster system plays a crucial role in system performance [27, 3, 30]. Especially, the locality-based request distribution schemes [27, 3] enhance the performance of web cluster systems through efficient memory management of backend servers. A web cluster handles multiple connections from the same client independently. Therefore, different backend servers in the web cluster reply to the same client. When a backend server prefetches web objects, it is not guaranteed that the next incoming request will be forwarded to the backend server which has the web objects through prefetch. To handle multiple connections, distributor should keep the prefetch information in the backend server of the web cluster system.

Moreover, the locality-based schemes with prefetching fail to achieve load balancing. The more files are prefetched at the backend servers, the more requests are forwarded to them. It results in consuming available memory space in the backend servers, and degrading the performance of the web cluster systems. A simply aggressive prefetch scheme in web cluster systems increases the inequality of request distribution, and drops performance dramatically.

4. Adaptive Web Prefetch Scheme in Web Cluster System

4.1. Double P.P.M. Scheme (DPS)

Although the previous work [19, 8, 38] provides hybrid schemes which lie between short-term and long-term approaches, they do not consider the modern web framework and cluster environments. At server side, access patterns are dynamic depending on the location of web clients and configuration of objects. The prefetch scheme at server side must be able to tolerate the randomness of access patterns.

We propose a two-level hybrid scheme, referred to as *DPS*. *DPS* handles main and embedded objects in different ways. Figure 2 shows how *DPS* finds the relationship between web objects from web traces. For the first step, *DPS* classifies web objects into a number of groups, which consists of one main object and its related embedded objects. During the grouping of web objects, *DPS* records the intra-section relationship defined by the request frequency of embedded objects after the request of the main object. *DPS* differentiates main objects from embedded objects according to the file extension. Web objects with file extension such as 'html', 'php' and 'jsp' are classified into main objects. In Figure 2, gray circles, white circles and arrows denote main objects, embedded objects and relationship, respectively. Each arrow has a prediction value, which shows the probability that the request of an embedded object will follow that of a main object. In Figure 2, *DPS* finds three groups including 'A', 'B' and 'C'. In the second step, it searches for the inter-section relationship between groups. *DPS* focuses on only the access to main objects that is representative of the group. In this step, *DPS* defines the access to main objects as the access to their corresponding groups. In Figure 2, *DPS* detects two access patterns to main objects including 'A $\rightarrow$ B $\rightarrow$ C' and 'A $\rightarrow$ C $\rightarrow$ B'. We can make the graph based on access patterns.

Although access patterns are changed by the web cache or configuration of web objects, *DPS* can find access patterns of objects. Intra-section relationship finds related embedded objects, while inter-section relationship can find the related main objects.

4.2. Adaptive Rate Controller

Using the relation information provided by *DPS*, Adaptive Rate Controller (*ARC*) calculates the prefetch rate dynamically. It determines which objects should be prefetched considering memory status. Memory is one of critical resources in web server system. Aggressive prefetching does not always guarantee the performance enhancement.

Figure 3 shows the response time over the variable hit rate of the prefetched data. We use 100 file groups for each

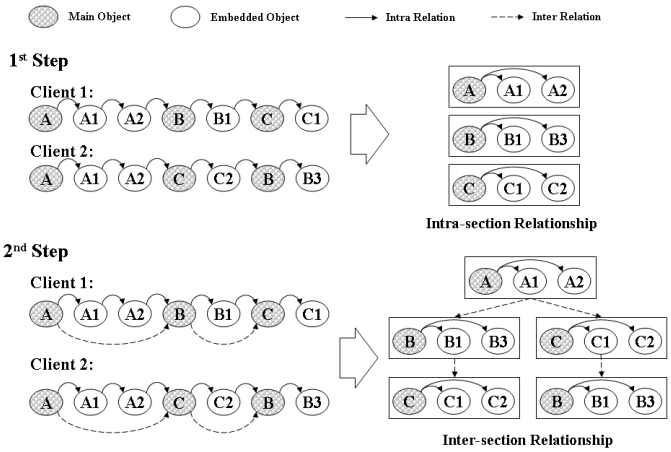

Figure 2. Double P.P.M. Scheme

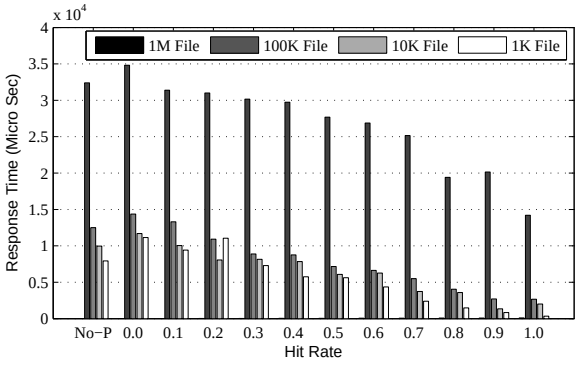

Figure 3. Prefetch Efficiency

file size; 1MB, 100KB, 10KB and 1KB. A file group includes 20 files of the same size. A client selects one group, and then generates requests to two files in the group at an interval of 2 seconds. After processing the first incoming request, the web server prefetches one file from the disk. The client creates the requests to the prefetched object or non-prefetched object according to the test configurations. With less than 10% hit rates, it takes longer than non-prefetch scheme. This result proves that the inefficient memory management degrades the performance in the web server system. It is a critical problem to figure out which files are prefetched in the memory.

We should formulate system improvements considering disk workloads and memory status. The following equation shows the access improvement.

$$P = RT - RT^{'}, \quad (1)$$

where RT and $RT^{'}$ are the average response time without prefetching the predicted object and the average response time with prefetching the predicted object, respec-

tively. *ARC* manages the prefetch memory for P in equation (1) to be positive. A web server is modeled by employing an M/G/1 round-robin queuing system where web server processes share one CPU and disk. Also, all cached pages in buffer cache have the same access probability in the future. First, the average response time (RT) in a web system is

$$RT = H_{sys} \times T_{mem} + (1 - H_{sys}) \times T_{disk}, \quad (2)$$

where H_{sys}, T_{mem} and T_{disk} are the hit rate of memory, memory transmission and disk response time, respectively. T_{disk} is the disk response time denoted by $\frac{DS}{1-\rho}$, where DS is the disk service time and ρ is the disk utilization. Disk utilization, ρ, is the miss rate of memory multiplied by the disk service time, DS. The miss rate of memory is $\lambda \times (1 - H_{sys})$, where λ is the request rate to a page.

When requests hit on memory, requested pages are located in the prefetch memory or the buffer cache. Thus, H_{sys} is $H_{pref} + H_{buf}$, where H_{pref} is the hit rate of prefetch memory and H_{buf} is the hit rate of buffer cache. We drive equation (2) to equation (3) as follows.

$$RT = (H_{pref} + H_{buf}) \times T_{mem} + F_{sys} \times \frac{DS}{\Delta}, \quad (3)$$

where F_{sys} is the miss rate of memory, the same as $1-H_{sys}$. Δ is $1 - \lambda \times F_{sys} \times DS$. H_{pref} and H_{buf} are hit rates of prefetch memory and buffer cache, respectively. When the requested page misses on memory, the response time becomes $\frac{DS}{\Delta}$, which is greater than zero. Also, DS and Δ are greater than zero.

$$\Delta = 1 - \lambda \times F_{sys} \times DS > 0 \quad (4)$$

When there is not enough space to prefetch related files, OS kicks out some cached objects in buffer cache to increase available memory space. It increases the hit rate of prefetch memory, while decreasing the hit rate of buffer cache. After prefetching related files, the average response time ($RT^{'}$) is defined as below.

$$RT^{'} = ({H_{pref}}^{'} + {H_{buf}}^{'}) \times T_{mem} + {F_{sys}}^{'} \times \frac{DS}{\Delta^{'}} \quad (5)$$

where ${H_{pref}}^{'}$, ${H_{buf}}^{'}$ and ${F_{sys}}^{'}$ are the hit rate of prefetch memory, the hit rate of buffer cache and the miss rate of memory, respectively. $\Delta^{'}$ is $1 - \lambda^{'} \times {F_{sys}}^{'} \times DS$, where $\lambda^{'}$ is the request rate to a page. When the requested page misses, the response time becomes $\frac{DS}{1-\lambda^{'} \times {F_{sys}}^{'} \times DS}$, which is greater than zero. Also, DS and $\Delta^{'}$ are greater than zero.

$$\Delta^{'} = 1 - \lambda^{'} \times {F_{sys}}^{'} \times DS > 0 \quad (6)$$

The access time to cached object in memory is negligible, therefore T_{mem} becomes zero. We rewrite equation (1) as below.

$$P = RT - RT^{'} = \frac{DS \times (F_{sys} \times \Delta^{'} - {F_{sys}}^{'} \times \Delta)}{\Delta \times \Delta^{'}} \quad (7)$$

For P in equation (7) to be positive, Δ, $\Delta^{'}$, DS and $F_{sys} \times \Delta^{'} - {F_{sys}}^{'} \times \Delta$ should be also positive. In equation (4) and (6), Δ and $\Delta^{'}$ are greater than zero. DS is positive, because it is the disk service time. Therefore, $F_{sys} \times \Delta^{'} - {F_{sys}}^{'} \times \Delta$ should be greater than zero. In equation (5), $\Delta^{'}$ is $1 - \lambda^{'} \times {F_{sys}}^{'} \times DS$. $F_{sys} \times \Delta^{'} - {F_{sys}}^{'} \times \Delta > 0$ is rewritten as below.

$$\begin{aligned} F_{sys} \quad &- \quad \lambda^{'} \times {F_{sys}}^{'} \times F_{sys} \times DS - {F_{sys}}^{'} \\ &+ \quad \lambda \times {F_{sys}}^{'} \times F_{sys} \times DS > 0 \end{aligned} \quad (8)$$

Web prefetch system processes not only demanded requests but also prefetch requests. Therefore, $\lambda^{'}$ is the sum of λ for demanded requests and λ_{pref} for prefetch requests. We rewrite inequality (8) to (9).

$${F_{sys}}^{'} < \frac{F_{sys}}{1 + \lambda_{pref} \times F_{sys} \times DS} \quad (9)$$

${F_{sys}}^{'}$ is $1-{H_{pref}}^{'}-{H_{buf}}^{'}$, where ${H_{pref}}^{'}$ and ${H_{buf}}^{'}$ are hit rates of prefetch memory and buffer cache, respectively. We assume the cached objects have the same probability to be accessed in the future. Therefore, all pages in buffer cache contribute uniformly to the H_{buf}. When $N(B)$ pages are in buffer cache, each page contributes $\frac{H_{buf}}{N(B)}$ to the cache hit ratio. When more objects are prefetched, ${H_{pref}}^{'}$ increases. ${H_{pref}}^{'}$ is the sum of H_{pref} and H_{pred} where H_{pred} is the prediction value of prefetched files.

$$\begin{aligned} H_{pred} > 1 \quad &- \quad H_{pref} - H_{buf} \times \frac{N(B) - N(R)}{N(B)} \\ &- \quad \frac{F_{sys}}{1 + \lambda_{pref} \times F_{sys} \times DS}, \end{aligned} \quad (10)$$

where $N(B)$ and $N(R)$ are the buffer cache size and the released buffer size for prefetched object, respectively. To maximize the performance improvement, our web predictor monitors the hit rate and size of the buffer cache, and then prefetches web objects which satisfy inequality (10). We get prediction values from *DPS*, while the hit rate and size of buffer cache are obtained using PAPI [32] and *meminfo* in Linux Kernel.

4.3. Memory Aware Request Distribution

There are two groups of nodes in a web cluster system; web distributor and backend nodes. A web distributor forwards incoming requests to backend servers based on its policy. In our simulation, web distribution process uses a locality-based distribution and TCP splicing scheme.

In process-based web servers such as Apache or ISS, there are several idle processes waiting for new requests. When there are no more idle processes, the web server creates a new web process and consumes the memory space which could be used to cache file otherwise. Therefore, too many web processes can decrease the benefit of the buffer cache in the web server. To avoid the skewed distribution, *MARD* checks the number of idle web processes in each of backend servers. Distributor does not forward a new request to those with no more idle process. *MARD* prevents the memory shortage due to too many web processes.

5. Experimental Results

5.1. System Configuration in Simulation

Our simulator is composed of 2 days of web traces, a web analyzer and a web cluster simulator. The *Web analyzer* gets the relation information based on the first day's web traces. Then, the *web cluster simulator* simulates prefetch schemes using the second day's web traces and the information from *web analyzer*.

We design the memory management module in the web cluster system based on the Linux kernel 2.6. When the web server processes access files through the kernel, a number of kernel components work for improving I/O performance. The buffer cache releases the overhead of disks by reducing the number of on-demand I/O requests. The I/O prefetch module reads consecutive blocks in advance, and I/O cluster module reads a cluster of blocks at a time. We design I/O operation in our simulator based on the Linux kernel's I/O prefetch and cluster schemes. LRU scheme is employed as a cache replacement policy in our simulation. The web cluster system in our simulation is configured with one web distributor and four backend servers. Each backend server has its own disk that contains the whole web objects. In addition, the disk simulator, DiskSim [7], is embedded in our simulator for accurate low-level I/O simulation. The simulation system parameters are shown in Table 2.

We measure the size of kernel memory and web server processes while running Apache 2.2 on the Linux kernel 2.6.18. Web cluster distributor and backend servers are connected to the local area network that operates at the speed of 100 Mbps. A web client can create up to 4 connections for delivering web documents, and each backend server maintains 5 to 10 idle web processes for new requests.

Table 2. Simulation Parameters

Backend Server	
Parameter	Value
Physical Memory	512 MB
Kernel Space	100 MB
Web Process Size	20 MB
Disk Latency	Provided by DiskSim [7]
Memory Latency (Hit)	500 Mbps
Network Latency	100 Mbps
Web Distributor	
Splice Hit	160 *us* per packet [31]
Splice Miss	182173 *us* per packet [31]
Session Time	15 Seconds [1]
Web Client	
Concurrent Connection	Up to 4

Table 3 shows real traces from 2 web sites including Department of Computer Science and Engineering in Texas A&M University, ClarkNet and synthetic traces from SPECweb2005 benchmark [34]. Although web clients use HTTP1.0 in the ClarkNet, we assume that they work based on the persistent HTTP.

Table 3. Requests to The Embedded Objects

Name	Day 1	Day 2	HTTP
CS TAMU	25479	20018	HTTP 1.1
ClarkNet	210908	229944	HTTP 1.0
SPECweb2005	6304	116302	HTTP 1.1

5.2. Evaluation Results

We compare the performance of our adaptive prefetch scheme with existing P.P.M. [9, 12] and cluster schemes [10, 28]. Our adaptive scheme uses *DPS* to get the relation information of web objects and *ARC* to dynamically calculate the threshold value for prefetching. P.P.M. and cluster schemes use 6 different static threshold values. For the completeness of our study, we also include results of non-prefetching scheme. In Figure 4, Y axis denotes the response time from web cluster system and X axis represents prefetch schemes. First 12 entries in X axis show results of P.P.M and cluster schemes using 6 static threshold values. Each of them has a character, P or C, followed by a number. P, C and the number stands for P.P.M. scheme, cluster scheme and threshold value which means minimum prediction rate in order to be prefetched, respectively. NO-P and ADA are non-prefetching and our adaptive prefetch schemes. 3 different orders are used for each scheme.

Adaptive prefetch scheme outperforms others by maximum 40 percent. *DPS* has a great advantage in modern

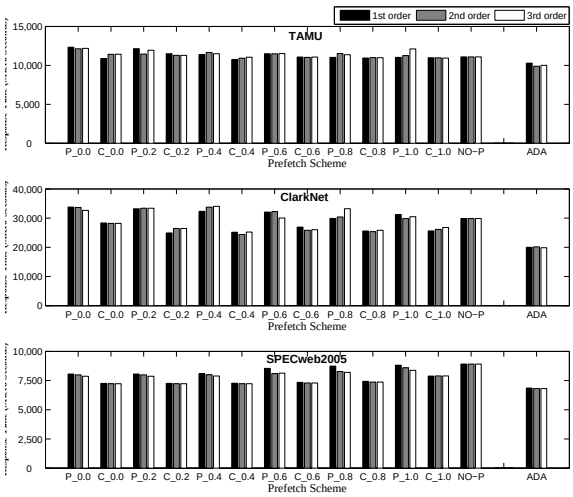

Figure 4. Web Response Time in Prefetch Schemes

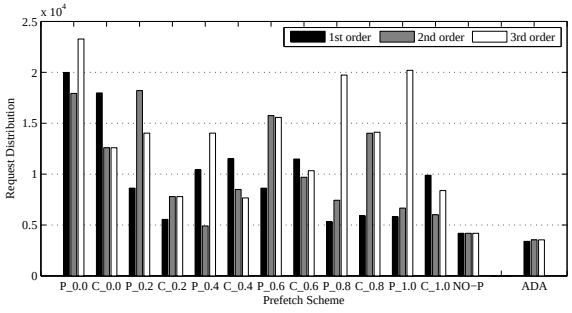

Figure 5. Standard Deviation of Distribution

web framework compared with P.P.M and cluster schemes. It can tolerate the randomness of access pattern caused by web cache or client. *ARC* monitors the hit rate and the size of buffer cache and calculates optimal prefetch threshold dynamically. It makes a decision of prefetching using the dynamic threshold and relation information provided by *DPS*. Since optimal prefetch rate keeps changing over time depending on the system condition, using static threshold values is not a good solution to manage the system properly. Too high prefetch rate drops the performance through the shortage of memory and I/O bandwidth, and too low prefetch rate loses the chance to improve the performance through prefetching.

Although high-order prediction scheme gives more accurate prediction of future requests, performance gap between high and low order is not distinguishable. High-order scheme even shows worse performance than low-order scheme, as in threshold 1 of TAMU or threshold 0.8 and 1 of ClarkNet.

Although locality-based request distribution increases the performance using the locality in processing requests, skewed distribution incurs excessive process creation in some backend nodes and drains them of memory space. *MARD* distributes incoming requests considering the number of idle web processes in each backend node. It enables fair request distribution and efficient memory usage. Figure 5 shows request distribution in backend nodes under ClarkNet workload. X axis represents prefetch schemes and Y axis denotes standard deviation of incoming requests. In the case of aggressive prefetch schemes, they try to prefetch more objects in some of backend nodes, which results in skewed request distribution. However, *MARD* distributes requests almost as fairly as non-prefetch scheme.

6. Conclusions

Rapid growth in the number of web users and current HTTP frameworks makes it difficult to improve the performance through a prefetch scheme. Also, the access pattern of a web server system is not easily predictable because of the web cache mechanism and web object configuration. For proper web prefetching in a cluster environment, we introduce the *DPS* scheme to obtain the relation information of objects and increase the hit rate of prefetched data. Also, we propose the *ARC* scheme to perform an efficient management of prefetch memory in cluster environments. Finally, we suggest the *MARD* to distribute web workload to improve the efficiency in web prefetch.

For evaluation, we implement the prototype of web prefetch engine using PAPI and Apache web server in the Linux. Also, we perform the simulation for verifying the benefit of our scheme in cluster environments. Our experimental results show that our prefetch scheme improves the performance of web cluster system up to 40% in various web workloads. There are three reasons for the improved performance of our scheme. i) Although our prefetch scheme loses the chance to increase the hit rate on the prefetch memory, it avoids the memory saturation and performance degradation caused by an excessive prefetching. ii) Our prefetch scheme provides the adaptive prefetch rate at run time to maximize the prefetch benefit. iii) Our prefetch scheme is adopted to the modern web framework and workloads. We expect that our prefetch scheme can be expanded into the modern web-based applications; e-learning and digital library.

References

[1] Apache Software Foundation. *http://httpd.apache.org/*.

[2] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. Long. Managing flash crowds on the internet. *In Proc. of MOSCOTS*, 2003.

[3] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. *In Proc. of USENIX 2000 Annual Technical Conf.*, 2000.

[4] Z. Ban, Z. Gu, and Y. Jin. An online ppm prediction model for web prefetching. *In Proc. of Web Information and Data Management*, 2007.

[5] J. Borges and M. Levene. Data mining of user navigation patterns. *In Proc. of WebKDD*, pages 92–111, 1999.

[6] C. Bouras, A. Konidaris, and D. Kostoulas. Predictive prefetching on the web and its potential impact in the wide area. *In Proc. of World Wide Web: Internet and Web Information System*, 2003.

[7] J. Bucy and G. Ganger. The disksim simulation environment version 3.0. 2003.

[8] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Model-based clustering and visualization of navigation patterns on a web site. *In Proc. of Data Mining and Knowledge Discovery*, 2003.

[9] X. Chen and X. Zhang. A popularity-based prediction model fore web prefetching. *In Proc. of IEEE Computer*, 2003.

[10] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz. Efficient and adaptive web replication using content clustering. *In Proc. of IEEE Journal on Selected Areas in Communications*, 21:979–994, 2003.

[11] D. Cheng, R. Kannan, S. Vempala, and G. Wang. A divide-and-merge methodology for clustering. *In Proc. of SIG on Management of Data*, 2005.

[12] B. D. Davison. Learning web request patterns. *In Proc. of Web Dynamics: Adapting to Change in Content, Size, Topology and Use*, pages 435–460, 2004.

[13] M. Deshpande and G. Karypis. Selective markove models for predicting web page accesses. *In Proc. of ACM Transactions on Internet Technology*, 4:163–184, 2004.

[14] J. Domenech, A. Pont, J. Sahuquillo, and J. A. Gil. A user-focused evaluation of web prefetching algorithmsa. *in Proc. of the Computer Communications*, 2007.

[15] J. Domenech, J. Sahuquillo, J. A. Gil, and A. Pont. The impact of the web prefetching architecture on the limits of reducing user's perceived latency. *in Proc. of IEEE/WIC/ACM Int'l Conf. on Web Intelligence*, 2006.

[16] S. Drakatos, N. Pissinou, K. Makki, and C. Douligeris. A context-aware prefetching strategy for mobile computing environments. *In Proc. of Int'l Conf' CMC*, pages 1109–1116, 2006.

[17] P. Ferragina and A. Gulli. A personalized search engine based on web snippet hierarchical clustering. *In Proc. of World Wide Web*, 2005.

[18] B. S. Gill and L. A. D. Bathen. Optimal multistream sequential prefetching in a shared cache. *In Proc. of ACM Transactions on Storage*, 3, October.

[19] F. Khalil, J. Li, and H. Wang. Integrating markov model with clustering for predicting web page accesses. *In Proc. of Australasian World Wide Web*, 2007.

[20] D. Kim, N. Adam, V. Alturi, M. Bieber, and Y. Yesha. A clickstream-based collaborative filtering personalization model: Towards a better performans. *In Proc. of WIDM*, 2004.

[21] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. Nps: A non-interfering deployable web prefetching system. *In Proc. of the USENIX Sym. on Internet Technologies and Systems*, 2003.

[22] C. Li and K. Shen. Managing prefetch memory for data-intensive online servers. *In Proc. of USENIX Conf.e on File and Storage Technologies*, 2005.

[23] L. Lu, M. Dunham, and Y. Meng. Discovery of significant usage patterns from clusters of clickstream data. *In Proc. of WebKDD*, 2005.

[24] E. Meneses and O. Rodriguez-Rojas. Using symbolic objects to cluster web documents. *In Proc. of World Wide Web*, 2006.

[25] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *In Proc. of IEEE Transaction on Knowledge and Data Engineering*, 2003.

[26] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *In Proc. of Computer Communication Review*, 26:22–36, 1996.

[27] 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. of ASPLOS*, 1998.

[28] G. Pallis and A. Vakali. Insight and perspectives for content delivery networks. *In Proc. of Communications of the ACM*, 49:101–106, 2006.

[29] C. G. Quinones, C. Madriles, J. Sanchez, P. Marcuello, A. Gonzalez, and D. M. Tullsen. Mitosis compiler: An infrastructure for speculative treading based on pre-computation slices. *In Proc. of PLDI*, 2005.

[30] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. *In Proc. of IEEE Int'l Sym. on Performance Analysis of Systems and Software*, 2003.

[31] M. Rosu and D. Rosu. An evaluation of tcp splice benefits in web proxy servers. *In Proc. of Int'l Conf. on World Wide Web*, 2002.

[32] S. Browne and C. Deane and G. Ho and P. Mucci. Papi: A portable interface to hardware performance counters. 1999.

[33] A. Serbinski and A. Abhari. Improving the delivery of multimedia embedded in web pages. *In Proc. of Int'l Conf. on Multimedia*, pages 779–782, 2007.

[34] SPECweb2005. *http://www.spec.org/web2005/.*

[35] N. Tuah, M. Kumar, and S. Venkatesh. Resource-aware speculative prefetching in wireless networks. *In Proc. of Wireless Networks*, 9:61–72, 2003.

[36] C. Yan, J. Shen, and Q. Peng. Parallel web prefetching on cluster server. *In Proc. of Canadian Conf. on Electrical and Computer Engineering*, 2005.

[37] Z. Zhang, X. M. K. Lee, and Y. Zhou. Pfc: Transparent optimization of existing prefetching strategies for multi-level storage systems. *In Proc. of ICDCS*, 2008.

[38] J. Zhu, J. Hong, and J. G. Hughes. Using markov models for web site link prediction. *In Proc. of Hypertext*, 2002.